



UNIVERSIDADE ESTADUAL PAULISTA
“JÚLIO DE MESQUITA FILHO”
Faculdade de Ciências e Tecnologia
Câmpus de Presidente Prudente

Modelagem computacional de escoamentos não-isotérmicos em meios porosos utilizando o método de elementos finitos com FEniCSx

Mateus Mitsuo Goto Dakuzaku

Orientadora: Profa. Dra. Gilcilene Sanchez de Paulo

Programa: Matemática Aplicada e Computacional

Presidente Prudente, Fevereiro de 2025

UNIVERSIDADE ESTADUAL PAULISTA

Faculdade de Ciências e Tecnologia de Presidente Prudente

Programa de Pós-Graduação em Matemática Aplicada e Computacional

Modelagem computacional de escoamentos não-isotérmicos em meios porosos utilizando o método de elementos finitos com FEniCSx

Mateus Mitsuo Goto Dakuzaku

Orientadora: Profa. Dra. Gilcilene Sanchez de Paulo

Dissertação apresentada ao Programa de Pós-Graduação em Matemática Aplicada e Computacional da Faculdade de Ciências e Tecnologia da UNESP para a obtenção do título de Mestre em Matemática Aplicada e Computacional.

Presidente Prudente, Fevereiro de 2025

D135m Dakuzaku, Mateus Mitsuo Goto
Modelagem computacional de escoamentos não-isotérmicos em meios porosos utilizando o método de elementos finitos com FEniCSx / Mateus Mitsuo Goto Dakuzaku. -- Presidente Prudente, 2025
140 f.

Dissertação (mestrado) - Universidade Estadual Paulista (UNESP), Faculdade de Ciências e Tecnologia, Presidente Prudente
Orientadora: Gilcilene Sanchez de Paulo

1. FEniCSx. 2. Método de elementos finitos. 3. Meios porosos. 4. Dinâmica dos fluidos computacional. I. Título.

CERTIFICADO DE APROVAÇÃO

TÍTULO DA DISSERTAÇÃO: Modelagem computacional de escoamentos não-isotérmicos em meios porosos utilizando o método de elementos finitos com FEniCSx

AUTOR: MATEUS MITSUO GOTO DAKUZAKU

ORIENTADORA: GILCILENE SANCHEZ DE PAULO

Aprovado como parte das exigências para obtenção do Título de Mestre em Matemática Aplicada e Computacional, pela Comissão Examinadora:

Profa. Dra. GILCILENE SANCHEZ DE PAULO (Participação Presencial)
Departamento de Matematica e Computacao / Faculdade de Ciências e Tecnologia de Presidente Prudente

Prof. Dr. FABRICIO SIMEONI DE SOUSA (Participação Virtual)
Matemática Aplicada e Estatística / Universidade de São Paulo (USP)

Profa. Dra. CRISTIANE OLIVEIRA DE FARIA (Participação Virtual)
Análise Matemática / Universidade do Estado do Rio de Janeiro - UERJ

Presidente Prudente, 10 de março de 2025

Assinatura: Fabricio Simeoni de Sousa
Fabricio Simeoni de Sousa (10 de março de 2025 14:03 CDT)

Email: fsimeoni@icmc.usp.br

Assinatura: 

Email: gilcilene.sanchez@unesp.br

Assinatura: 

Email: cofaria@ime.uerj.br

Certificado Mateus

Relatório de auditoria final

2025-03-10

Criado em:	2025-03-10 (Horário Padrão do Uruguai)
Por:	Sistema USPAssina (uspassina@usp.br)
Status:	Assinado
ID da transação:	CBJCHBCAABAApHbvLAoZZHs8kyc9mTPMzDIBLtQh4XPV

Histórico de "Certificado Mateus"

-  Documento criado por Sistema USPAssina (uspassina@usp.br)
2025-03-10 - 16:02:04 ADT
-  Documento enviado por email para fsimeoni@icmc.usp.br para assinatura
2025-03-10 - 16:02:08 ADT
-  Documento enviado por email para gilcilene.sanchez@unesp.br para assinatura
2025-03-10 - 16:02:08 ADT
-  Documento enviado por email para cofaria@ime.uerj.br para assinatura
2025-03-10 - 16:02:08 ADT
-  Email visualizado por cofaria@ime.uerj.br
2025-03-10 - 16:02:57 ADT
-  O signatário fsimeoni@icmc.usp.br inseriu o nome Fabricio Simeoni de Sousa ao assinar
2025-03-10 - 16:03:10 ADT
-  Documento assinado eletronicamente por Fabricio Simeoni de Sousa (fsimeoni@icmc.usp.br)
Signing Link is created by Sistema USPAssina (uspassina@usp.br)
Data da assinatura: 2025-03-10 - 16:03:12 ADT - Fonte da hora: servidor
-  Email visualizado por gilcilene.sanchez@unesp.br
2025-03-10 - 16:03:33 ADT
-  O signatário gilcilene.sanchez@unesp.br inseriu o nome Gilcilene Sanchez de Paulo ao assinar
2025-03-10 - 16:05:08 ADT
-  Documento assinado eletronicamente por Gilcilene Sanchez de Paulo (gilcilene.sanchez@unesp.br)
Data da assinatura: 2025-03-10 - 16:05:10 ADT - Fonte da hora: servidor



Adobe Acrobat Sign



O signatário cofaria@ime.uerj.br inseriu o nome Cristiane ao assinar

2025-03-10 - 16:05:50 ADT



Documento assinado eletronicamente por Cristiane (cofaria@ime.uerj.br)

Data da assinatura: 2025-03-10 - 16:05:52 ADT - Fonte da hora: servidor



Contrato finalizado.

2025-03-10 - 16:05:52 ADT



Adobe Acrobat Sign

Dedico este trabalho aos meus pais e minha esposa, que sempre me apoiaram.

Agradecimentos

Agradeço a minha esposa, Debora, por estar sempre ao meu lado, me apoiando em todos os momentos. Seu incentivo foi essencial nesta jornada.

Aos meus pais, que sempre me encorajaram e apoiaram minhas escolhas, sou profundamente grato. Os valores e ensinamentos que me transmitiram foram fundamentais para que eu pudesse trilhar este caminho.

Minha sincera gratidão à minha orientadora, Profa. Dra. Gilcilene Sanchez de Paulo, por sua orientação valiosa e pelas reuniões enriquecedoras que ampliaram meu conhecimento sobre o tema.

Agradeço aos professores Dr. Fabrício e Dr. Roberto, bem como a todos os participantes das diversas reuniões, pelas valiosas contribuições que fortaleceram meu aprendizado ao longo deste processo.

Sou igualmente grato aos professores e colegas que, de diferentes formas, contribuíram para meu desenvolvimento acadêmico.

Aos meus amigos, pelos momentos de descontração e pelo apoio nos momentos necessários.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001. E apoio do Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), processo nº 130987/2023-9.

Resumo

Este trabalho apresenta uma introdução ao método de elementos finitos, destacando seus conceitos fundamentais e sua aplicação na solução de equações diferenciais parciais (EDPs). São explorados exemplos clássicos, detalhando o desenvolvimento da formulação variacional, a implementação computacional de códigos para obtenção das soluções numéricas e as formas de pós-processamento para sua análise e interpretação. Uma das características do método de elementos finitos (MEF) é dividir o domínio do problema em uma quantidade finita de subdomínios, conhecidos por elementos, gerando assim um domínio discreto. Outro aspecto importante do MEF é trabalhar com a formulação variacional do problema de EDPs, a qual é apresentada neste trabalho utilizando o método de Galerkin contínuo sobre o espaço de Sobolev H^1 , onde a solução aproximada é contínua entre os elementos. Para problemas que exigem conservação de massa e que envolvem duas funções incógnitas, a formulação variacional é apresentada pelo método misto, em que uma função incógnita pertence ao espaço L^2 e a outra ao espaço $H(\text{div})$, por meio de elementos do espaço de Raviart-Thomas para garantir a continuidade dos fluxos normais entre os elementos adjacentes e a conservação de massa global. A implementação computacional é realizada em *Python*TM utilizando a biblioteca FEniCSx, que se destaca por automatizar diversas etapas do processo numérico, facilitando a implementação e agilizando a obtenção de resultados. A dissertação aborda a estrutura e o funcionamento desta ferramenta, proporcionando uma visão detalhada sobre sua aplicação prática. Na etapa de pós-processamento, as respostas obtidas são analisadas tanto por meio de representações gráficas ou por tabelas quanto por testes de convergência, que são importantes para garantir a confiabilidade dos resultados e sua coerência com a teoria. A aplicação principal deste trabalho é resolver um problema de transporte passivo da temperatura em meios porosos, modelando a interação entre o escoamento de um fluido e a condutividade térmica. Esta dissertação de Mestrado preocupa-se, também, em manter um nível de detalhamento dos assuntos com o objetivo de se tornar um material didático acessível a pesquisadores e estudantes interessados na introdução e implementação do método de elementos finitos.

Palavras-Chave: *FEniCSx, Método de Elementos Finitos, Meios Porosos, Dinâmica dos Fluidos Computacional.*

Abstract

This work provides an introduction to the finite element method, highlighting its fundamental concepts and its application in solving partial differential equations (PDEs). Classic examples are explored, detailing the development of the variational formulation, the computational implementation of codes to obtain numerical solutions, and the post-processing methods for their analysis and interpretation. One of the key characteristics of the finite element method (FEM) is dividing the problem domain into a finite number of subdomains, known as elements, thereby generating a discrete domain. Another important aspect of FEM is working with the variational formulation of the PDE problem, which is presented in this work using the continuous Galerkin method on the Sobolev space H^1 , where the approximate solution is continuous between elements. For problems requiring mass conservation and involving two unknown functions, the variational formulation is presented through the mixed method, where one unknown function belongs to the L^2 space and the other to the $H(\text{div})$ space, using elements from the Raviart-Thomas space to ensure the continuity of normal fluxes between adjacent elements and global mass conservation. The computational implementation is carried out in *Python*TM using the FEniCSx library, which stands out for automating various stages of the numerical process, facilitating implementation and speeding up result generation. The dissertation addresses the structure and functioning of this tool, providing a detailed overview of its practical application. In the post-processing phase, the obtained results are analyzed both through graphical representations and tables, as well as convergence tests, which are essential to ensure the reliability of the results and their consistency with theory. The main application of this work is solving a passive temperature transport problem in porous media, modeling the interaction between fluid flow and thermal conductivity. This Master's dissertation also aims to maintain a level of detail in the topics to serve as an accessible teaching material for researchers and students interested in the introduction and implementation of the finite element method.

Keywords: *FEniCSx, Finite Element Method, Porous Media, Computational Fluid Dynamics.*

Lista de Figuras

2.1	Discretização do domínio.	26
2.2	Representação de u_h como a projeção ortogonal de u em V_h	36
2.3	Domínio Discretizado	37
2.4	Função base $\phi_1(x)$	38
2.5	Função base $\phi_2(x)$	39
2.6	Função base $\phi_3(x)$	39
2.7	Função base $\phi_4(x)$	39
2.8	Esboço da Solução Aproximada u_h e da Analítica $u(x)$ do problema (2.12) com condição de contorno do tipo Dirichlet na fronteira de Ω	41
2.9	Transformação T_e através do elemento mestre.	42
2.10	Representação do elemento mestre e o mapeamento T_e para o domínio 1D discretizado com n elementos.	43
2.11	Formato da matriz $\mathbf{A}$	45
3.1	Distribuição da velocidade utilizando as bibliotecas NumPy & SymPy.	54
3.2	Distribuição da velocidade utilizando as bibliotecas Firedrake.	56
3.3	Distribuição da velocidade utilizando as bibliotecas FEniCSx.	59
3.4	Distribuição da velocidade ao longo de Ω	61
3.5	Ordem de convergência com aproximação polinomial de grau 1.	63
3.6	Convergência com aproximação polinomial de grau 2.	64
3.7	Convergência com aproximação polinomial de grau 3.	65
3.8	Análise de convergência da solução (P_1).	66
3.9	Distribuição da velocidade no espaço P_1 ao longo de Ω , com malha não estruturada.	68
3.10	Distribuição da velocidade no espaço P_1 , com malha não estruturada.	69
3.11	Decaimento da norma do erro absoluto em função de h , na escala log-log.	71
3.12	Distribuição da velocidade utilizando uma malha não estruturada.	72
3.13	Distribuição da velocidade (solução aproximada) sobre a geometria de um paralelepípedo.	74
3.14	Distribuição da velocidade (solução analítica) sobre a geometria de um paralelepípedo.	74
3.15	Distribuição da velocidade (solução aproximada) sobre uma geometria cilíndrica.	76
3.16	Distribuição da velocidade (solução analítica) sobre uma geometria cilíndrica.	77
3.17	Distribuição da temperatura utilizando o método implícito em diferentes tempos.	80
3.18	Decaimento da norma do erro dos métodos explícito e implícito em relação ao espaçamento h	81
3.19	Decaimento da norma do erro dos métodos explícito e implícito em relação ao espaçamento Δt	82

3.20	Domínio do tipo <i>a quarter of the five-spot</i> . Figura extraída do livro [1].	83
3.21	Campo da pressão utilizando o método implícito em diferentes tempos.	84
3.22	Evolução térmica no espaço P_1 , utilizando o método implícito.	86
4.1	Representação dos graus de liberdade em RT_0	90
4.2	Representação das Funções base de RT_0 em Ω_e . Figura extraída do livro [2].	90
4.3	Distribuição da magnitude da velocidade.	94
4.4	Campo de pressão.	95
5.1	Representação do domínio Ω	100
5.2	Campo de permeabilidade da camada 36 do projeto SPE 10 (disponível em [3]).	100
5.3	Campo da pressão.	102
5.4	Distribuição da magnitude da velocidade em escala logarítmica.	102
5.5	Aproximação da temperatura no tempo $t = 11,8976$ s.	103
5.6	Aproximação da temperatura no tempo $t = 34,7776$ s.	103
5.7	Aproximação da temperatura no tempo $t = 103,4176$ s.	103
5.8	Aproximação da temperatura no tempo $t = 252,1376$ s.	103

Lista de Tabelas

3.1	Relação da norma do erro absoluto com o aumento do grau polinomial. . .	62
3.2	Norma do erro para P_1	62
3.3	Norma do erro para P_2	63
3.4	Norma do erro para P_3	64
3.5	Convergência do problema (3.6).	70
3.6	Análise de convergência do problema 3D com geometria de um paralelepípedo.	75
3.7	Análise de convergência do problema 3D com geometria cilíndrica.	77
3.8	Análise de convergência do problema envolvendo a equação do calor.	81
3.9	Análise de convergência do problema modelado pela equação de advecção-difusão.	86
4.1	Convergência do problema misto.	95

Sumário

Resumo	11
Abstract	13
Lista de Figuras	14
Lista de Tabelas	16
1 Introdução	21
1.1 Objetivos	22
1.2 Justificativa	23
1.3 Estrutura da dissertação	23
2 Conceitos do Método de Elementos Finitos	25
2.1 Malha e elementos finitos	26
2.2 Espaço de funções	26
2.3 Espaços e Subespaços de Sobolev	28
2.4 Métodos variacionais	29
2.5 Formulação Variacional	30
2.6 Formulação Variacional Unidimensional	32
2.7 Método de Galerkin	33
2.8 Problema bem posto	34
2.9 Melhor aproximação e Ortogonalidade de Galerkin	36
2.10 Exemplo: Problema de Poisson 1D	36
2.11 Elemento Mestre	42
2.12 Montagem da matriz (<i>Assembly</i>)	43
2.13 Estimativa do erro	45
2.14 Tratamento da evolução temporal	46
2.14.1 Equação do calor	46
2.14.2 Estabilidade numérica	49
2.14.3 Equação de Advecção-Difusão	49
3 Resultados Numéricos Utilizando FEniCSx	51
3.1 Equação de Poisson	51
3.1.1 Problemas 1D	51
3.1.2 Problemas 2D	60
3.1.3 Problemas 3D	73
3.2 Equação do calor	78
3.3 Problema five-spot simplificado	82
3.4 Equação de Advecção-Difusão	85

4	Escoamento Monofásico em Meios Porosos	89
4.1	Espaços $H(\operatorname{div}; \Omega)$	89
4.2	Espaço $RT_k(\Omega_e)$	89
4.3	Condição LBB	91
4.4	Formulação Variacional do problema misto	91
4.5	Aproximação numérica e sistema linear	92
4.6	Simulação do escoamento em um campo homogêneo	93
5	Transporte Passivo da Temperatura em Meios Porosos	97
5.1	Descrição do problema	97
5.2	Características do problema	98
5.3	Metodologia de resolução do problema	101
5.4	Resultados numéricos e discussões	102
6	Conclusão e Estudos Futuros	105
	Referências	106
A	Códigos gerados em FEniCSx	111
A.1	Problema 2D da subseção 3.1.2.1	111
A.2	Problema 2D da subseção 3.1.2.2	112
A.3	Problema 2D com malha não estruturada da subseção 3.1.2.3	114
A.4	Primeiro problema com domínio irregular da subseção 3.1.2.4.1	116
A.5	Segundo problema com domínio irregular da subseção 3.1.2.4.2	118
A.6	Problema 3D num tubo paralelepípedo da seção 3.13	121
A.7	Problema 3D num tubo cilíndrico da seção 3.1.3.2	122
A.8	Problema de Poisson da seção 3.2	124
A.9	Problema five-spot da seção 3.3	127
A.10	Problema de Advecção-Difusão da seção 3.4	129
A.11	Simulação do escoamento em um campo homogêneo da seção 4.6	131
A.12	Problema de Transporte Passivo da Temperatura em Meios Porosos do capítulo 5	135

Introdução

A modelagem matemática de fenômenos naturais é uma ferramenta essencial para a compreensão e simulação de processos físicos, como escoamento de fluidos, variação de temperatura, propagação de umidade, direção dos ventos e previsão do tempo. Em muitos casos, a complexidade do sistema ou a inexistência de uma solução analítica exata exige a adoção de abordagens numéricas para obter aproximações viáveis [4].

Entre as diversas abordagens numéricas disponíveis, destacam-se métodos como diferenças finitas, volumes finitos e elementos finitos. A escolha do método mais adequado depende das características do problema a ser resolvido. O método de elementos finitos, em particular, tem sido amplamente aplicado devido à sua flexibilidade na representação de domínios complexos e na formulação de problemas envolvendo equações diferenciais parciais. Ele é especialmente útil para simulações em áreas como engenharias, mecânica dos fluidos e geociências [5; 6].

A modelagem computacional também desempenha um papel fundamental na indústria petrolífera, auxiliando na tomada de decisões e na otimização dos processos de extração. Simulações numéricas permitem avaliar a eficiência de estratégias de recuperação de petróleo, como a injeção de água ou gás para manter a pressão no reservatório e melhorar as taxas de produção. Além disso, esses modelos possibilitam a análise do escoamento em meios altamente heterogêneos, combinando efeitos químicos e térmicos da extração [1].

A modelagem da transferência de calor em meios porosos ocorre em diversos contextos industriais e naturais. A formulação matemática desses problemas envolve a combinação da equação da conservação de energia com o escoamento de fluidos, permitindo a simulação de processos térmicos em materiais porosos [7].

Para a implementação computacional desse tipo de problema, diversas ferramentas numéricas tem sido desenvolvidas para facilitar a discretização e o processo de resolução de equações diferenciais parciais. O FEniCS legacy [8] e o Firedrake [9] são dois pacotes amplamente utilizados para a implementação do método de elementos finitos, ambos permitindo a formulação variacional de problemas e sua resolução de forma eficiente. Mais recentemente, o FEniCSx surgiu como uma modernização do FEniCS legacy, trazendo melhorias na estrutura do código, maior flexibilidade e maior integração com bibliotecas externas. Ele é composto pelas ferramentas: UFL, Basix, FFCx e DOLFINx [10; 11; 12; 13] e os algoritmos são projetados com o objetivo de trabalhar com cálculos massivos em paralelo [13].

Assim, o FEniCSx se destaca como uma ferramenta robusta e moderna para a implementação do método de elementos finitos, permitindo a resolução eficiente de problemas complexos em diversas áreas da engenharia e ciências aplicadas. No contexto desta dissertação, sua utilização se mostra particularmente vantajosa para a modelagem de problemas

envolvendo transporte passivo de temperatura em meios porosos heterogêneos. Neste caso, o termo ‘passivo’ se refere ao fato de que a temperatura depende da velocidade, enquanto a velocidade não é influenciada pela temperatura. A seguir, são apresentados os objetivos deste trabalho, bem como a justificativa para a escolha do problema abordado.

1.1 Objetivos

Um dos objetivos deste trabalho é utilizar a ferramenta computacional FEniCSx, em conjunto com o método de elementos finitos, para calcular a solução aproximada de um problema envolvendo o transporte passivo de temperatura em meios porosos. Este problema engloba a Lei de Darcy, a equação da conservação de massa e a equação da energia, representando uma transferência de calor convectiva associada ao movimento de um fluido. Tal transferência combina os mecanismos de advecção e difusão [7].

O problema estudado envolve um meio poroso com permeabilidade heterogênea, composto por uma fase sólida e outra líquida. As variáveis associadas a cada fase são descritas pelas seguintes equações diferenciais. A fase sólida é dada por:

$$(1 - \varphi)\tilde{\rho}\tilde{c}_p \frac{\partial \tilde{T}}{\partial t} = (1 - \varphi)\nabla \cdot (\tilde{\kappa}\nabla \tilde{T}) - ah(\tilde{T} - T), \quad (1.1)$$

e a fase do fluido é dada por:

$$\varphi\rho c_p \frac{\partial T}{\partial t} + \rho c_p \nabla T \cdot \mathbf{v} = \varphi\nabla \cdot (\kappa\nabla T) + ah(\tilde{T} - T), \quad (1.2)$$

onde as variáveis com til estão associadas à fase sólida (rocha) e sem til à fase fluida. ρ representa a densidade; c_p a capacidade de calor; T a temperatura; t o tempo; κ a condutividade termal; $\mathbf{v}$ o fluxo de volume chamado de velocidade Darcy; a é a área de superfície específica; h é o coeficiente de transferência de calor e φ a porosidade efetiva que é dada por:

$$\varphi = \frac{V_p}{V_p + V_s},$$

onde V_p representa o volume dos poros que contribuem para o escoamento do fluido e V_m o volume de poros isolados e de regiões sem saída.

Este trabalho busca, além da solução do problema principal, alcançar os seguintes objetivos:

- Servir como um produto didático para aqueles interessados em aprender sobre o método de elementos finitos, oferecendo uma introdução clara e progressiva dos conceitos fundamentais.
- Apresentar e explorar a ferramenta FEniCSx, destacando sua aplicação em problemas variados de equações diferenciais parciais. Para isso, são discutidos problemas clássicos como a Equação de Poisson, a Equação do calor e a Equação de advecção-difusão, cujas implementações são disponibilizadas no Apêndice para referência prática.

Com isso, esta dissertação tem o propósito de contribuir tanto para o estudo específico do transporte de temperatura em meios porosos com permeabilidade heterogênea, quanto para a disseminação do conhecimento sobre o método de elementos finitos e a utilização da ferramenta FEniCSx em aplicações práticas.

1.2 Justificativa

O método de elementos finitos é amplamente reconhecido como uma ferramenta poderosa para a solução numérica de equações diferenciais parciais, inclusive aquelas sem solução analítica. Uma de suas principais vantagens é que a solução obtida não se limita a valores discretos em pontos específicos, mas é representada como uma função interpolada em todo o domínio. Isso permite avaliar o valor da solução em qualquer ponto do domínio. Além disso, o método é uma ótima opção para lidar com geometrias complexas.

No entanto, sua implementação direta pode ser trabalhosa e exigir um nível avançado de conhecimento matemático e de programação. Para superar esse desafio, ferramentas computacionais como o FEniCSx desempenham um papel essencial, automatizando grande parte do processo de implementação. O FEniCSx economiza tempo do usuário ao fornecer funcionalidades prontas, otimizações que reduzem o tempo computacional e suporte para computações massivas em paralelo, sendo adequado para problemas de alta complexidade e inclusive tem sido utilizado para resolver problemas com mais de 1 trilhão de células [13].

Além disso, o FEniCSx oferece exemplos prontos [14; 15; 16] de implementação e conta com uma comunidade ativa de desenvolvedores, o que facilita sua adoção por novos usuários e garante um contínuo aprimoramento da ferramenta. Apesar da existência de tutoriais e exemplos sobre o método de elementos finitos e o uso do FEniCSx, este trabalho busca contribuir ao reunir e estruturar uma sequência progressiva de problemas resolvidos. O objetivo é fornecer uma abordagem didática que integre os fundamentos teóricos do método com aplicações práticas no FEniCSx, servindo como um recurso adicional para estudantes e pesquisadores que desejam se aprofundar na área.

Por fim, ao aplicar o MEF ao estudo do transporte passivo de temperatura em meios porosos com permeabilidade heterogênea, este trabalho explora um problema relevante e desafiador, demonstrando o potencial da ferramenta em modelar fenômenos complexos. A justificativa está, portanto, na importância de contribuir com a compreensão de problemas físicos reais e, ao mesmo tempo, fornecer um material didático e acessível para futuros estudos e aplicações.

1.3 Estrutura da dissertação

Esta dissertação está organizada em seis capítulos, incluindo a Introdução apresentada neste primeiro. O Capítulo 2 explora os fundamentos do método de elementos finitos. Nele são apresentados conceitos essenciais para garantir uma compreensão aprofundada da teoria e viabilizar sua aplicação na solução de problemas diversos. Em seguida, os estudos de casos estão no Capítulo 3, onde são discutidas as resoluções de problemas pelo método de Galerkin contínuo no espaço $H^1(\Omega)$ com a ferramenta computacional FEniCSx. Este tópico inclui implementações realizadas em linguagem PythonTM no Google Colab, acompanhadas de análises dos resultados obtidos. O Capítulo 4 introduz problemas mistos, com foco nos espaços $H(\text{div}; \Omega)$ e $RT_k(\Omega_e)$. Além disso, aborda a formulação matemática e o método de resolução utilizado neste tipo de abordagem. Já o Capítulo 5 concentra-se no problema principal, apresentando a modelagem matemática detalhada por meio das equações governantes e as características físicas envolvidas, além dos resultados numéricos e suas análises. O desfecho deste trabalho vem com a conclusão no Capítulo 6, para sintetizar as principais contribuições da pesquisa e apresentar as possibilidades para estudos futuros.

Conceitos do Método de Elementos Finitos

O método de elementos finitos é uma técnica computacional amplamente utilizada para a solução de equações diferenciais, com aplicações que abrangem diversas áreas da ciência e engenharia. Trata-se de uma generalização de métodos variacionais clássicos e residuais, cujo objetivo é encontrar uma solução aproximada para a equação diferencial em estudo. Essa solução pertence a um espaço de funções de dimensão finita e é obtida por meio de uma combinação linear de funções base, selecionadas de forma adequada para atender às características do problema. Um aspecto atrativo desse método é que essas funções base são definidas de maneira a serem não nulas apenas em regiões limitadas do domínio, o que confere eficiência ao processo de cálculo.

Uma das grandes vantagens do método de elementos finitos é sua capacidade de lidar com problemas definidos em regiões com geometrias complexas. Para essas situações, torna-se difícil determinar funções de aproximação que satisfaçam as condições de contorno. A estratégia adotada por este método é subdividir o domínio em um conjunto de pequenos elementos finitos, que podem ter diferentes formas geométricas. Essa abordagem facilita a resolução de problemas em domínios irregulares, ampliando consideravelmente o alcance e a aplicabilidade do método.

As etapas fundamentais para a aplicação do método de elementos finitos são descritas a seguir:

- Discretização do domínio em um conjunto de elementos finitos (malha);
- Formulação variacional da equação diferencial;
- Aplicação de um método de aproximação, como o método de Galerkin;
- Montagem do sistema de equações a partir dos elementos finitos;
- Inclusão das condições de contorno;
- Resolução do sistema linear, obtendo-se uma função que aproxima a solução do problema original.

As subseções a seguir detalham esses procedimentos, além de apresentar outros fundamentos necessários para a compreensão do método. Para uma introdução mais detalhada ao método e suas bases teóricas, recomenda-se a consulta aos livros [5; 6; 17].

2.1 Malha e elementos finitos

O domínio precisa ser discretizado em uma quantidade finita de elementos. Essa discretização pode ser ajustada de acordo com as especificidades do problema ou o nível de precisão desejado. Há diversos tipos de elementos que podem compor a malha. No caso unidimensional, cada elemento é um subintervalo do domínio. Para problemas 2D os elementos mais comuns são triângulos e quadrados, enquanto em três dimensões, os tetraedros e cubos são frequentemente utilizados. A Figura 2.1 apresenta um exemplo bidimensional em que o domínio Ω é dividido em elementos triangulares Ω_e . As notações Γ e Γ_e representam, respectivamente, a fronteira do domínio Ω e a fronteira do elemento Ω_e .

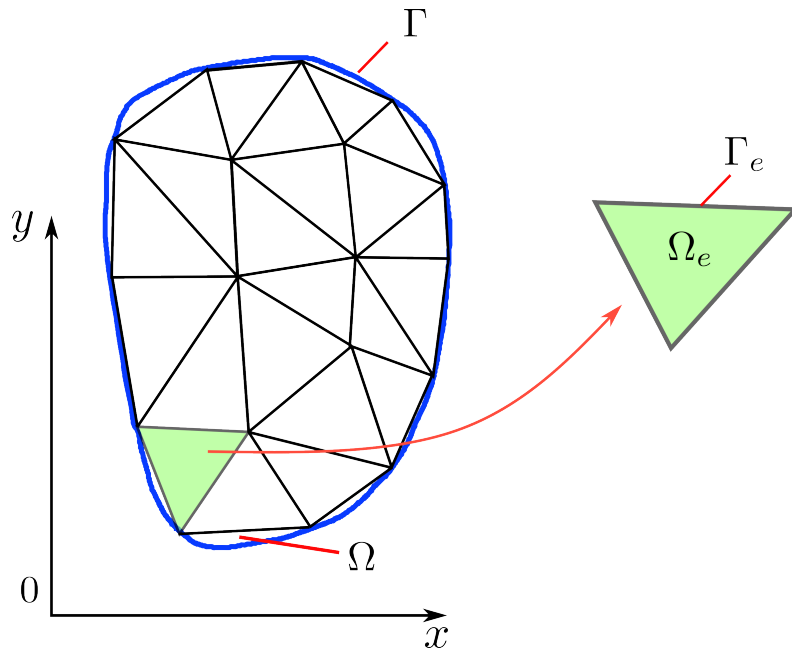


Figura 2.1: Discretização do domínio.

2.2 Espaço de funções

Uma das etapas do método de elementos finitos é a formulação variacional de equações diferenciais, que utiliza o espaço de funções. Como esses espaços tem dimensão infinita, é necessário compreender conceitos básicos de análise funcional para estudar o método de forma adequada. Em particular, a formulação é desenvolvida em subespaços de dimensão finita, subespaços dos espaços de Sobolev, que são adequados para representar as funções de forma discreta.

O conceito de dimensionalidade é muito importante neste estudo. Um espaço vetorial é dito n -dimensional se pudermos encontrar n elementos linearmente independentes, mas $n + 1$ elementos são linearmente dependentes. Agora suponha que seja possível encontrar n elementos linearmente independentes para todo n , então o espaço é dito infinitamente dimensional.

Para estudar os espaços vetoriais compostos por funções, é preciso estudar os conceitos de *continuidade*, *diferenciabilidade* e *integrabilidade* [18].

O espaço das funções Reais contínuas em $\Omega \subset \mathbb{R}^n$ é dado por:

$$C^0(\Omega) = \{f : \Omega \rightarrow \mathbb{R} : f \text{ é contínua, } \forall x \in \Omega\}.$$

Para funções contínuas que possuem derivadas contínuas até a ordem 2, temos:

$$C^2(\Omega) = \{f : \Omega \rightarrow \mathbb{R} : f \text{ tem derivadas contínuas até a ordem 2, } \forall x \in \Omega\}.$$

De forma geral, o espaço $C^k(\Omega)$ é o conjunto de todas as funções de valores reais que são contínuas e cujas derivadas parciais até a ordem k também são contínuas em Ω .

No contexto da integrabilidade, um exemplo importante é o espaço das funções quadrado integrável escrito da forma:

$$L^2(\Omega) = \{f : \Omega \subset \mathbb{R}^n \rightarrow \mathbb{R} : \int_{\Omega} f^2 dx < +\infty\}.$$

Para funções quadrado integrável, cujas derivadas também sejam quadrado integrável, o espaço delas é dado por:

$$H^1(\Omega) = \{f : \Omega \subset \mathbb{R}^n \rightarrow \mathbb{R} : f \in L^2(\Omega) \text{ e } \int_{\Omega} (\nabla f \cdot \nabla f) dx < +\infty\}.$$

Um subespaço de $H^1(\Omega)$, composto por funções, cujo valor na fronteira Γ seja igual a zero, é representado abaixo.

$$H_0^1(\Omega) = \{f \in H^1(\Omega) : f|_{\Gamma_D} = 0\},$$

onde a notação Γ_D representa a parte da fronteira Γ com condição de contorno do tipo Dirichlet e $f|_{\Gamma_D}$ é usada para denotar que a função f é restrita à fronteira de Ω (Γ_D), ou seja, os valores de $f(x)$, $\forall x \in \Gamma_D$.

Outro espaço importante que será muito utilizado é o espaço das funções polinomiais. Considere o intervalo (a,b), o espaço das funções polinomiais de grau no máximo k é definido da seguinte maneira:

$$P_k(a, b) = \{v, v = \sum_{i=0}^k \alpha_i x^i\}.$$

Por exemplo: seja $k = 1$, temos o espaço de funções polinomiais de grau no máximo 1, que pode ser representado da forma:

$$P_1(a, b) = \{v, v = \alpha_0 + \alpha_1 x\}.$$

Para a análise das funções nos espaços considerados, utilizaremos as normas que quantificam o tamanho (magnitude) das funções e suas derivadas, permitindo medir a aproximação e a convergência das soluções. Alguns exemplos são definidos abaixo:

$$\|f\|_{L^2(\Omega)} = \left(\int_{\Omega} [f(x)]^2 dx \right)^{\frac{1}{2}}, \quad (2.1)$$

$$\|f\|_{L^\infty(\Omega)} = \sup \{ |f(x)| : x \in \Omega \}, \quad (2.2)$$

$$\|f\|_{H^1(\Omega)} = \left(\int_{\Omega} [(f(x))^2 + \nabla f(x) \cdot \nabla f(x)] dx \right)^{\frac{1}{2}}, \quad (2.3)$$

onde $\Omega \subset \mathbb{R}^n$.

Uma completa referência desta seção pode ser encontrada em [18; 19; 20; 21].

2.3 Espaços e Subespaços de Sobolev

Na teoria das equações diferenciais parciais, os espaços de Banach e Hilbert desempenham um papel fundamental. Eles oferecem as bases matemáticas necessárias para a formulação variacional do problema. A seguir, são apresentadas suas definições [18].

Definição 1 *Um espaço de Banach V é um espaço vetorial munido de uma norma $\|\cdot\|_V$, que também é completo com respeito à esta norma, ou seja, todas as sequências de Cauchy em V convergem para um elemento $v \in V$.*

Definição 2 *Uma sequência dos elementos $v_1, v_2, v_3, \dots$ é chamada de sequência de Cauchy (ou sequência fundamental) se para todo $\epsilon > 0$, existe um inteiro positivo N tal que para todo $m, n > N$, $\|v_n - v_m\| < \epsilon$.*

Um espaço muito importante é o espaço de Hilbert que é também um espaço de Banach (pois pode-se sempre definir uma norma induzida pelo produto interno) e tem a seguinte definição:

Definição 3 *Um espaço de Hilbert é um espaço vetorial equipado com um produto interno que é também um espaço métrico completo com respeito a norma induzida pelo produto interno.*

Os produtos internos $L^2(\Omega)$ e $H^1(\Omega)$ são exemplos importantes definidos abaixo.

$$\langle v, w \rangle_{L^2(\Omega)} = \int_{\Omega} v w dx,$$

$$\langle v, w \rangle_{H^1(\Omega)} = \int_{\Omega} (v w + \nabla v \cdot \nabla w) dx.$$

A norma de Lebesgue L^p para $1 \leq p < \infty$ é dada por:

$$\|f\|_{L^p(\Omega)} = \left(\int_{\Omega} |f(x)|^p dx \right)^{\frac{1}{p}}.$$

E para o caso L^∞ tem-se:

$$\|f\|_{L^\infty(\Omega)} = \sup\{|f(x)| : x \in \Omega\}.$$

Então, o espaço de Lebesgue é definido da forma:

$$L^p(\Omega) = \{f : \Omega \rightarrow \mathbb{R} : \|f\|_{L^p(\Omega)} < +\infty\}.$$

A seguir, mais alguns outros conceitos importantes para a definição dos espaços de Sobolev.

O espaço $\mathcal{D}(\Omega)$ é definido como:

$$\mathcal{D}(\Omega) = \{\phi \in C^\infty(\Omega) : \phi \text{ tem um suporte compacto em } \Omega\},$$

em que ϕ ter suporte compacto significa que esta função é não nula apenas numa região limitada e fechada (compacta).

Definição 4 (*Derivada fraca*): Uma dada função $f : \Omega \rightarrow \mathbb{R}$ é dita ter uma derivada fraca com respeito a x_i , $D_w^i f$, se existir uma função g tal que:

$$\int_{\Omega} g(x) \phi(x) dx = - \int_{\Omega} f(x) \frac{\partial \phi}{\partial x_i}(x) dx, \quad \forall \phi \in \mathcal{D}(\Omega). \quad (2.4)$$

Se g existe, nós definimos $D_w^i f = \frac{\partial f}{\partial x_i} = g$.

Esta definição pode ser generalizada para derivadas de ordem superior. Dada uma função $f : \Omega \rightarrow \mathbb{R}$ e $\vec{\alpha} = (\alpha_1, \dots, \alpha_n) \in \mathbb{N}^n$, $|\vec{\alpha}| = \alpha_1 + \dots + \alpha_n$, então $D_w^{\vec{\alpha}} f = \frac{\partial^{|\vec{\alpha}|} f}{\partial x_1^{\alpha_1} \dots \partial x_n^{\alpha_n}}$ é a derivada fraca de f de ordem $|\vec{\alpha}|$, se existe uma função g tal que

$$\int_{\Omega} g(x) \phi(x) dx = (-1)^{|\vec{\alpha}|} \int_{\Omega} f(x) \phi^{(|\vec{\alpha}|)}(x) dx \quad \forall \phi \in \mathcal{D}(\Omega),$$

onde $(|\vec{\alpha}|)$ representa a derivada de ordem $|\vec{\alpha}|$ de ϕ .

A partir dos fundamentos básicos descritos nesta seção, já é possível definir espaços de funções de $L^p(\Omega)$ em que cada função admite derivadas fracas de diversas ordens pertencentes também a $L^p(\Omega)$.

Definição 5 (*espaço de Sobolev*): Sejam $m \in \mathbb{N}$, $1 \leq p \leq \infty$ e $\Omega \subset \mathbb{R}^n$. O subespaço vetorial de $L^p(\Omega)$

$$W^{m,p}(\Omega) = \{f : \Omega \rightarrow \mathbb{R} : f \in L^p(\Omega), D_w^{\vec{\alpha}} f \in L^p(\Omega) \quad \forall \vec{\alpha} \in \mathbb{N}^n, |\vec{\alpha}| \leq m\}, \quad (2.5)$$

é um espaço de Sobolev.

E a norma apropriada:

$$\|f\|_{W^{m,p}(\Omega)} = \left(\sum_{|\vec{\alpha}| \leq m} \|D_w^{\vec{\alpha}} f\|_{L^p(\Omega)}^p \right)^{\frac{1}{p}}$$

Casos especiais do espaço de Sobolev incluem:

- Para $m = 0$, tem-se $W^{0,p}(\Omega) = L^p(\Omega)$, que corresponde ao espaço de Lebesgue de ordem p . Em particular, $W^{0,2}(\Omega) = L^2(\Omega)$ é o espaço das funções quadrado integrável.
- Para $p = 2$, o espaço $W^{m,2}(\Omega) = H^m(\Omega)$ é um espaço de Hilbert. Como caso particular, $W^{1,2}(\Omega) = H^1(\Omega)$ é o espaço das funções quadrado integrável, cujas derivadas de primeira ordem também pertençam a $L^2(\Omega)$.

Um estudo mais abrangente sobre estes espaços podem ser visto nas referências [18; 20].

2.4 Métodos variacionais

Os métodos variacionais buscam encontrar uma solução aproximada u_h em um espaço de funções apropriado e de dimensão finita. Essa solução é interpretada como uma projeção da solução exata nesse subespaço e pode ser expressa como uma combinação linear de

funções base pertencentes a esse espaço. O processo de discretização resulta em um sistema de equações, no qual as incógnitas correspondem aos coeficientes dessa combinação linear [17].

Dentre os métodos variacionais, destaca-se o método de elementos finitos, amplamente utilizado devido a várias vantagens: as funções base possuem suporte compacto, ou seja, são não nulas apenas em uma pequena região do domínio, o que simplifica os cálculos; o método lida de forma natural com geometrias complexas; e sua formulação é robusta e eficiente.

Supondo que a solução aproximada seja escrita como:

$$u_h(x) = \sum_{i=1}^N c_i \phi_i,$$

onde os coeficientes c_i são as incógnitas e as ϕ_i as funções bases. A simples substituição desta aproximação na equação diferencial não garante a obtenção de N equações algébricas linearmente independentes. Para superar essa limitação, o problema é reformulado em sua forma fraca. Essa reformulação, baseada em integrais por partes, reduz as restrições impostas à função aproximada u_h , tornando possível reduzir a ordem das derivadas presentes na equação. Em um problema unidimensional, por exemplo, uma equação originalmente dependente da segunda derivada passa a depender apenas da primeira derivada, exigindo apenas que u_h satisfaça $\frac{du_h}{dx} \in L^2(\Omega)$. Esse princípio se aplica de maneira análoga a problemas em duas ou três dimensões.

2.5 Formulação Variacional

Considere o problema modelado pela equação de Poisson, com domínio $\Omega \subset \mathbb{R}^d$, $d = 1, 2$ ou 3 , expresso como:

$$\begin{cases} -\nabla \cdot (\mu(\mathbf{x}) \nabla u(\mathbf{x})) = f(\mathbf{x}) & , \mathbf{x} \in \Omega, \\ u(\mathbf{x}) = g(\mathbf{x}) & , \mathbf{x} \in \Gamma_D, \\ -\mu(\mathbf{x}) \nabla u(\mathbf{x}) \cdot \mathbf{n} = h(\mathbf{x}) & , \mathbf{x} \in \Gamma_N, \end{cases} \quad (2.6)$$

onde $f : \Omega \rightarrow \mathbb{R}$ representa o termo fonte, $g : \Gamma_D \rightarrow \mathbb{R}$ e $h : \Gamma_N \rightarrow \mathbb{R}$ são funções conhecidas e descrevem, respectivamente, as condições de contorno de Dirichlet sobre Γ_D e de Neumann sobre Γ_N , $\mu : \Omega \rightarrow \mathbb{R}$ é a função coeficiente de difusão, que dependendo da aplicação pode representar condutividade térmica, viscosidade ou permeabilidade, por exemplo; e $u : \Omega \rightarrow \mathbb{R}$ é a função incógnita.

A formulação variacional do problema (2.6) pode ser obtida conforme descrito em [18]. Para tanto, multiplica-se a equação diferencial por uma função peso $v(\mathbf{x})$, definida em Ω , tal que $v(\mathbf{x}) = 0$, $\forall \mathbf{x} \in \Gamma_D$ (a função peso vale zero quando aplicada no contorno onde a condição é essencial. Isso porque neste ponto a função já é conhecida [17]). Assim, tem-se:

$$-(\nabla \cdot (\mu(\mathbf{x}) \nabla u(\mathbf{x})))v(\mathbf{x}) = f(\mathbf{x})v(\mathbf{x}).$$

Integrando sobre Ω e omitindo a dependência explícita de $\mathbf{x}$ para simplificar a notação, tem-se:

$$-\int_{\Omega} (\nabla \cdot (\mu \nabla u))v \, dx = \int_{\Omega} f v \, dx. \quad (2.7)$$

Antes de prosseguir, observe que:

$$\begin{aligned}
\nabla \cdot ((\mu \nabla u)v) &= \left(\frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2}, \dots, \frac{\partial}{\partial x_n} \right) \cdot ((\mu u_{x_1}, \mu u_{x_2}, \dots, \mu u_{x_n})v) \\
&= \left(\frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2}, \dots, \frac{\partial}{\partial x_n} \right) \cdot (\mu u_{x_1}v, \mu u_{x_2}v, \dots, \mu u_{x_n}v) \\
&= \frac{\partial(\mu u_{x_1}v)}{\partial x_1} + \frac{\partial(\mu u_{x_2}v)}{\partial x_2} + \dots + \frac{\partial(\mu u_{x_n}v)}{\partial x_n} \\
&= \left(\frac{\partial(\mu u_{x_1})}{\partial x_1}v + \mu u_{x_1} \frac{\partial v}{\partial x_1} \right) + \dots + \left(\frac{\partial(\mu u_{x_n})}{\partial x_n}v + \mu u_{x_n} \frac{\partial v}{\partial x_n} \right) \\
&= \left(\frac{\partial(\mu u_{x_1})}{\partial x_1}v + \dots + \frac{\partial(\mu u_{x_n})}{\partial x_n}v \right) + \left(\mu u_{x_1} \frac{\partial v}{\partial x_1} + \dots + \mu u_{x_n} \frac{\partial v}{\partial x_n} \right) \\
&= \left(\frac{\partial(\mu u_{x_1})}{\partial x_1} + \dots + \frac{\partial(\mu u_{x_n})}{\partial x_n} \right) v + \mu \left(u_{x_1} \frac{\partial v}{\partial x_1} + \dots + u_{x_n} \frac{\partial v}{\partial x_n} \right) \\
&= \left(\left(\frac{\partial}{\partial x_1}, \dots, \frac{\partial}{\partial x_n} \right) \cdot (\mu u_{x_1}, \dots, \mu u_{x_n}) \right) v + \mu (u_{x_1}, \dots, u_{x_n}) \cdot \left(\frac{\partial v}{\partial x_1}, \dots, \frac{\partial v}{\partial x_n} \right) \\
&= (\nabla \cdot (\mu \nabla u)) v + \mu \nabla u \cdot \nabla v.
\end{aligned}$$

Logo, ao manipular a equação e integrá-la sobre o domínio Ω , obtém-se:

$$\int_{\Omega} (\nabla \cdot (\mu \nabla u)) v \, dx = \underbrace{\int_{\Omega} \nabla \cdot ((\mu \nabla u)v) \, dx}_{(I)} - \int_{\Omega} \mu \nabla u \cdot \nabla v \, dx. \quad (2.8)$$

Ao aplicar o Teorema de Gauss no termo (I) da equação (2.8),

$$\int_{\Omega} \nabla \cdot ((\mu \nabla u)v) \, dx = \int_{\Gamma} v \mu \nabla u \cdot \mathbf{\check{n}} \, ds.$$

Ao separar a integral na fronteira Γ em duas integrais, a primeira sendo a integral sobre a fronteira com condição de contorno de Dirichlet Γ_D e a segunda sobre a fronteira com condição de contorno de Neumann Γ_N , obtém-se:

$$\int_{\Omega} \nabla \cdot ((\mu \nabla u)v) \, dx = \int_{\Gamma} v \mu \nabla u \cdot \mathbf{\check{n}} \, ds = \int_{\Gamma_D} \overset{0}{\cancel{v \mu \nabla u \cdot \mathbf{\check{n}}}} \, d\Gamma_D + \int_{\Gamma_N} v \underbrace{\mu \nabla u \cdot \mathbf{\check{n}}}_{=-h} \, d\Gamma_N.$$

Observa-se que a integral sobre Γ_D se anula. Além disso, sobre Γ_N , a condição de contorno de Neumann fornece $-\mu \nabla u(\mathbf{x}) \cdot \mathbf{\check{n}} = h(\mathbf{x})$. Assim, segue que:

$$\int_{\Omega} \nabla \cdot ((\mu \nabla u)v) \, dx = - \int_{\Gamma_N} h v \, d\Gamma_N. \quad (2.9)$$

Substituindo (2.9) em (2.8) e multiplicando por -1 , obtém-se:

$$- \int_{\Omega} (\nabla \cdot (\mu \nabla u)) v \, dx = \int_{\Gamma_N} h v \, d\Gamma_N + \int_{\Omega} \mu \nabla u \cdot \nabla v \, dx. \quad (2.10)$$

Finalmente, substituindo (2.10) em (2.7) chega-se à formulação variacional do problema:

$$\underbrace{\int_{\Omega} \mu \nabla u(\mathbf{x}) \cdot \nabla v(\mathbf{x}) \, dx}_{a(u,v)} = \underbrace{\int_{\Omega} f(\mathbf{x})v(\mathbf{x}) \, dx - \int_{\Gamma_N} h(\mathbf{x})v(\mathbf{x}) \, d\Gamma_N}_{l(v)}, \quad (2.11)$$

onde os termos dos lados esquerdo e direito da equação (2.11) são, respectivamente, uma forma bilinear a e uma forma linear l .

Nessa formulação, a condição de contorno de Neumann surge naturalmente (condição natural), enquanto a condição de Dirichlet é imposta na definição do espaço de funções teste (condição essencial).

2.6 Formulação Variacional Unidimensional

Agora, considera-se a formulação para um problema unidimensional, análogo ao problema (2.6). Neste caso, impõe-se uma condição de Dirichlet em $x = a$ e uma condição de Neumann em $x = b$, permitindo analisar a influência de cada uma na formulação variacional.

Seja $\Omega = [a, b]$, o problema é dado por:

$$\begin{cases} -(\mu(x)u'(x))' &= f(x) \quad , \quad x \in \Omega, \\ u(a) &= g, \\ -\mu(b)u'(b) &= h, \end{cases} \quad (2.12)$$

onde $f : \Omega \rightarrow \mathbb{R}$ representa o termo fonte, $g \in \mathbb{R}$ e $h \in \mathbb{R}$ são constantes conhecidas que descrevem, respectivamente, as condições de contorno de Dirichlet sobre Γ_D e de Neumann sobre Γ_N , $\mu : \Omega \rightarrow \mathbb{R}$ é a função coeficiente de difusão e $u : \Omega \rightarrow \mathbb{R}$ é a função incógnita. A notação $u'(x)$ é utilizada para representar $\frac{du}{dx}$, visando simplificação.

A formulação variacional do problema inicia-se multiplicando a equação diferencial por uma função peso $v(x)$, onde $v : \Omega \rightarrow \mathbb{R}$, tal que $v(a) = 0$, pois neste ponto a função já é conhecida ($u(a) = g$). Assim, obtém-se:

$$-(\mu(x)u'(x))' v(x) = f(x)v(x).$$

Integrando sobre Ω , segue-se:

$$-\int_a^b (\mu(x)u'(x))' v(x) dx = \int_a^b f(x)v(x) dx. \quad (2.13)$$

Antes de dar continuidade, observe que:

$$\begin{aligned} [(\mu(x)u'(x))v(x)]' &= (\mu(x)u'(x))'v(x) + \mu(x)u'(x)v'(x), \\ \Rightarrow (\mu(x)u'(x))'v(x) &= [(\mu(x)u'(x))v(x)]' - \mu(x)u'(x)v'(x), \end{aligned}$$

Aplicando a integral em ambos os lados, tem-se:

$$\Rightarrow \int_a^b (\mu(x)u'(x))'v(x)dx = \underbrace{\int_a^b [(\mu(x)u'(x))v(x)]' dx}_{(II)} - \int_a^b \mu(x)u'(x)v'(x)dx, \quad (2.14)$$

Pelo Teorema Fundamental do Cálculo, o termo (II) da equação (2.14) resulta em:

$$\int_a^b [(\mu(x)u'(x))v(x)]' dx = \underbrace{\mu(b)u'(b)v(b)}_{=-h} - \underbrace{\mu(a)u'(a)v(a)}_{v(a)=0}.$$

Como $v(a) = 0$ e $-\mu(b)u'(b) = h$ segue que:

$$\int_a^b [(\mu(x)u'(x))v(x)]' dx = -hv(b). \quad (2.15)$$

Substituindo (2.15) em (2.14) e multiplicando por (-1) , obtém-se:

$$\Rightarrow - \int_a^b (\mu(x)u'(x))'v(x)dx = hv(b) + \int_a^b \mu(x)u'(x)v'(x)dx, \quad (2.16)$$

Por fim, substituindo (2.16) em (2.13), obtém-se a formulação fraca do problema:

$$\underbrace{\int_a^b \mu(x)u'(x)v'(x)dx}_{a(u,v)} = \underbrace{\int_a^b f(x)v(x) dx}_{l(v)} - hv(b). \quad (2.17)$$

2.7 Método de Galerkin

Neste trabalho, será utilizado o importante método de Galerkin para construir uma solução aproximada. [5; 18; 22].

O objetivo é encontrar $u \in V$ tal que:

$$a(u, v) = l(v) \quad \forall v \in V, \quad (2.18)$$

formas bilinear e linear que definem a formulação variacional (2.11) (ou (2.17) no caso unidimensional).

O método de Galerkin consiste em discretizar o problema (2.18), substituindo o espaço V pelo espaço de dimensão finita $V_h \subset V$ com o objetivo de encontrar $u_h \in V_h$ por meio da formulação variacional discreta dada por:

$$a(u_h, v_h) = l(v_h) \quad \forall v_h \in V_h. \quad (2.19)$$

Sejam $\phi_1, \phi_2, \dots, \phi_n \in V_h$ funções linearmente independentes, o conjunto formado por essas ϕ_j , $j = 1, \dots, n$ forma uma base, sendo ϕ_j denominadas funções base.

O objetivo é determinar uma função $u_h(\mathbf{x}) \in V_h$ que satisfaça (2.19). A solução aproximada $u_h(\mathbf{x})$ é expressa como uma combinação linear das funções base:

$$u_h(\mathbf{x}) = \sum_{j=1}^n U_j \phi_j(\mathbf{x}), \quad (2.20)$$

onde $\mathbf{x} \in \Omega \subset \mathbb{R}^d$, $d = 1, 2$ e 3 .

Para simplificar a notação e facilitar a explicação, considere o caso unidimensional, ou seja, $d = 1$, com $\Omega = [a, b] \subset \mathbb{R}$.

Como o espaço V_h possui dimensão n , é necessário obter n equações linearmente independentes para determinar os coeficientes U_j . Uma abordagem comum, adotada no método de Galerkin, é escolher as funções v como as próprias funções base ϕ_i . Dessa forma, cada equação é associada a uma função ϕ_i , isso é, na primeira equação, toma-se $v = \phi_1$, na segunda, $v = \phi_2$ e assim sucessivamente. Desta maneira, obtém-se n equações linearmente independentes, onde a equação correspondente a cada índice i assume a seguinte forma:

$$\begin{aligned} \int_a^b \mu(x) \left(\sum_{j=1}^n U_j \phi_j'(x) \right) \phi_i'(x) dx &= \int_a^b f(x) \phi_i(x) dx + h(a) \phi_i(a) - h(b) \phi_i(b) \Rightarrow \\ \Rightarrow \sum_{j=1}^n \left(\int_a^b \mu(x) \phi_j'(x) \phi_i'(x) dx U_j \right) &= \int_a^b f(x) \phi_i(x) dx + h(a) \phi_i(a) - h(b) \phi_i(b), \\ &(i = 1, 2, \dots, n). \end{aligned} \quad (2.21)$$

Esse sistema de n equações pode ser escrito na forma matricial $\mathbf{AU} = \mathbf{F}$, onde os elementos da matriz $\mathbf{A}$ são dados por:

$$A_{ij} = a(\phi_i, \phi_j) = \int_a^b \mu(x) \phi_j'(x) \phi_i'(x) dx, \quad (2.22)$$

e os elementos do vetor $\mathbf{F}$ calculados como:

$$F_i = l(\phi_i) = \int_a^b f(x) \phi_i(x) dx + h(a) \phi_i(a) - h(b) \phi_i(b).$$

A solução aproximada $u_h(x)$ pode então ser obtida determinando os coeficientes U_j .

É importante notar que os elementos do vetor $\mathbf{F}$ foram aqui apresentados de forma genérica, sem levar em conta possíveis mudanças decorrentes das condições de contorno. A influência dessas condições, sejam elas de Dirichlet ou Neumann, foi discutida na seção anterior, onde a formulação variacional unidimensional foi apresentada.

2.8 Problema bem posto

Para que um problema seja resolvido de maneira significativa, é essencial garantir que ele admita solução e que ela seja única e estável. Esse conceito, introduzido por Jacques Hadamard (1865-1963), define um problema como bem posto se satisfazer os seguintes critérios:

- Existência: Existe ao menos uma solução;
- Unicidade: Se a solução existe, ela é única;
- Estabilidade: pequenas perturbações nos dados de entrada geram pequenas perturbações nos resultados.

Suponha que o problema contínuo 2.18 seja bem posto. Isso garante que a formulação discreta 2.19 também o seja? [18].

Para responder essa questão, considere os seguintes pontos:

- Seja V um espaço de Hilbert, então $V_h \subset V$ é também um espaço de Hilbert;
- Desde que a forma bilinear a seja limitada em V , ou seja, $|a(u, v)| \leq N_a \|u\|_V \|v\|_V, \forall u, v \in V$, então a também é em particular limitada em V_h ;
- Desde que a forma bilinear a seja fortemente coerciva, ou seja, $a(u, u) \geq \alpha \|u\|_V^2 \forall u \in V$, então a também é em particular coerciva em V_h ;
- De maneira análoga, se a forma linear $l(v)$ for contínua em V , então $l(v)$ é também contínua em V_h .
- A forma bilinear a é limitada se, e somente se, a for contínua.

Para o caso em que a forma bilinear a é simétrica, considere também o Teorema 1 a seguir.

Teorema 1 *Seja V um espaço de Hilbert e $a : V \times V \rightarrow \mathbb{R}$ uma forma bilinear simétrica, contínua e fortemente coerciva. Para cada funcional linear contínuo $l \in V^*$, existe um único $u \in V$ tal que:*

$$a(u, v) = l(v) \quad \forall v \in V.$$

Além disso, a solução u é estável, ou seja, $\|u\|_V \leq \frac{1}{\alpha} \|l\|_{V^*}$, para uma constante $\alpha > 0$.

Caso $a(\cdot, \cdot)$ não seja simétrica, aplica-se o teorema de Lax-Milgram:

Teorema 2 (Lax-Milgram) *Seja V um espaço de Hilbert e $a : V \times V \rightarrow \mathbb{R}$ uma forma bilinear contínua e fortemente coerciva. Então, o problema:*

Encontre $u \in V$ tal que:

$$a(u, v) = l(v) \quad \forall v \in V$$

é bem posto para qualquer funcional linear $l \in V^*$ e $\|u\|_V \leq \frac{1}{\alpha} \|l\|_{V^*}$.

Dessa forma, independente da simetria de $a(\cdot, \cdot)$, o seguinte corolário se aplica:

Corolário 1 *Existe um único $u_h \in V_h$ que satisfaz um problema variacional discreto: Encontre $u_h \in V_h$ tal que:*

$$a(u_h, v_h) = l(v_h) \quad \forall v_h \in V_h.$$

Para calcular u_h pelo método de Galerkin, escolhe-se uma base $\{\phi_1, \phi_2, \dots, \phi_n\}$ de V_h e substitui-se u_h , equação (2.20), e $v_h = \phi_i$ na equação 2.19, resultando em:

$$a \left(\sum_{j=1}^n U_j \phi_j, \phi_i \right) = \sum_{j=1}^n \underbrace{a(\phi_j, \phi_i)}_{A_{ij}} U_j = \underbrace{l(\phi_i)}_{F_i}, \quad i = 1, \dots, n.$$

Obtendo um sistema linear $\mathbf{A}\mathbf{U} = \mathbf{F}$, onde $\mathbf{U} \in \mathbb{R}^n$, $\mathbf{A} \in \mathbb{R}^{n \times n}$ e $\mathbf{F} \in \mathbb{R}^n$.

Com relação à matriz A :

- Se a forma bilinear a for simétrica, ou seja, $a(\phi_j, \phi_i) = a(\phi_i, \phi_j)$, então A também é simétrica;
- Se $a(v_h, v_h) > 0$, tomando qualquer v_h arbitrário não nulo, $v_h = \sum_{j=1}^n V_j \phi_j$ (ou seja, $\mathbf{V} \neq \mathbf{0}$), obtém-se:

$$a \left(\sum_{j=1}^n V_j \phi_j, \sum_{i=1}^n V_i \phi_i \right) = \sum_{i=1}^n \sum_{j=1}^n V_i A_{ij} V_j = \mathbf{V}^T \mathbf{A} \mathbf{V} > 0, \quad \mathbf{V} \neq \mathbf{0},$$

Assim, $\mathbf{A}$ é positiva definida e, portanto, invertível, garantindo a existência e unicidade de $\mathbf{U}$, que define u_h .

Conclui-se, portanto, que se o problema contínuo for bem posto, sua formulação discreta também o será [18].

2.9 Melhor aproximação e Ortogonalidade de Galerkin

No método de elementos finitos, a propriedade de melhor aproximação e a ortogonalidade de Galerkin desempenham um papel fundamental na análise da qualidade das soluções aproximadas [18].

Seja $u \in V$ a solução analítica do problema variacional e $u_h \in V_h$ a solução aproximada no subespaço V_h , o Lemma de J. Céa fornece uma estimativa do erro, considerando a norma em V .

Lema 1 (J. Céa) *Seja $a(\cdot, \cdot)$ e $l(\cdot)$ formas contínuas em V com $a(\cdot, \cdot)$ fortemente coercivo, então:*

$$\|u - u_h\|_V \leq \frac{N_a}{\alpha} \min_{v_h \in V_h} \|u - v_h\|_V,$$

onde N_a é a constante da continuidade e α a constante da coercividade.

Nos casos em que $a(\cdot, \cdot)$ é simétrica, o método de Galerkin garante que a solução u_h fornece a melhor aproximação no espaço V_h com respeito à norma da energia, definida por $\|\cdot\|_a = \sqrt{a(\cdot, \cdot)}$, onde vale a seguinte relação:

$$\|u - u_h\|_a \leq \|u - v_h\|_a \quad \forall v_h \in V_h.$$

A simetria também expressa a ortogonalidade de Galerkin por meio da forma bilinear associada ao problema:

$$a(u - u_h, v_h) = 0 \quad \forall v_h \in V_h.$$

Essa relação indica que o erro $e = u - u_h$ é ortogonal ao subespaço V_h com respeito à forma bilinear, garantindo que o resíduo seja minimizado [18]. Esta relação é ilustrada pela Figura 2.2.

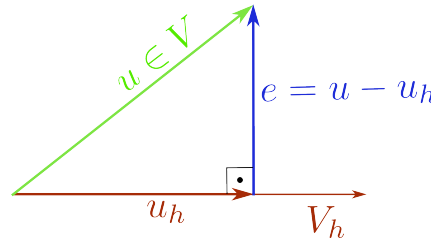


Figura 2.2: Representação de u_h como a projeção ortogonal de u em V_h .

2.10 Exemplo: Problema de Poisson 1D

Neste exemplo, considere o problema unidimensional de Poisson (2.12), porém apenas com condição de contorno do tipo Dirichlet. Através da formulação fraca deste problema (2.17), e assumindo que $\mu(x) = 1$, $\forall x \in \Omega$, o problema pode ser expresso na seguinte forma:

$$\begin{cases} \underbrace{\int_0^1 u'(x)v'(x) dx}_{a(u,v)} = \underbrace{\int_0^1 f v(x) dx}_{l(v)} & , x \in \Omega, \forall v_h \in V(\Omega), \\ u(0) = u(1) = 0 & , x \in \Gamma_D, \end{cases} \quad (2.23)$$

onde $\Omega = (0, 1)$ representa o interior do domínio do problema, $\Gamma_D = \{0, 1\}$ é a fronteira de Ω e $f(x) = 2$, $\forall x \in \Omega$ é o termo fonte constante.

Para fins didáticos, o problema é discretizado em apenas três elementos finitos, conforme ilustrado na Figura 2.3.

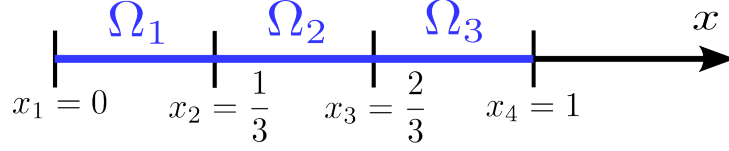


Figura 2.3: Domínio Discretizado

O espaço das soluções V_h adotado é o das funções polinomiais de grau 1 em cada elemento e definidas de modo que a solução aproximada global seja contínua entre os elementos, da seguinte forma:

$$V_h = \{v, v|_{(x_1, x_2)} \in P_1(x_1, x_2), v|_{(x_2, x_3)} \in P_1(x_2, x_3), v|_{(x_3, x_4)} \in P_1(x_3, x_4)\} \cap C^0(x_1, x_4).$$

Os graus de liberdade (U_j) são definidos de modo que correspondam aos valores da função nos pontos extremos de cada intervalo (nos nós). Como a malha foi discretizada em 3 elementos, existem 4 nós e, portanto 4 graus de liberdade:

$$\begin{aligned} U_1 &= v(x_1), \\ U_2 &= v(x_2), \\ U_3 &= v(x_3), \\ U_4 &= v(x_4). \end{aligned}$$

O número de graus de liberdade é igual ao número de funções base ϕ_j e estas funções podem ser escritas conforme (2.24).

$$\phi_j(x) = \begin{cases} \alpha_j + \beta_j x & , \text{ se } x \in (x_1, x_2), \\ \gamma_j + \xi_j x & , \text{ se } x \in (x_2, x_3), \\ \theta_j + \omega_j x & , \text{ se } x \in (x_3, x_4). \end{cases} \quad (2.24)$$

Assim, para $j = 1, \dots, 4$:

$$\begin{aligned} \phi_j(x_1^+) &= \phi_j(x_1) = \alpha_j + \beta_j x_1, \\ \phi_j(x_2^-) &= \phi_j(x_2) = \alpha_j + \beta_j x_2, \\ \phi_j(x_2^+) &= \phi_j(x_2) = \gamma_j + \xi_j x_2, \\ \phi_j(x_3^-) &= \phi_j(x_3) = \gamma_j + \xi_j x_3, \\ \phi_j(x_3^+) &= \phi_j(x_3) = \theta_j + \omega_j x_3, \\ \phi_j(x_4^-) &= \phi_j(x_4) = \theta_j + \omega_j x_4. \end{aligned}$$

Note que a continuidade entre os elementos implica em $\phi_j(x_2^-) = \phi_j(x_2) = \phi_j(x_2^+)$ e as notações de limites laterais foram simplificadas da forma: $\lim_{x \rightarrow x_2^-} \phi_j(x) = \phi_j(x_2^-)$ e

$$\lim_{x \rightarrow x_2^+} \phi_j(x) = \phi_j(x_2^+).$$

A propriedade de Kronecker definida em (2.25)

$$\phi_i(x_j) = \delta_{ij} \Rightarrow \phi_i(x_j) = \begin{cases} 1 & \text{se } i = j, \\ 0 & \text{se } i \neq j, \end{cases} \quad (2.25)$$

é aplicada nas funções base (2.24), resultando em:

$$\left\{ \begin{array}{l} \alpha_1 + \beta_1 x_1 = 1 \\ \alpha_1 + \beta_1 x_2 = 0 \\ \gamma_1 + \xi_1 x_2 = 0 \\ \gamma_1 + \xi_1 x_3 = 0 \\ \theta_1 + \omega_1 x_3 = 0 \\ \theta_1 + \omega_1 x_4 = 0 \end{array} \right\}, \left\{ \begin{array}{l} \alpha_2 + \beta_2 x_1 = 0 \\ \alpha_2 + \beta_2 x_2 = 1 \\ \gamma_2 + \xi_2 x_2 = 1 \\ \gamma_2 + \xi_2 x_3 = 0 \\ \theta_2 + \omega_2 x_3 = 0 \\ \theta_2 + \omega_2 x_4 = 0 \end{array} \right\}, \left\{ \begin{array}{l} \alpha_3 + \beta_3 x_1 = 0 \\ \alpha_3 + \beta_3 x_2 = 0 \\ \gamma_3 + \xi_3 x_2 = 0 \\ \gamma_3 + \xi_3 x_3 = 1 \\ \theta_3 + \omega_3 x_3 = 1 \\ \theta_3 + \omega_3 x_4 = 0 \end{array} \right\}, \left\{ \begin{array}{l} \alpha_4 + \beta_4 x_1 = 0 \\ \alpha_4 + \beta_4 x_2 = 0 \\ \gamma_4 + \xi_4 x_2 = 0 \\ \gamma_4 + \xi_4 x_3 = 0 \\ \theta_4 + \omega_4 x_3 = 0 \\ \theta_4 + \omega_4 x_4 = 1 \end{array} \right\}.$$

O primeiro sistema pode ser resolvido da forma:

$$\begin{aligned} \beta_1(x_2 - x_1) = -1 &\Rightarrow \beta_1 = \frac{-1}{x_2 - x_1} \Rightarrow \alpha_1 = \frac{x_2}{x_2 - x_1}, \\ \xi_1(x_3 - x_2) = 0 &\Rightarrow \xi_1 = 0, \text{ pois } x_2 \neq x_3 \Rightarrow \gamma_1 = 0, \\ \omega_1(x_4 - x_3) = 0 &\Rightarrow \omega_1 = 0, \text{ pois } x_3 \neq x_4 \Rightarrow \theta_1 = 0. \end{aligned}$$

Portanto,

$$\phi_1(x) = \begin{cases} \frac{x_2 - x}{x_2 - x_1} = -3x + 1 & , \text{ se } x \in (x_1, x_2), \\ 0 & , \text{ caso contrário.} \end{cases}$$

De forma análoga, ao resolver os outros três sistemas de forma, obtém-se:

$$\begin{aligned} \phi_2(x) &= \begin{cases} \frac{x - x_1}{x_2 - x_1} = 3x & , \text{ se } x \in (x_1, x_2), \\ \frac{x_3 - x}{x_3 - x_2} = -3x + 2 & , \text{ se } x \in (x_2, x_3), \\ 0 & , \text{ se } x \in (x_3, x_4). \end{cases} \\ \phi_3(x) &= \begin{cases} 0 & , \text{ se } x \in (x_1, x_2), \\ \frac{x - x_2}{x_3 - x_2} = 3x - 1 & , \text{ se } x \in (x_2, x_3), \\ \frac{x_4 - x}{x_4 - x_3} = -3x + 3 & , \text{ se } x \in (x_3, x_4). \end{cases} \\ \phi_4(x) &= \begin{cases} \frac{x - x_3}{x_4 - x_3} = 3x - 2 & , \text{ se } x \in (x_3, x_4), \\ 0 & , \text{ caso contrário.} \end{cases} \end{aligned}$$

As representações destas funções bases no plano cartesiano são mostradas nas Figuras 2.4, 2.5, 2.6 e 2.7.

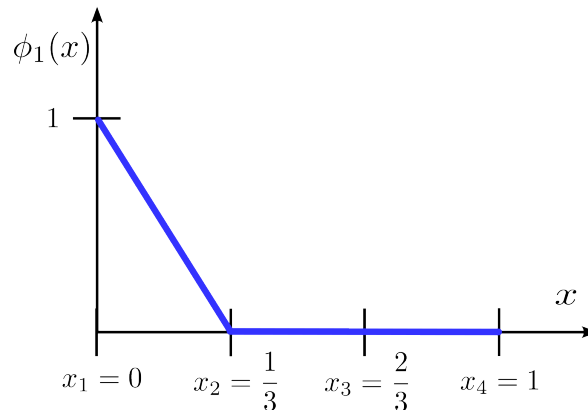
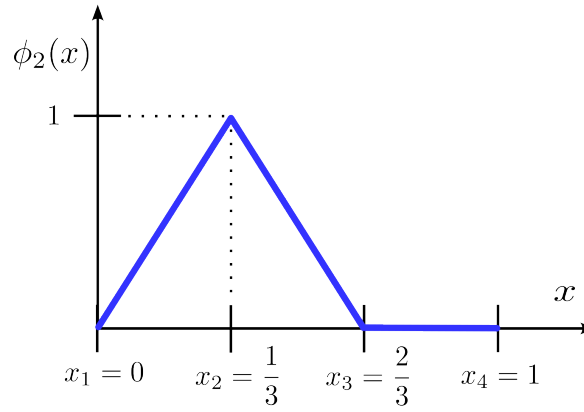
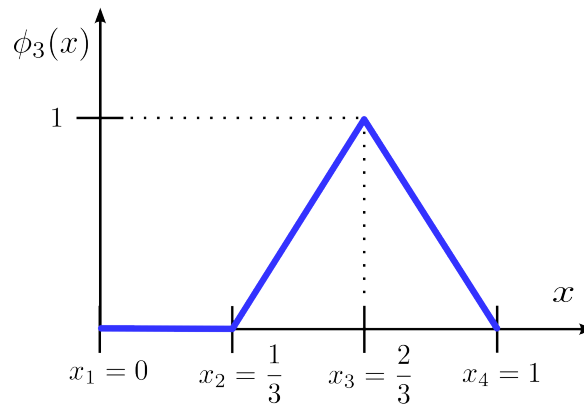
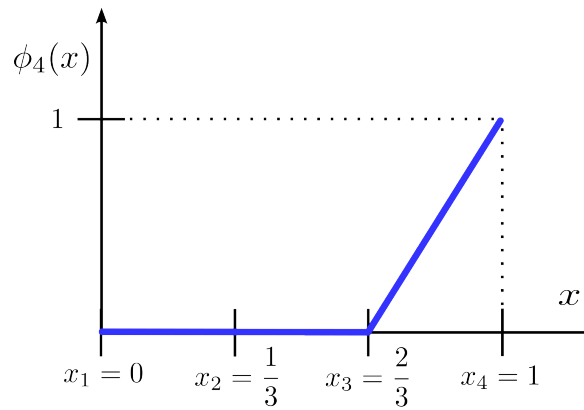


Figura 2.4: Função base $\phi_1(x)$

Figura 2.5: Função base $\phi_2(x)$ Figura 2.6: Função base $\phi_3(x)$ Figura 2.7: Função base $\phi_4(x)$

Analisando as Figuras 2.4, 2.5, 2.6 e 2.7 fica evidente que estas funções foram construídas de forma que $\phi_i(x) \neq 0$ apenas nos elementos que fazem fronteira com x_i . A propriedade de Kronecker (2.25) assegura essa importante característica, que será comentada posteriormente ressaltando os benefícios nas implementações computacionais.

No método de Galerkin, visto que as funções ϕ_j já foram construídas, o próximo passo é determinar as incógnitas U_j (ver eq. (2.20)). As incógnitas U_1 e U_4 são imediatamente determinadas pela imposição das condições de contorno de Dirichlet $u(0) = U_1 = 0$ e $u(1) = U_4 = 0$. Para determinar as duas incógnitas restantes, U_2 e U_3 , a formulação

variacional (2.17), adaptada para o caso da condição de Dirichlet em toda fronteira de Ω , será empregada considerando as funções peso como $v = \phi_2$ e $v = \phi_3$. Iniciando com $v_h(x) = \phi_2$:

$$\begin{aligned} \int_0^1 (U_1 \phi_1'(x) + U_2 \phi_2'(x) + U_3 \phi_3'(x) + U_4 \phi_4'(x)) \phi_2'(x) dx &= \int_0^1 f \phi_2(x) dx \\ \Rightarrow \int_0^1 (U_1 \phi_1'(x) \phi_2'(x) + U_2 \phi_2'(x) \phi_2'(x) + U_3 \phi_3'(x) \phi_2'(x) + U_4 \phi_4'(x) \phi_2'(x)) dx &= \\ &= \int_0^1 f \phi_2(x) dx. \\ \Rightarrow U_1 \int_0^1 (\phi_1'(x) \phi_2'(x)) dx + U_2 \int_0^1 (\phi_2'(x) \phi_2'(x)) dx + U_3 \int_0^1 (\phi_3'(x) \phi_2'(x)) dx + \\ &+ U_4 \int_0^1 (\phi_4'(x) \phi_2'(x)) dx = \int_0^1 f \phi_2(x) dx. \end{aligned}$$

Agora ao passar os termos conhecidos para o lado direito da equação, obtém-se:

$$\begin{aligned} & \underbrace{U_2 \int_0^1 (\phi_2'(x) \phi_2'(x)) dx}_{a(\phi_2, \phi_2)} + \underbrace{U_3 \int_0^1 (\phi_3'(x) \phi_2'(x)) dx}_{a(\phi_3, \phi_2)} = \\ &= \underbrace{\int_0^1 2\phi_2(x) dx}_{l(\phi_2)} - \underbrace{U_1 \int_0^1 (\phi_1'(x) \phi_2'(x)) dx}_{a(\phi_1, \phi_2)} - \underbrace{U_4 \int_0^1 (\phi_4'(x) \phi_2'(x)) dx}_{a(\phi_4, \phi_2)}. \end{aligned} \quad (2.26)$$

E de forma análoga, obtém-se para $v_h(x) = \phi_3(x)$:

$$\begin{aligned} & \underbrace{U_2 \int_0^1 (\phi_2'(x) \phi_3'(x)) dx}_{a(\phi_2, \phi_3)} + \underbrace{U_3 \int_0^1 (\phi_3'(x) \phi_3'(x)) dx}_{a(\phi_3, \phi_3)} = \\ &= \underbrace{\int_0^1 2\phi_3(x) dx}_{l(\phi_3)} - \underbrace{U_1 \int_0^1 (\phi_1'(x) \phi_3'(x)) dx}_{a(\phi_1, \phi_3)} - \underbrace{U_4 \int_0^1 (\phi_4'(x) \phi_3'(x)) dx}_{a(\phi_4, \phi_3)}. \end{aligned} \quad (2.27)$$

As equações (2.26) e (2.27) podem ser escritas na forma matricial. Note que trata-se de um sistema linear da forma $\mathbf{A} \mathbf{U} = \mathbf{F}$:

$$\underbrace{\begin{bmatrix} a(\phi_2, \phi_2) & a(\phi_3, \phi_2) \\ a(\phi_2, \phi_3) & a(\phi_3, \phi_3) \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} U_2 \\ U_3 \end{bmatrix}}_{\mathbf{U}} = \underbrace{\begin{bmatrix} l(\phi_2) - U_1 a(\phi_1, \phi_2) - U_4 a(\phi_4, \phi_2) \\ l(\phi_3) - U_1 a(\phi_1, \phi_3) - U_4 a(\phi_4, \phi_3) \end{bmatrix}}_{\mathbf{F}}.$$

As formas bilineares e lineares que compõem as matrizes $\mathbf{A}$ e $\mathbf{F}$ são calculadas por meio das integrais definidas, conforme ilustrado abaixo:

$$\begin{aligned} a(\phi_2, \phi_2) &= \int_0^1 (\phi_2'(x) \phi_2'(x)) dx \\ &= \int_0^{\frac{1}{3}} (\phi_2'(x) \phi_2'(x)) dx + \int_{\frac{1}{3}}^{\frac{2}{3}} (\phi_2'(x) \phi_2'(x)) dx + \int_{\frac{2}{3}}^1 \left(\cancel{\phi_2'(x)}^0 \cancel{\phi_2'(x)}^0 \right) dx \\ &= \int_0^{\frac{1}{3}} 3 dx + \int_{\frac{1}{3}}^{\frac{2}{3}} -3 dx \\ &= 6. \end{aligned}$$

$$\begin{aligned}
l(\phi_2) &= \int_0^1 (2\phi_2(x)) dx \\
&= \int_0^{\frac{1}{3}} (2\phi_2(x)) dx + \int_{\frac{1}{3}}^{\frac{2}{3}} (2\phi_2(x)) dx + \int_{\frac{2}{3}}^1 \left(2\phi_2(x) \right)^0 dx \\
&= \int_0^{\frac{1}{3}} (6x) dx + \int_{\frac{1}{3}}^{\frac{2}{3}} (-6x + 4) dx \\
&= \frac{2}{3}.
\end{aligned}$$

Os demais elementos são calculados de maneira similar, resultando em:

$$\begin{bmatrix} 6 & -3 \\ -3 & 6 \end{bmatrix} \begin{bmatrix} U_2 \\ U_3 \end{bmatrix} = \begin{bmatrix} \frac{2}{3} \\ \frac{2}{3} \end{bmatrix}.$$

Resolvendo este sistema linear obtém-se:

$$U = \begin{bmatrix} \frac{2}{9} & \frac{2}{9} \end{bmatrix}^T.$$

Logo, da eq. (2.20), vem que,

$$u_h = \frac{2}{9}\phi_2 + \frac{2}{9}\phi_3.$$

E portanto, a solução do problema é dada por:

$$u_h = \begin{cases} \frac{2x}{3} & , \text{ se } x \in \left(0, \frac{1}{3}\right), \\ \frac{2}{9} & , \text{ se } x \in \left(\frac{1}{3}, \frac{2}{3}\right), \\ \frac{2-2x}{3} & , \text{ se } x \in \left(\frac{2}{3}, 1\right). \end{cases}$$

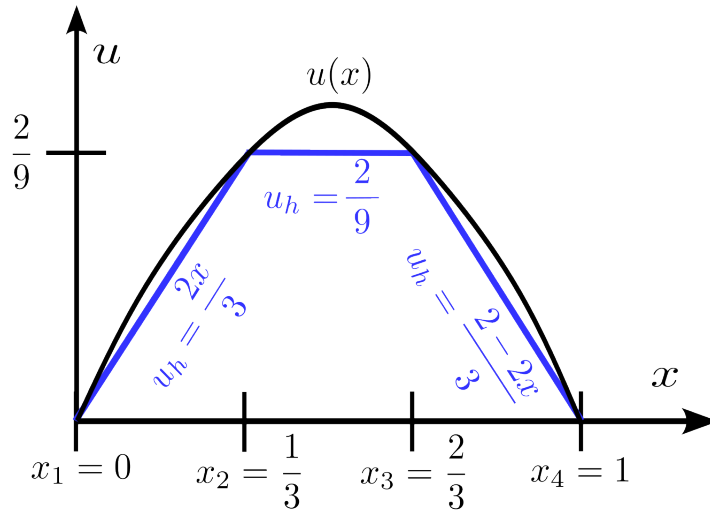


Figura 2.8: Esboço da Solução Aproximada u_h e da Analítica $u(x)$ do problema (2.12) com condição de contorno do tipo Dirichlet na fronteira de Ω .

A Figura 2.8 ilustra a solução aproximada u_h (em azul), obtida pelo método de elementos finitos, juntamente com a solução analítica $u(x)$ (em preto). Note que, em razão das escolhas feitas no processo de discretização, a solução aproximada é composta por

funções polinomiais de grau 1 em cada elemento, sendo contínuas entre os elementos. Além disso, destaca-se que $u(x) = u_h$ nos 4 graus de liberdade localizados nos nós $x = x_1$, $x = x_2$, $x = x_3$ e $x = x_4$.

À medida que o número de elementos finitos aumenta, a matriz resultante do sistema linear torna-se esparsa, pois cada função base é não nula apenas em elementos vizinhos. Essa característica é explorada computacionalmente para reduzir custos de armazenamento e acelerar a resolução do sistema.

2.11 Elemento Mestre

A implementação do Método dos Elementos Finitos envolve uma quantidade significativa de cálculos, especialmente ao aumentar o número de elementos, elevar o grau dos polinômios ou lidar com problemas em duas ou três dimensões. Para mitigar essa complexidade, utiliza-se o conceito do elemento mestre $\hat{\Omega}$ (ou elemento de referência), que se torna uma ferramenta essencial na formulação e implementação computacional do método.

A ideia fundamental do elemento mestre é realizar uma transformação de coordenadas que mapeia qualquer elemento da malha para uma referência padronizada, simplificando a avaliação das integrais envolvidas na formulação variacional. Em particular, quando o domínio é irregular, ele é discretizado em elementos quadriláteros ou triângulos. No entanto, as integrais sobre tais elementos tornam-se algebricamente complexas, exigindo o uso de integração numérica. Nesse contexto, as quadraturas de Gauss-Legendre são amplamente empregadas, pois requerem que as integrais sejam expressas em um domínio padronizado, usualmente um quadrado de lado 2 com coordenadas variando entre -1 e 1. Essa transformação facilita a implementação do método e a avaliação das integrais envolvidas, conforme discutido em [6].

Dado um elemento mestre, cada elemento da malha pode ser obtido por uma transformação $T_e : \hat{\Omega} \rightarrow \Omega_e$. Essa transformação define um mapeamento entre as coordenadas do elemento mestre e as coordenadas do elemento na malha. Por exemplo, no caso de elementos triangulares, as coordenadas locais $(\hat{x}, \hat{y})$ do elemento mestre são transformadas nas coordenadas reais (x, y) por meio da equação (2.28) e ilustrada pela Figura 2.9:

$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} = T_e(\hat{x}_i, \hat{y}_i) = \underbrace{\begin{bmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{bmatrix}}_{B_{\Omega_e}} \begin{bmatrix} \hat{x}_i \\ \hat{y}_i \end{bmatrix} + \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}. \quad (2.28)$$

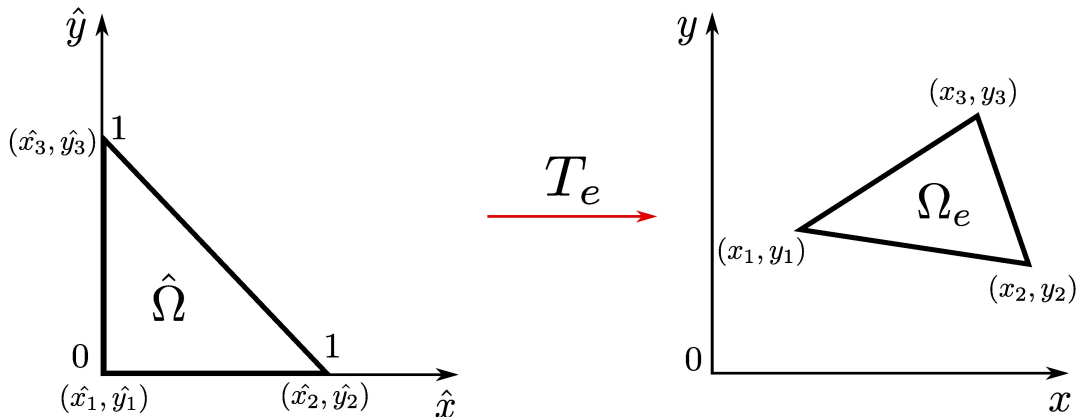


Figura 2.9: Transformação T_e através do elemento mestre.

Essa transformação é aplicada a todos os elementos da malha, garantindo que qualquer cálculo necessário possa ser realizado no elemento mestre $\hat{\Omega}$ e, em seguida, transferido para os elementos reais Ω_e . Além disso, propriedades fundamentais dos elementos finitos, como número e posição dos nós e funções de forma, podem ser definidas no elemento mestre e transportadas para a malha por meio das expressões:

$$\hat{\psi}(\hat{x}_i, \hat{y}_i) = \psi(T_e(\hat{x}_i, \hat{y}_i)) = \psi(x_i, y_i), \quad (2.29)$$

$$\nabla \psi(x_i, y_i) = B_{\Omega_e}^{-T} \cdot \hat{\nabla} \hat{\psi}(\hat{x}_i, \hat{y}_i), \quad (2.30)$$

Aqui $\hat{\psi}(\hat{x}_i, \hat{y}_i)$ são as funções de forma no elemento mestre, enquanto $\psi(x_i, y_i)$ são as funções de forma dos elementos da malha. Essas funções são fundamentais para gerar as funções base, que são usadas na formulação variacional do método. A referência completa sobre elemento mestre pode ser encontrada em [5; 18].

2.12 Montagem da matriz (*Assembly*)

Na prática, efetua-se todas as devidas transformações do elemento mestre para os elementos que compõem a malha e, em seguida, realizam-se os procedimentos de montagem da matriz $\mathbf{A}$ e dos vetores $\mathbf{U}$ e $\mathbf{F}$, que serão abordados nesta seção.

A fim de facilitar a exposição e cumprir os fins didáticos, considera-se um domínio unidimensional discretizado em n elementos Ω_e conforme ilustrado na Figura 2.10. À esquerda, observa-se o elemento mestre, enquanto à direita encontra-se a malha discretizada.

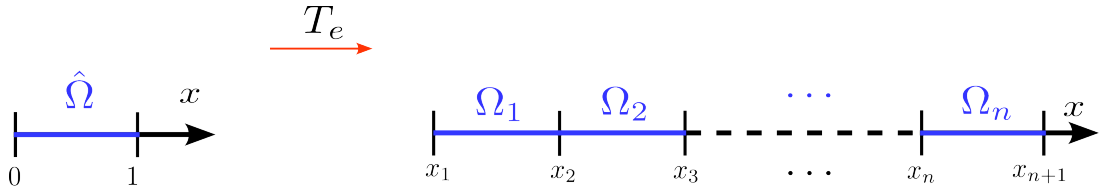


Figura 2.10: Representação do elemento mestre e o mapeamento T_e para o domínio 1D discretizado com n elementos.

Inicialmente considere o elemento mestre $\hat{\Omega}$, que possui dois nós ($x = 0$ e $x = 1$). Aplicando a equação (2.21) obtêm-se,

$$\begin{cases} \tilde{A}_{11}^{\hat{\Omega}} U_0^{\hat{\Omega}} + \tilde{A}_{12}^{\hat{\Omega}} U_1^{\hat{\Omega}} = \int_0^1 f(x) \phi_1(x) dx + h(0) \phi_1(0), \\ \tilde{A}_{21}^{\hat{\Omega}} U_0^{\hat{\Omega}} + \tilde{A}_{22}^{\hat{\Omega}} U_1^{\hat{\Omega}} = \int_0^1 f(x) \phi_2(x) dx - h(1) \phi_2(1), \end{cases} \quad (2.31)$$

onde $h(x) = -\mu(x)u'(x)$ representa o fluxo nas fronteiras do elemento. A notação $\tilde{A}_{ij}^{\hat{\Omega}}$ indica a componente da linha i e coluna j da matriz $\tilde{A}$ do elemento $\hat{\Omega}$ definida como $\tilde{A}_{ij}^{\hat{\Omega}} = a(\phi_i, \phi_j)^{\hat{\Omega}} = \int_{a^{\hat{\Omega}}}^b \mu(x) \phi_j'(x) \phi_i'(x) dx$. Os limites $a^{\hat{\Omega}}$ e $b^{\hat{\Omega}}$ correspondem, respectivamente, aos valores inicial e final do intervalo em que o domínio $\hat{\Omega}$ está definido.

Após as devidas transformações de coordenadas, estas equações são adaptadas para cada um dos n elementos Ω_e , permitindo a construção do sistema discreto completo por meio da união das equações de todos os elementos da malha.

Para ilustrar esse processo, considera-se a união das equações para dois elementos consecutivos, Ω_1 e Ω_2 . Para o elemento Ω_1 , tem-se:

$$\begin{cases} \tilde{A}_{11}^{\Omega_1} U_1 + \tilde{A}_{12}^{\Omega_1} U_2 = \int_{x_1}^{x_2} f(x)\phi_1(x) dx + h(x_1^+)\phi_1(x_1^+), \\ \tilde{A}_{21}^{\Omega_1} U_1 + \tilde{A}_{22}^{\Omega_1} U_2 = \int_{x_1}^{x_2} f(x)\phi_2(x) dx - h(x_2^-)\phi_2(x_2^-), \end{cases} \quad (2.32)$$

e para o elemento Ω_2 :

$$\begin{cases} \tilde{A}_{11}^{\Omega_2} U_2 + \tilde{A}_{12}^{\Omega_2} U_3 = \int_{x_2}^{x_3} f(x)\phi_2(x) dx + h(x_2^+)\phi_2(x_2^+), \\ \tilde{A}_{21}^{\Omega_2} U_2 + \tilde{A}_{22}^{\Omega_2} U_3 = \int_{x_2}^{x_3} f(x)\phi_3(x) dx - h(x_3^-)\phi_3(x_3^-). \end{cases} \quad (2.33)$$

O nó x_2 pertence simultaneamente aos elementos Ω_1 e Ω_2 , gerando uma contribuição para os dois elementos. Dessa forma, ao somar a segunda equação de Ω_1 com a primeira de Ω_2 , obtém-se:

$$\begin{cases} \tilde{A}_{11}^{\Omega_1} U_1 + \tilde{A}_{12}^{\Omega_1} U_2 = \int_{x_1}^{x_2} f(x)\phi_1(x) dx + h(x_1^+)\phi_1(x_1^+), \\ \tilde{A}_{21}^{\Omega_1} U_1 + (\tilde{A}_{22}^{\Omega_1} + \tilde{A}_{11}^{\Omega_2}) U_2 + \tilde{A}_{12}^{\Omega_2} U_3 = \int_{x_1}^{x_2} f(x)\phi_2(x) dx + \int_{x_2}^{x_3} f(x)\phi_2(x) dx - \\ \quad - h(x_2^-)\phi_2(x_2^-) + h(x_2^+)\phi_2(x_2^+), \\ \tilde{A}_{21}^{\Omega_2} U_2 + \tilde{A}_{22}^{\Omega_2} U_3 = \int_{x_2}^{x_3} f(x)\phi_3(x) dx - h(x_3^-)\phi_3(x_3^-). \end{cases} \quad (2.34)$$

Por construção (impondo continuidade entre os elementos) as funções peso ϕ_i aplicadas em x_2^- e x_2^+ são iguais, ou seja, $\phi_2(x_2^-) = \phi_2(x_2) = \phi_2(x_2^+)$. E note que $h(x_2^-)$ e $h(x_2^+)$ são respectivamente, a saída do fluxo no elemento Ω_1 e a entrada do fluxo em Ω_2 , e desde que não exista um termo fonte em x_2 , então $h(x_2^-) = h(x_2) = h(x_2^+)$ e portanto os termos se anulam na segunda equação do sistema (2.34), resultando:

$$\begin{cases} \tilde{A}_{11}^{\Omega_1} U_1 + \tilde{A}_{12}^{\Omega_1} U_2 = \int_{x_1}^{x_2} f(x)\phi_1(x) dx + h(x_1)\phi_1(x_1), \\ \tilde{A}_{21}^{\Omega_1} U_1 + (\tilde{A}_{22}^{\Omega_1} + \tilde{A}_{11}^{\Omega_2}) U_2 + \tilde{A}_{12}^{\Omega_2} U_3 = \int_{x_1}^{x_2} f(x)\phi_2(x) dx + \int_{x_2}^{x_3} f(x)\phi_2(x) dx, \\ \tilde{A}_{21}^{\Omega_2} U_2 + \tilde{A}_{22}^{\Omega_2} U_3 = \int_{x_2}^{x_3} f(x)\phi_3(x) dx - h(x_3)\phi_3(x_3). \end{cases}$$

De forma análoga, o mesmo procedimento é aplicado aos demais elementos da malha, conduzindo ao sistema completo (2.35). Observe que em todas as fronteiras onde existe intersecção de elementos, os termos se anulam, pois $h(x_i^-)\phi_i(x_i^-) = h(x_i^+)\phi_i(x_i^+)$, $i = 2, 3, \dots, n$, sobrando apenas $h(x_1)\phi_1(x_1)$ na primeira equação e $h(x_{n+1})\phi_{n+1}(x_{n+1})$ na última.

$$\begin{cases} \tilde{A}_{11}^{\Omega_1} U_1 + \tilde{A}_{12}^{\Omega_1} U_2 = \int_{x_1}^{x_2} f(x)\phi_1(x)dx + h(x_1)\phi_1(x_1), \\ \tilde{A}_{21}^{\Omega_1} U_1 + (\tilde{A}_{22}^{\Omega_1} + \tilde{A}_{11}^{\Omega_2}) U_2 + \tilde{A}_{12}^{\Omega_2} U_3 = \int_{x_1}^{x_2} f(x)\phi_2(x)dx + \int_{x_2}^{x_3} f(x)\phi_2(x)dx, \\ \vdots \\ \tilde{A}_{21}^{\Omega_{n-1}} U_{n-1} + (\tilde{A}_{22}^{\Omega_{n-1}} + \tilde{A}_{11}^{\Omega_n}) U_n + \tilde{A}_{12}^{\Omega_n} U_{n+1} = \int_{x_{n-1}}^{x_n} f(x)\phi_{x_n}(x)dx + \\ \quad + \int_{x_n}^{x_{n+1}} f(x)\phi_{x_n}(x)dx, \\ \tilde{A}_{21}^{\Omega_n} U_n + \tilde{A}_{22}^{\Omega_n} U_{n+1} = \int_{x_n}^{x_{n+1}} f(x)\phi_{x_{n+1}}(x)dx - \\ \quad - h(x_{n+1})\phi_{n+1}(x_{n+1}). \end{cases} \quad (2.35)$$

A matriz $\mathbf{A}$ obtida é tridiagonal e composta por blocos 2×2 , conforme ilustrado na Figura 2.11, onde cada bloco representa a contribuição de um elemento Ω_e . A subnotação ij de $\tilde{A}_{ij}^{\Omega_e}$ indica a posição local na matriz do elemento Ω_e , que pode diferir da posição global na matriz A_{ij} do sistema completo. Por exemplo, o elemento global A_{22} resulta da soma das contribuições dos elementos adjacentes, ou seja, $A_{22} = (\tilde{A}_{22}^{\Omega_1} + \tilde{A}_{11}^{\Omega_2})$. De

forma geral, com exceção da primeira e da última linha, todos os elementos da diagonal principal da matriz global A são influenciados pelos pontos internos e seus elementos adjacentes, ou seja, pode-se generalizar os termos da diagonal principal de A por $A_{kk} = \left(\tilde{A}_{22}^{\Omega_{k-1}} + \tilde{A}_{11}^{\Omega_k} \right)$, $k = 2, \dots, n$. Os elementos nulos são representados por zeros na Figura 2.11, evidenciando a esparsidade da matriz.

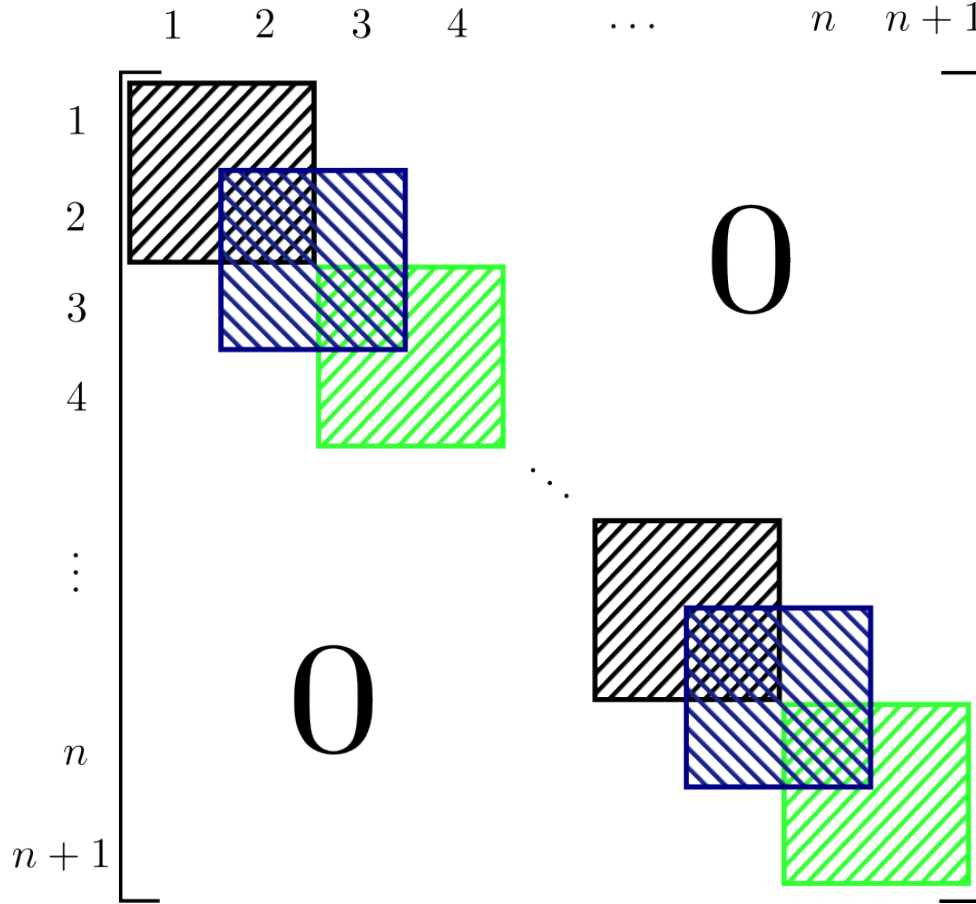


Figura 2.11: Formato da matriz A .

Maiores detalhes da montagem da matriz podem ser vistos no livro [5].

2.13 Estimativa do erro

Nesta seção será apresentada uma síntese sobre a estimativa do erro, baseada nas ideias e abordagens apresentadas no livro [5].

Seja $u : \Omega \subset \mathbb{R} \rightarrow \mathbb{R}$, que satisfaça a equação (2.36) do tipo:

$$-\frac{d}{dx} \left(\mu(x) \frac{du(x)}{dx} \right) + c(x) \frac{du(x)}{dx} + b(x)u(x) = f(x) \quad x \in \Omega, \quad (2.36)$$

em que esta equação (2.36) faça parte de um problema bem posto, suponha que a solução analítica u tenha as propriedades de que suas derivadas de ordem s sejam quadrado-integrável em Ω , enquanto as derivadas de ordem $s+1$ ou superiores não são. Aqui, s é um inteiro maior que uma unidade. Suponha ainda que as funções de forma ψ utilizadas sejam compostas por polinômios de grau no máximo k . Tomando $q = \min(k, s)$ e considerando uma malha uniforme de elementos com comprimento h , então o erro de aproximação medido pela norma H^1 satisfaz:

$$\| u - u_h \|_{H^1} \leq C h^q, \quad (2.37)$$

onde C é uma constante independente de h , u representa a solução analítica do problema e u_h a solução aproximada do problema discretizado.

Já o erro de aproximação com respeito à norma L^2 é dado por:

$$\| u - u_h \|_{L^2} \leq C_1 h^{q+1}, \quad (2.38)$$

onde C_1 é uma constante independente de h . Assim, quando a solução u é regular ($s > k$), é possível aumentar a taxa de convergência ao aumentar o grau do polinômio de aproximação até $k < s$, uma vez que $q = \min(k, s) = k < s$. Quando $k \geq s$, o aumento do grau polinomial não afetará a taxa de convergência, pois $q = \min(k, s) = s \leq k$.

Para o caso em que s tende a infinito, então $q = \min(k, s) = k$, $\forall k \in \mathbb{N}$ e portanto, o aumento do grau do polinômio de aproximação sempre aumentará a taxa de convergência.

Para o caso bidimensional, considere $h = \max\{h_e\}$, onde h_e (denominado diâmetro) é a maior distância entre dois nós de um elemento Ω_e . Neste caso, as estimativas dos erros (2.37) e (2.38) também são válidas, desde que u_h contenha um polinômio completo de grau k . Por exemplo, se $g_h = a_0 + a_1x + a_2y + a_3xy$, ele contém um termo quadrático xy , sendo um polinômio completo de grau $k = 1$ (mas não de grau $k = 2$). Assim, a estimativa do erro é de ordem $O(h)$.

2.14 Tratamento da evolução temporal

Até o momento, foram discutidos problemas nos quais a função incógnita u depende unicamente da posição espacial $\mathbf{x}$, ou seja, $u(\mathbf{x})$, com $\mathbf{x} \in \Omega \subset \mathbb{R}^d$, $d = 1, 2$ ou 3 . Nesta seção, será apresentada uma técnica para resolver problemas transientes, nos quais a função u passa a depender também do tempo t , ou seja, $u(\mathbf{x}, t)$, em que $t \in [0, T]$ e $T \in \mathbb{R}_+^*$ representa o tempo final do domínio temporal.

A técnica adotada para tratar de problemas transientes é composta por dois estágios de aproximação, conforme descrito nas obras [5; 6]. O primeiro estágio consiste de uma discretização espacial, baseada nas ideias dos problemas estacionários vistos na seção 2.5, o que resultará em um sistema de Equações Diferenciais Ordinárias (EDO) na variável t . O segundo estágio, denominado aproximação temporal, utiliza esquemas de diferenças finitas para integração temporal destas EDOs.

A seguir, esta técnica será aplicada à equação do calor transiente com os objetivos de exemplificar e discutir cada um destes estágios.

2.14.1 Equação do calor

Considere o problema modelado pela equação do calor transiente com termo fonte sujeita a uma condição inicial e condições de contorno sobre as fronteiras, Γ_D e Γ_N , do domínio espacial $\Omega \subset \mathbb{R}^d$, $d = 1, 2$ ou 3 , descrito por

$$\left\{ \begin{array}{ll} \beta \frac{\partial u(\mathbf{x}, t)}{\partial t} - \nabla \cdot (\mu \nabla u(\mathbf{x}, t)) &= f(\mathbf{x}, t) \quad , \quad \mathbf{x} \in \Omega, \\ u(\mathbf{x}, t) &= g, \quad \mathbf{x} \in \Gamma_D, \\ -\mu u'(\mathbf{x}, t) &= h(t), \quad \mathbf{x} \in \Gamma_N, \\ u(\mathbf{x}, 0) &= u_0(\mathbf{x}), \quad \mathbf{x} \in \Omega, \end{array} \right. \quad (2.39)$$

onde $u_0 : \Omega \rightarrow \mathbb{R}$ é uma condição inicial fornecida, $g \in \mathbb{R}$ e $h : [0, T] \rightarrow \mathbb{R}$ são também conhecidos e descrevem, respectivamente, as condições de contorno de Dirichlet sobre Γ_D

e de Neumann sobre Γ_N , $f : \Omega \times [0, T] \rightarrow \mathbb{R}$ representa o termo fonte da equação do calor, β e $\mu \in \mathbb{R}$ são duas constantes positivas, $t \in [0, T]$ é a variável independente temporal, $T \in \mathbb{R}_+^*$ representa o tempo final a ser estabelecido, e $u : \Omega \times [0, T] \rightarrow \mathbb{R}$ é a função incógnita, ou simplesmente, a variável do problema.

O problema transiente (2.39) será aproximado em dois estágios, como descrito a seguir.

Primeiro estágio: inicia-se o primeiro estágio pelo procedimento análogo da seção 2.5 para obtenção da forma fraca do problema (2.39) multiplicando os dois lados da equação transiente do calor por $v(\mathbf{x})$ e aplicando a integral sobre o domínio Ω ,

$$\int_{\Omega} \beta \frac{\partial u(\mathbf{x}, t)}{\partial t} v(\mathbf{x}) d\mathbf{x} - \int_{\Omega} (\nabla \cdot (\mu \nabla u(\mathbf{x}, t))) v(\mathbf{x}) d\mathbf{x} = \int_{\Omega} f(\mathbf{x}, t) v(\mathbf{x}) d\mathbf{x}. \quad (2.40)$$

Note que, mesmo para o problema transiente a função peso v é independente do tempo.

Após aplicar o Teorema da Divergência e as condições de contorno como no processo da formulação variacional da seção 2.5, obtém-se,

$$\int_{\Omega} \beta \frac{\partial u(\mathbf{x}, t)}{\partial t} v(\mathbf{x}) d\mathbf{x} + \int_{\Omega} \mu \nabla u(\mathbf{x}) \cdot \nabla v(\mathbf{x}) d\mathbf{x} = \int_{\Omega} f(\mathbf{x}, t) v(\mathbf{x}) d\mathbf{x} - \int_{\Gamma_N} h(t) v(\mathbf{x}) d\Gamma_N. \quad (2.41)$$

A equação (2.41) fornecerá uma solução aproximada u_h do método de elementos finitos sobre um domínio espacial discreto. Para isto, assume-se que para a função aproximação u_h a dependência temporal pode ser separada da dependência espacial, como,

$$u_h(\mathbf{x}, t) = \sum_{j=1}^{n_u} U_j(t) \phi_j(\mathbf{x}), \quad (2.42)$$

onde n_u representa o número de graus de liberdade associados à variável u e U_j os graus de liberdade que agora são funções dependentes do tempo $U_j(t)$, enquanto as funções base ϕ_j permanecem dependentes apenas da variável espacial $\mathbf{x}$.

Dessa forma, obtém-se a semi-discretização da equação (2.41), que é o sistema de equações diferenciais ordinárias (2.43) cuja i -ésima equação resultante é representada da seguinte forma,

$$\sum_{j=1}^{n_u} \left(M_{ij} \frac{dU_j(t)}{dt} + K_{ij} U_j(t) \right) = F_i - H_i, \quad i, j = 1, 2, \dots, n_u, \quad (2.43)$$

onde,

$$\begin{aligned} M_{ij} &= \int_{\Omega} \beta \phi_j(\mathbf{x}) \phi_i(\mathbf{x}) d\mathbf{x}, \\ K_{ij} &= \int_{\Omega} \mu \phi_j'(\mathbf{x}) \phi_i'(\mathbf{x}) d\mathbf{x}, \\ F_i &= \int_{\Omega} f(\mathbf{x}, t) \phi_i(\mathbf{x}) d\mathbf{x}, \\ H_i &= h(t) \phi_i(b). \end{aligned}$$

A equação (2.43) pode ser escrita na forma matricial como,

$$\mathbf{M} \dot{\mathbf{U}} + \mathbf{K} \mathbf{U} = \mathbf{F} - \mathbf{H}, \quad (2.44)$$

definindo $\dot{\mathbf{U}} = \frac{d\mathbf{U}}{dt}$.

Segundo estágio: a integração temporal da equação diferencial ordinária (2.44) pode ser realizada por meio da família α de aproximação, definida por,

$$\mathbf{U}_{k+1} = \mathbf{U}_k + \Delta t[(1 - \alpha)\dot{\mathbf{U}}_k + \alpha\dot{\mathbf{U}}_{k+1}], \quad 0 \leq \alpha \leq 1, \quad (2.45)$$

onde $\Delta t = \frac{T}{m}$ é o tamanho do passo temporal que define a discretização do intervalo $[0, T]$ em m subintervalos, sendo $t_0 = 0$ o tempo inicial, $t_{k+1} = t_k + \Delta t$ para $k = 0, 1, \dots, m-2$ os tempos intermediários, e $t_m = T$ o tempo final; e, cada parâmetro α escolhido determina um esquema de aproximação diferente.

No presente trabalho serão adotados dois esquemas clássicos: o método de Euler explícito ($\alpha = 0$) e o método de Euler implícito ($\alpha = 1$).

Exemplificando cada um destes esquemas. Para $\alpha = 0$, método explícito, a derivada de u em relação ao tempo é aproximada da forma:

$$\dot{\mathbf{U}} \approx \frac{\mathbf{U}_{k+1} - \mathbf{U}_k}{\Delta t}, \quad k = 0, 1, \dots, m-1. \quad (2.46)$$

Então, ao substituir (2.46) em (2.43), obtém-se:

$$\begin{aligned} \mathbf{M} \left(\frac{\mathbf{U}_{k+1} - \mathbf{U}_k}{\Delta t} \right) + \mathbf{K} \mathbf{U}_k &= \mathbf{F}_k - \mathbf{H}_k, \\ \Rightarrow \mathbf{M} (\mathbf{U}_{k+1} - \mathbf{U}_k) + \Delta t \mathbf{K} \mathbf{U}_k &= \Delta t \mathbf{F}_k - \Delta t \mathbf{H}_k, \\ \Rightarrow \mathbf{M} \mathbf{U}_{k+1} = \mathbf{M} \mathbf{U}_k - \Delta t \mathbf{K} \mathbf{U}_k + \Delta t \mathbf{F}_k - \Delta t \mathbf{H}_k, \\ \Rightarrow \mathbf{M} \mathbf{U}_{k+1} = (\mathbf{M} - \Delta t \mathbf{K}) \mathbf{U}_k + \Delta t \mathbf{F}_k - \Delta t \mathbf{H}_k, \\ \Rightarrow \mathbf{U}_{k+1} = \mathbf{M}^{-1} [(\mathbf{M} - \Delta t \mathbf{K}) \mathbf{U}_k + \Delta t \mathbf{F}_k - \Delta t \mathbf{H}_k], \end{aligned} \quad (2.47)$$

para cada $k = 0, 1, \dots, m-1$.

A expressão do lado direito da equação (2.47) contém matrizes com elementos conhecidos e integrais que podem ser calculadas, logo a única incógnita da equação é o vetor $\mathbf{U}_{k+1}$, que será determinado a cada passo no tempo.

Se a opção for pelo método implícito, $\alpha = 1$, a derivada de u em relação ao tempo é aproximada da forma,

$$\dot{\mathbf{U}} \approx \frac{\mathbf{U}_k - \mathbf{U}_{k-1}}{\Delta t}, \quad (2.48)$$

Procedendo de maneira similar, ao substituir (2.48) em (2.43) e efetuar as manipulações algébricas, obtém-se,

$$(\mathbf{M} + \Delta t \mathbf{K}) \mathbf{U}_{k+1} = \mathbf{M} \mathbf{U}_k + \Delta t \mathbf{F}_{k+1} - \Delta t \mathbf{H}_{k+1}, \quad (2.49)$$

para cada $k = 0, 1, \dots, m-1$.

Observe que apesar das matrizes $\mathbf{F}_{k+1}$ e $\mathbf{H}_{k+1}$ estarem no tempo t_{k+1} , seus elementos são funções conhecidas para todo $t \in [0, T]$ e portanto, a única incógnita da equação (2.49) é o vetor $\mathbf{U}_{k+1}$, que será determinado a cada passo no tempo ao resolver esse sistema linear.

2.14.2 Estabilidade numérica

A análise de estabilidade do método (apresentada em [6]) é um aspecto crucial para avaliar o comportamento numérico da aproximação. Foram apresentados dois métodos para a aproximação temporal: um explícito e um implícito. Os métodos implícitos são conhecidos por serem incondicionalmente estáveis, enquanto os explícitos apresentam estabilidade apenas sob certas condições, ou seja, são condicionalmente estáveis.

A condição de estabilidade para os métodos explícitos, é dada por:

$$\Delta t < \frac{2}{(1 - 2\alpha)\lambda_{max}}, \quad \alpha < \frac{1}{2}, \quad (2.50)$$

onde λ_{max} é o maior autovalor do problema de autovalor associado à equação matricial (2.44):

$$|(\mathbf{M} + \alpha\Delta t\mathbf{K}) - \lambda\mathbf{I}| = 0. \quad (2.51)$$

Os autovalores estão relacionados ao sistema de matrizes $\mathbf{M}$ e $\mathbf{K}$. Pode-se demonstrar que:

$$\lambda_{max} \sim \frac{\mu}{\beta h_{min}^2}, \quad (2.52)$$

em que h_{min} é a dimensão característica do menor elemento finito na malha. Considere $h_{min} = \min\{h_{\Omega_e}\}$, $e = 1, \dots, n$.

Portanto, o tamanho do passo no tempo é limitado à:

$$\Delta t < \frac{2\beta h_{min}^2}{(1 - 2\alpha)\mu}. \quad (2.53)$$

2.14.3 Equação de Advecção-Difusão

Com vistas na aplicação final deste trabalho, esta seção será dedicada a apresentação da formulação em elementos finitos para uma equação de advecção-difusão.

Dado um problema de advecção-difusão transiente com uma dimensão espacial, deseja-se calcular $u : \Omega \rightarrow \mathbb{R}$, $\Omega = (a, b)$ tal que:

$$\left\{ \begin{array}{l} \underbrace{\beta \frac{\partial u(x, t)}{\partial t}}_{\text{termo transiente}} + \underbrace{\mathcal{U} \frac{\partial u(x, t)}{\partial x}}_{\text{termo advectivo}} - \underbrace{\frac{\partial}{\partial x} \left(\mu(x) \frac{\partial u(x, t)}{\partial x} \right)}_{\text{termo difusivo}} = f(x, t) \quad , \quad x \in \Omega, \\ u(a, t) = g, \quad t \in [0, T], \\ -\mu(b)u'(b, t) = h(t), \quad t \in [0, T], \\ u(x, 0) = u_0(x), \quad x \in \Omega, \end{array} \right. \quad (2.54)$$

onde $u_0 : \Omega \rightarrow \mathbb{R}$ é a condição inicial, $g \in \mathbb{R}$ e $h : [0, T] \rightarrow \mathbb{R}$ as condições de contorno, $f : \Omega \times [0, T] \rightarrow \mathbb{R}$ o termo fonte, $\mu : \Omega \rightarrow \mathbb{R}$ a função coeficiente de difusão, $\beta, \mathcal{U} \in \mathbb{R}$ duas constantes e $t \in [0, T]$ é a variável independente temporal, $T \in \mathbb{R}_+^*$.

A resolução do problema (2.54) segue um procedimento semelhante ao problema da equação do calor (2.39), mas agora inclui um termo advectivo no lado esquerdo da equação. A semi-discretização espacial deste termo é dada pela seguinte expressão:

$$J_{ij} = \mathcal{U} \int_a^b \phi_j'(x) \phi_i(x) dx. \quad (2.55)$$

Em seguida, ao escolher o método de Euler explícito para a aproximação temporal, o vetor $\mathbf{U}_{N+1}$ é calculado por meio da fórmula matricial (2.56).

$$\mathbf{U}_{N+1} = \mathbf{M}^{-1} [(\mathbf{M} - \Delta t \mathbf{J} - \Delta t \mathbf{K}) \mathbf{U}_N + \Delta t \mathbf{F}_N - \Delta t \mathbf{H}_N]. \quad (2.56)$$

No entanto, caso a escolha seja pelo método de Euler implícito para a aproximação temporal, o vetor $\mathbf{U}_{N+1}$ é calculado por meio do sistema linear (2.57) escrito na forma matricial.

$$(\mathbf{M} + \Delta t \mathbf{J} + \Delta t \mathbf{K}) \mathbf{U}_{N+1} = \mathbf{M} \mathbf{U}_N + \Delta t \mathbf{F}_{N+1} - \Delta t \mathbf{H}_{N+1}. \quad (2.57)$$

Um exemplo numérico clássico de aplicação desta metodologia será apresentado em detalhes no Capítulo 3, seção 3.4 deste trabalho.

Resultados Numéricos Utilizando FEniCSx

A plataforma FEniCSx tem se mostrado uma excelente ferramenta computacional, robusta e acessível para a implementação do método de elementos finitos, permitindo a formulação e solução de problemas complexos por meio de uma linguagem simbólica intuitiva e uma sintaxe clara baseada na linguagem de programação PythonTM [14]. Neste capítulo, exploramos suas capacidades por meio da resolução de diferentes classes de equações diferenciais parciais, destacando sua flexibilidade e eficiência na modelagem numérica. Além disso, seu uso está alinhado aos objetivos de uma formação em Matemática Aplicada e Computacional, proporcionando uma abordagem prática para a solução de problemas matemáticos.

Para esse fim, são discutidos problemas fundamentais que ilustram diferentes aspectos do método, abrangendo equações elíptica, parabólica e equações mistas, através da equação de advecção-difusão, que são importantes na modelagem de escoamentos não-isotérmicos em meios porosos. A implementação desses exemplos no FEniCSx fornece um panorama abrangente de sua aplicação e serve como referência para desenvolvimentos mais avançados. Os códigos implementados são disponibilizados no Apêndice, proporcionando um material prático para consulta e experimentação.

3.1 Equação de Poisson

Considere um problema envolvendo o escoamento de um fluido incompressível no estado estacionário modelado pela equação de Poisson com domínio $\Omega \subset \mathbb{R}^d$, $d = 1, 2$ ou 3 (onde cada componente de $\mathbf{x}$ está em metros), descrito por:

$$\begin{cases} -\nabla \cdot (\mu(\mathbf{x}) \nabla u(\mathbf{x})) &= f(\mathbf{x}) \quad , \quad \mathbf{x} \in \Omega \quad , \\ u(\mathbf{x}) &= g(\mathbf{x}) \quad , \quad \mathbf{x} \in \Gamma_D, \end{cases} \quad (3.1)$$

onde $f : \Omega \rightarrow \mathbb{R}$ representa o gradiente de pressão (N/m^3), $g : \Gamma_D \rightarrow \mathbb{R}$ a condição de Dirichlet sobre Γ_D , $\mu : \Omega \rightarrow \mathbb{R}$ a viscosidade do fluido ($Pa \cdot s$) e $u : \Omega \rightarrow \mathbb{R}$ a velocidade (m/s).

3.1.1 Problemas 1D

Para o problema unidimensional, considere $\Omega = [0, 1]$, $\Gamma_D = \{0, 1\}$, $g(0) = g(1) = 0m/s$, $f(x) = 2N/m^3$, $\forall x \in \Omega$ e $\mu(x) = 1Pa \cdot s$, $\forall x \in \Omega$. Será mostrado a seguir três

formas de resolver a equação de Poisson (3.1) em PythonTM, todas pelo método de elementos finitos utilizando a plataforma Google Colaboratory (ou simplesmente, Colab). A primeira forma de programação foi utilizando as bibliotecas NumPy & SymPy; a segunda utilizando Firedrake e a terceira FEniCSx. Assim como o FEniCSx, o Firedrake também é uma plataforma usada para solucionar equações diferenciais parciais pelo método de elementos finitos, baseada em PythonTM. Maiores informações desta ferramenta podem ser vistas através do seu manual [9].

3.1.1.1 Programação utilizando NumPy & SymPy

A primeira implementação foi feita utilizando as bibliotecas NumPy & SymPy.

```

1 #----- IMPORTANDO BIBLIOTECAS -----
2 import numpy as np
3 import numpy.linalg as la
4 from sympy import *
5 from sympy import Function as Func
6 from sympy import dsolve, Eq, Derivative
7 from sympy import diff
8 from sympy import Matrix
9 from sympy.abc import x, y, z #x is the independent variable
10 init_printing(pretty_print=True)
11 from sympy.plotting import plot
12 import matplotlib.pyplot as plt
13 from google.colab import files
14
15 #----- DADOS DE ENTRADA -----
16 X_i = 0
17 X_f = 1
18 n = 3 #quantidade de elementos
19 u_I = 0; #condição de contorno do tipo Dirichlet u(0)
20 u_F = 0; #condição de contorno do tipo Dirichlet u(1)
21 f = 2; #Função f
22
23 #----- CRIAÇÃO DE VARIÁVEIS E MATRIZES -----
24 X = np.linspace(0,X_f, n+1) #ATENÇÃO, trata-se do X maiúsculo
25 h = (X_f-X_i)/n #considerando todos os espaçamentos h_i = h
26
27 phi = zeros(n+1, n) #teste de phi sendo criado como matriz pela biblioteca
    sympy
28         #phi(num_phi_i, intervalo_i)
29 A_old = np.zeros((n+1, n+1)) #matriz A completa
30 F_old = np.zeros(n+1) #F completo
31 U      = np.zeros(n+1)
32 U[0]   = u_I;
33 U[-1]  = u_F;
34 u_h    = zeros(n,1)
35
36 #----- FUNÇÃO PHI -----
37 for i in range(1,n):
38     phi[i,i-1] = (x-X[i-1])/h
39     phi[i,i]   = (X[i+1]-x)/h
40
41 phi[0,0] = (X[1]-x)/h
42 phi[-1,-1] = (x-X[-2])/h
43
44 #----- MATRIZ A -----
45 for i in range(n+1): #linha
46     for j in range(n+1): #coluna

```

```

47     for k in range(n): #elemento
48         A_old[i][j] += integrate( diff(phi[j,k], x)*diff(phi[i,k], x), (x, X[
         k], X[k+1]))
49 A = A_old[1:n, 1:n] #Matriz A "reduzida".
50
51 #----- VETOR F -----
52 for i in range(n+1):
53     for j in range(n):
54         F_old[i] += integrate(f*phi[i,j], (x, X[j], X[j+1]))
55 F = F_old[1:n] #F "reduzido"
56 #Condição de contorno
57 F[0] += -A_old[1][0]*U[0] - A_old[1][-1]*U[-1]
58 F[1] += -A_old[2][0]*U[0] - A_old[2][-1]*U[-1]
59
60 #----- CALCULO SOLUÇÃO U -----
61 U[1:-1] = la.solve(A, F)
62
63 #----- CALCULO SOLUÇÃO u_h -----
64 for k in range(n): #k é o número do elemento
65     for i in range(n+1):
66         u_h[k] += U[i]*phi[i,k]
67
68 #----- PLOTAGEM -----
69 fig, ax = plt.subplots(figsize=(8,6))
70
71 Analitica = plot(Sol_analitica, (x, 0, 1), show=False, line_color='black',
72                 label=r'Solução exata u(x)')
73 move_sympy_plot_to_axes(Analitica, ax)
74 for k in range(n):
75     move_sympy_plot_to_axes(plot(u_h[k], (x, X[k], X[k+1])), show=False,
76                             line_color='blue', ls='-', label=r'Solução aproximada u_h, no intervalo
77                             $h_{%d}$'%k), ax)
78
79 ax.get_lines()[0].set(linestyle='—') #config. estilo de linha da sol.
80     analitica
81 ax.get_lines()[1].set(linestyle='-') #config. estilo de linha da sol.
82     aproximada
83
84 ax.set_title('Equação de Poisson – Método Elementos Finitos')
85 ax.legend(['Solução exata', 'Solução aproximada'])
86 ax.set_xlabel('x')
87 ax.set_ylabel('u(x)')
88 ax.grid()
89
90 plt.savefig('Poisson_MEF.png', transparent=True, dpi=300); #gerando arquivo
91     no google colab
92 files.download('Poisson_MEF.png') #fazendo download do arquivo gerado no
93     google colab
94 plt.show()

```

Na linha 49 a matriz A recebe os valores de A_{old} com exceção das primeiras e últimas linhas e colunas, pois os valores de $U[0]$ e $U[n]$ já são conhecidos. Então o sistema $AU = F$ é a forma matricial que representa o sistema linear com $n - 1$ equações.

A Figura 3.1 mostra o gráfico das funções que representam as soluções aproximada e exata.

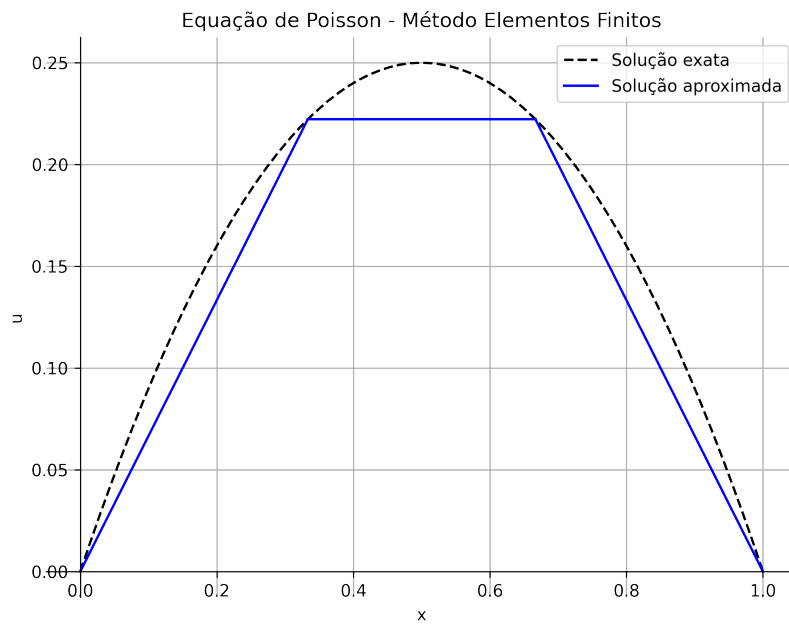


Figura 3.1: Distribuição da velocidade utilizando as bibliotecas NumPy & SymPy.

3.1.1.2 Programação em Python™ utilizando Firedrake

O segundo código foi feito utilizando o Firedrake.

O código para solucionar o problema de Poisson utilizando a biblioteca Firedrake foi escrito da forma:

```

1 # ----- IMPORTANDO BIBLIOTECAS -----
2 import os
3 arch = os.getenv("ARGS", "real")
4
5 try:
6     import google.colab # noqa: F401
7 except ImportError:
8     import firedrake
9 else:
10     try:
11         import firedrake
12     except ImportError:
13         if arch != "complex":
14             !wget "https://fem-on-colab.github.io/releases/firedrake-
install-release-real.sh" -O "/tmp/firedrake-install.sh" && bash "/tmp/
firedrake-install.sh"
15         else:
16             !wget "https://fem-on-colab.github.io/releases/firedrake-
install-release-complex.sh" -O "/tmp/firedrake-install.sh" && bash "/tmp
/firedrake-install.sh"
17         import firedrake
18
19 import numpy as np
20 from google.colab import files
21 from petsc4py import PETSc
22 if arch != "complex":
23     assert not np.issubdtype(PETSc.ScalarType, np.complexfloating)
24 else:

```

```

25     assert np.issubdtype(PETSc.ScalarType, np.complexfloating)
26
27 #----- IMPLEMENTAÇÃO DO CÓDIGO -----
28
29 #----- CRIAÇÃO DA MALHA -----
30 mesh = firedrake.UnitIntervalMesh(3) #definindo a malha com 3 elementos
    finitos
31
32 V = firedrake.FunctionSpace(mesh, "CG", 1) #definindo o Espaço utilizado
33 #CG = continuous Galerkin (Espaços polinomiais por partes, contínuos també
    m entre elementos), Grau do polinômio = 1
34 assert V.dim() == 4 #verifica se a dimensão do espaço é 4.
35
36 #Criando as funções u e v
37 u = firedrake.TrialFunction(V) #função u
38 v = firedrake.TestFunction(V) #função v
39
40 inner = firedrake.inner #define produto escalar
41 dx = firedrake.dx #Calcula a integral sobre o domínio Omega. dx tem todas
    as quadraturas.
42
43 f = firedrake.Function(V) #função
44 f.vector()[ :] = 2 #neste caso é uma função constante que vale 2. #np.arange
    (1, V.dim() + 1)
45
46 grad = firedrake.grad
47 a = inner(grad(u), grad(v)) * dx #cálculo do lado esquerdo da eq do
    problema. a_h(\phi_i, \phi_j)
48 F = inner(f, v) * dx #cálculo do lado direito da eq do problema
    . l(\phi_i)
49
50 import scipy
51 import scipy.sparse as sp
52 #----- MONTANDO SISTEMA LINEAR Ax = b -----
53 A = firedrake.assemble(a) #monta a matriz A
54 b = firedrake.assemble(F) #monta o vetor b
55 petsc_mat = A.M.handle
56 petsc_vec = b.vector
57 print(petsc_mat)
58 indptr, indices, data = petsc_mat.getValuesCSR()
59 scipy_mat = sp.csr_matrix((data, indices, indptr), shape=petsc_mat.getSize
    ())
60 print(scipy_mat)
61
62 #----- CONDIÇÕES DE CONTORNO – DIRICHLET -----
63 boundary_value = 0
64 bc = firedrake.DirichletBC(V, boundary_value, "on_boundary") #on_boundary =
    em toda a fronteira ou usar [1,2] #1 esquerda, 2 direita
65
66 #----- SOLUÇÃO APROXIMADA -----
67 solution = firedrake.Function(V)
68 poisson_problem = firedrake.LinearVariationalProblem(a, F, solution, bcs=bc
    )#ao invés de firedrake.solve(A, solution.vector(), b)
69 poisson_solver = firedrake.LinearVariationalSolver(poisson_problem)
70 poisson_solver.solve()
71
72 #----- SOLUÇÃO EXATA PARA COMPARAÇÃO -----
73 mesh_ex = firedrake.UnitIntervalMesh(100) #criando a mesh
74 V_ex = firedrake.FunctionSpace(mesh_ex, "CG", 1) #para sol. exata
75 x, = firedrake.SpatialCoordinate(mesh_ex)

```

```

76 u_exata = -x**2+x
77 exact_solution = firedrake.Function(V_ex).interpolate(u_exata)
78 exact_solution.rename("Solução Exata", "label")
79 x_values = mesh_ex.coordinates.vector().dat.data
80 exact_solution_values = exact_solution.vector().dat.data
81
82 # ----- PLOTAGEM -----
83 import matplotlib.pyplot as plt
84 from firedrake.pyplot import plot
85 plt.figure(figsize=(7,5))
86
87 plot(solution, color = 'blue', label = "Solução aproximada")
88 plt.plot(x_values, exact_solution_values, 'k—', label = "Solução Exata")
89 plt.grid()
90 plt.legend(loc="best")
91 plt.xlabel("x")
92 plt.ylabel(r"$u_h$")
93 plt.savefig('Poisson_MEF_Firedrake.png', transparent=True, dpi=300); #
94   gerando arquivo no google colab
95 files.download('Poisson_MEF_Firedrake.png') #fazendo download do arquivo
96   gerado no google colab
97 plt.show()

```

A Figura 3.2 mostra as soluções aproximada e exata.

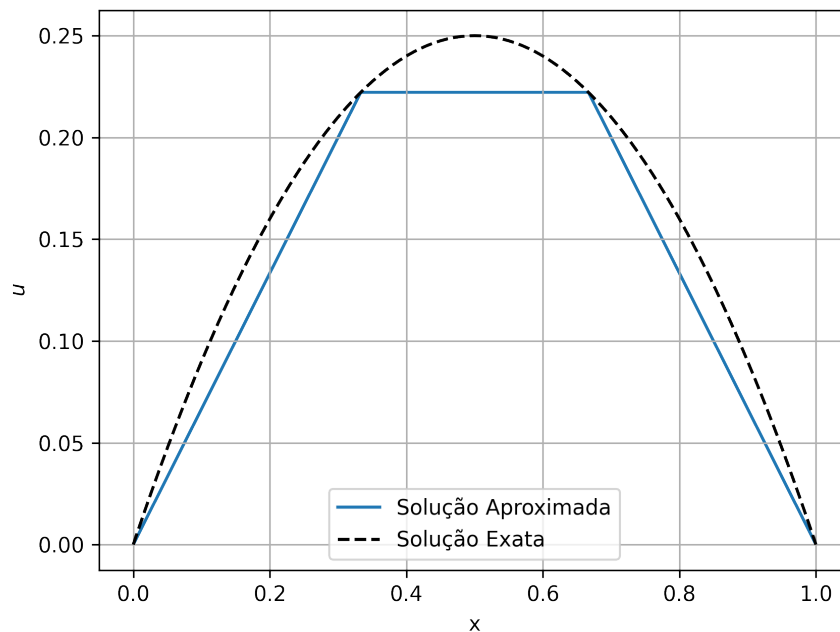


Figura 3.2: Distribuição da velocidade utilizando as bibliotecas Firedrake.

3.1.1.3 Programação em PythonTM utilizando FEniCSx

O FEniCSx é uma ferramenta computacional com código aberto utilizada para solucionar equações diferenciais parciais utilizando o método de elementos finitos. As principais publicações dos desenvolvedores dessa ferramenta são [10; 11; 12; 13].

O código criado em PythonTM para solucionar o problema utilizando o FEniCSx foi escrito da forma:

```

1 # ----- IMPORTANDO BIBLIOTECAS -----

```

```

2 import os
3 arch = os.getenv("ARGS", "real")
4 try:
5     import google.colab # noqa: F401
6 except ImportError:
7     import ufl
8     import dolfinx
9 else:
10     try:
11         import ufl
12         import dolfinx
13     except ImportError:
14         if arch != "complex":
15             !wget "https://fem-on-colab.github.io/releases/fenicsx-install-release-real.sh" -O "/tmp/fenicsx-install.sh" && bash "/tmp/fenicsx-install.sh"
16         else:
17             !wget "https://fem-on-colab.github.io/releases/fenicsx-install-release-complex.sh" -O "/tmp/fenicsx-install.sh" && bash "/tmp/fenicsx-install.sh"
18         import ufl
19         import dolfinx
20 import numpy as np
21 from google.colab import files
22 import pandas as pd
23 import petsc4py
24 import petsc4py.PETSc
25 if arch != "complex":
26     assert not np.issubdtype(petsc4py.PETSc.ScalarType, np.complexfloating)
27 else:
28     assert np.issubdtype(petsc4py.PETSc.ScalarType, np.complexfloating)
29 from dolfinx import mesh, fem, io, plot
30 import dolfinx.fem.petsc
31 from ufl import SpatialCoordinate, TestFunction, TrialFunction, Measure, dx, ds, grad, inner, as_vector, FacetNormal, dot
32 from dolfinx.fem import assemble_matrix
33 import mpi4py
34 import mpi4py.MPI
35 from mpi4py import MPI
36 from petsc4py.PETSc import ScalarType
37 import matplotlib.pyplot as plt
38 import scipy.sparse
39 import time
40 from slepc4py import SLEPc
41
42 #----- IMPLEMENTAÇÃO DO CÓDIGO -----
43
44 def solucao_MEF_1D(n, uleft, uright, degree, Lx, mu_value, f_value):
45
46     msh = mesh.create_unit_interval(mpi4py.MPI.COMM_WORLD, n) #Malha
47     V = fem.functionspace(msh, ("CG", degree)) #Espaço de funções continuous Galerkin
48
49 #----- IDENTIFICAÇÃO DOS GRAUS DE LIBERDADE -----
50     dofsL = fem.locate_dofs_geometrical(V, lambda x: np.isclose(x[0], 0))
51     dofsR = fem.locate_dofs_geometrical(V, lambda x: np.isclose(x[0], Lx))
52
53 #----- CONDIÇÃO DE CONTORNO -----
54     bcs = [fem.dirichletbc(ScalarType(uleft), dofsL, V), fem.dirichletbc(ScalarType(uright), dofsR, V)]

```

```

55
56 #----- FORMULAÇÃO VARIACIONAL -----
57
58 # Funções Trial and test
59 u = TrialFunction(V)
60 v = TestFunction(V)
61
62 #----- FORMA BILINEAR -----
63 mu = fem.Constant(msh, mu_value)
64 a = inner(mu*grad(u), grad(v)) * dx
65
66 #----- FORMA LINEAR -----
67 f = f_value
68 source = fem.Constant(msh, f)
69 L = (source * v) * dx
70
71 #----- SOLUCIONANDO O PROBLEMA -----
72 problem = fem.petsc.LinearProblem(a, L, bcs=bcs, petsc_options={"
    ksp_type": "preonly", "pc_type": "lu"})
73 uh = problem.solve()
74
75
76 #----- SOLUÇÃO ANALÍTICA COM MAIS PONTOS PARA PLOTAGEM -----
77 n_plot = n*30
78 x_plot_ex = np.linspace(0,Lx, n_plot+1) #x da sol. exata criado com mais
    pontos
79 uex1 = lambda x1: uleft + ( (uright - uleft)/Lx + 0.5*f*Lx ) * x1 - 0.5*f*
    x1**2
80 y_ex = uex1(x_plot_ex)
81
82 #----- PLOTAGEM -----
83 import matplotlib.pyplot as plt
84 from google.colab import files
85
86 x_plot = np.linspace(0,Lx, n+1) #vetor x para plotagem
87 uh_value = uh.x.array.real #valores de uh nos graus de liberdade.
88
89 plt.figure(1,dpi=300)
90 plt.axes(facecolor = "lightgray") #cor do fundo da imagem
91 plt.plot (x_plot, uh_value, label = "Solução Aproximada") #plot solução
    aproximada
92 plt.plot(x_plot_ex, y_ex, 'k—', label = "Solução Exata") #plot solução
    exata
93 plt.legend(loc="best") #inserindo e posicionando a legenda.
94 plt.grid() #inserindo o grid
95 filename = 'FenicsX_solution_n%d.png'%n
96 plt.savefig ( filename ) #definindo nome do arquivo
97 print ( ' Graphics saved as "%s"' % ( filename ) )
98 files.download(filename) #fazer download do arquivo
99 plt.show()
100 plt.close()
101
102 #----- SOLUÇÃO ANALÍTICA -----
103 x = SpatialCoordinate(msh) #Define coordenadas espaciais.
104 uex = lambda x: uleft + ( (uright - uleft)/Lx + 0.5*f*Lx ) * x[0] - 0.5*f*x
    [0]**2
105 eh = uh - uex(x)
106
107 #----- CALCULANDO NORMAS DO ERRO -----
108 #norma L2

```

```

109 eL2form = fem.form(eh**2*dx)
110 L2norm = np.sqrt(fem.assemble_scalar(eL2form))
111
112 #norma H1
113 eH1form = fem.form( inner(grad(eh), grad(eh))*dx)
114 H1norm = np.sqrt( fem.assemble_scalar(eL2form) + fem.assemble_scalar(
    eH1form) )
115
116 h = Lx/n
117 return uh, eh, L2norm, H1norm, h
118
119 uleft = 0      #Condição de contorno da esquerda
120 uright = 0     #Condição de contorno da direita
121 f_value = 2.0  #valor de f
122 Lx = 1         #comprimento final do domínio (0,Lx)
123 mu_value = 1.0 #valor da constante mu
124
125 degree = 1 #grau da função de aproximação.
126 n=3 #número de elementos
127 uh, eh, L2norm, H1norm, h = solucao_MEF_1D(n, uleft, uright, degree, Lx,
    mu_value, f_value)

```

A solução encontrada é mostrada pela Figura 3.3.

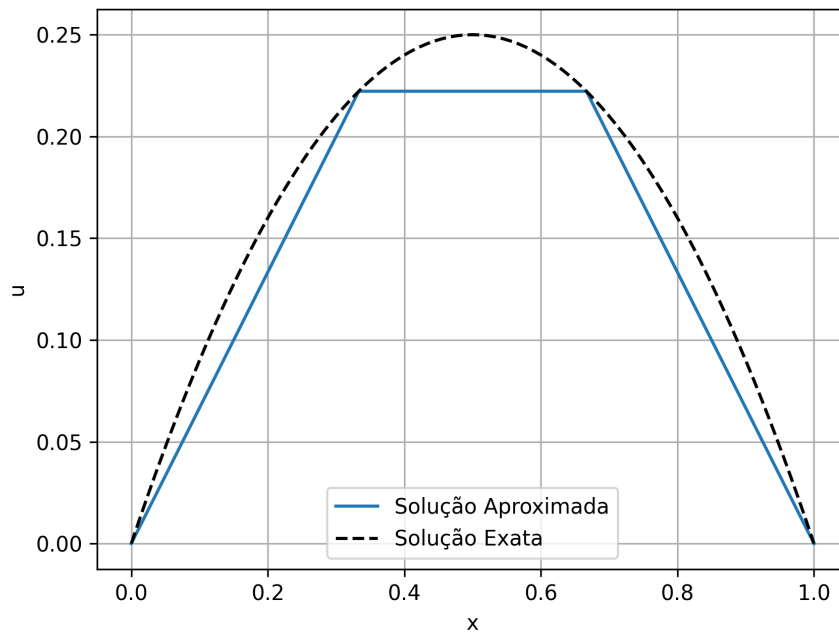


Figura 3.3: Distribuição da velocidade utilizando as bibliotecas FEniCSx.

3.1.1.4 Comparação entre as bibliotecas

Observa-se diferenças marcantes entre as abordagens empregadas nas três ferramentas apresentadas. Ao utilizar a biblioteca NumPy & SymPy, é necessário implementar manualmente as funções de base ϕ , construir a matriz $\mathbf{A}$ e $\mathbf{F}$ para considerar as condições de contorno, conforme evidenciado no código da seção 3.1.1.1 (linhas 33 a 53). Este processo, embora educativo, torna-se trabalhoso e suscetível a erros quando comparado às abordagens automatizadas das outras duas ferramentas.

Por outro lado, as bibliotecas Firedrake e FEniCSx oferecem uma abstração mais avançada. Nestas ferramentas, a definição das formas linear e bilinear é realizada de maneira mais concisa e elegante. Além disso, a aplicação das condições de contorno é feita separadamente, e o ajuste automático da matriz $\mathbf{A}$ e do vetor $\mathbf{F}$ é tratado internamente pelas bibliotecas, reduzindo significativamente a complexidade do código e permitindo ao usuário focar na formulação do problema.

Embora Firedrake e FEniCSx apresentem características semelhantes, a escolha pela utilização desta última neste trabalho se justifica por diversos fatores. Primeiramente, o FEniCSx é projetado com foco em computações massivamente paralelas, adotando uma abordagem rigorosa na qual apenas contribuições alinhadas a esse objetivo de design são aceitas [13]. Este aspecto é essencial para a resolução de problemas mais complexos e de grande escala, como aqueles que envolvem domínios tridimensionais ou simulações com alta demanda computacional. Apesar de ser uma ferramenta relativamente nova, possui uma comunidade ativa de desenvolvedores, o que a torna uma escolha promissora.

Por fim, a decisão de utilizá-la reflete a intenção de explorar e apresentar uma solução moderna e robusta para o Método dos Elementos Finitos, com o objetivo de fornecer aos leitores uma introdução prática e didática às suas capacidades. Assim, todos os problemas apresentados à seguir serão resolvidos exclusivamente utilizando o FEniCSx.

3.1.2 Problemas 2D

A transição do problema unidimensional para o bidimensional permite explorar características mais complexas da formulação do método de elementos finitos e da implementação do código computacional.

3.1.2.1 Condição de Fronteira do tipo Dirichlet

Este problema bidimensional é modelado pela equação de Poisson com domínio $\Omega = [0, 1] \times [0, 1]$ e condições de Dirichlet sobre a fronteira, descrito como:

$$\begin{cases} -\nabla \cdot [\mu(\mathbf{x}) \nabla u(\mathbf{x})] &= f(\mathbf{x}), & \mathbf{x} \in \Omega, \\ u(\mathbf{x}) &= u_D(\mathbf{x}), & \mathbf{x} \in \Gamma_D, \end{cases} \quad (3.2)$$

com:

$$\begin{aligned} u_D(\mathbf{x}) &= \text{sen}(2\pi x)\text{sen}(2\pi y), & \mathbf{x} \in \Gamma_D, \\ \mu(\mathbf{x}) &= 1 + \cos(2\pi x)\cos(2\pi y), & \mathbf{x} \in \Omega. \end{aligned} \quad (3.3)$$

Aqui $u_D(\mathbf{x})$ representa a condição de Dirichlet não homogênea, $\mu(\mathbf{x})$ é o coeficiente de viscosidade, e a solução manufaturada do problema foi extraída das notas de aulas em [18] e é apresentada da forma:

$$u(\mathbf{x}) = \text{sen}(2\pi x)\text{sen}(2\pi y), \quad \mathbf{x} \in \Omega.$$

O domínio é discretizado em n elementos triangulares de tamanhos iguais. O espaço de aproximação escolhido é formado por funções polinomiais de grau 1 por partes em cada elemento e contínuas entre os elementos:

$$V_h = \{v, v|_{(0,h)} \in P_1(0, h), v|_{(h,2h)} \in P_1(h, 2h), \dots, v|_{((n-1)h,1)} \in P_1((n-1)h, 1)\} \cap C^0(\Omega).$$

Destaques da Implementação

A transição do caso 1D para 2D requer algumas modificações significativas no código, principalmente na definição da malha e na imposição da condição de contorno.

A construção da malha para o domínio retangular, utilizando elementos triangulares é criada com o seguinte código:

```
1 msh = mesh.create_rectangle(comm=mpi4py.MPI.COMM_WORLD, points=((0.0,0.0),
    (Lx, Ly)), n=(N_x,N_y), cell_type = mesh.CellType.triangle)
```

onde $(0.0, 0.0)$ e (Lx, Ly) são, respectivamente, as coordenadas dos limites inferior esquerdo e superior direito do domínio, e (N_x, N_y) indica o número de células nas direções x e y (neste caso particular $N_x = N_y = N$).

A imposição das condições de contorno no caso 2D é feita de forma semelhante ao problema 1D, mas agora é necessário definir também as condições nas fronteiras $y = 0$ e $y = Ly$. No código, $x[0]$ corresponde à coordenada x e $x[1]$ à coordenada y . A função `fem.locate_dofs_geometrical` é utilizada para localizar os graus de liberdade nas bordas do domínio:

```
1 dofsL = fem.locate_dofs_geometrical(V, lambda x: np.isclose(x[0], 0))
2 dofsR = fem.locate_dofs_geometrical(V, lambda x: np.isclose(x[0], Lx))
3 dofsD = fem.locate_dofs_geometrical(V, lambda x: np.isclose(x[1], 0))
4 dofsU = fem.locate_dofs_geometrical(V, lambda x: np.isclose(x[1], Ly))
```

O diâmetro de cada elemento da malha foi armazenado em uma lista h_e e o maior diâmetro entre todos os elementos em h_{dim} :

```
1 #----- Calculo do valor de h (diâmetro do elemento) -----
2 tdim = msh.topology.dim #dimensão topológica
3 num_cells = msh.topology.index_map(tdim).size_local + msh.topology.
    index_map(tdim).num_ghosts #número de elementos
4 msh.topology.create_connectivity(tdim-1, tdim)
5 h_e = msh.h(tdim-1, np.arange(num_cells))
6 h_dim = max(h_e)
```

O código completo deste problema pode ser consultado no Apêndice A.1.

Resultados e análises

A Figura 3.4 ilustra a solução aproximada para $n = 512$ elementos.

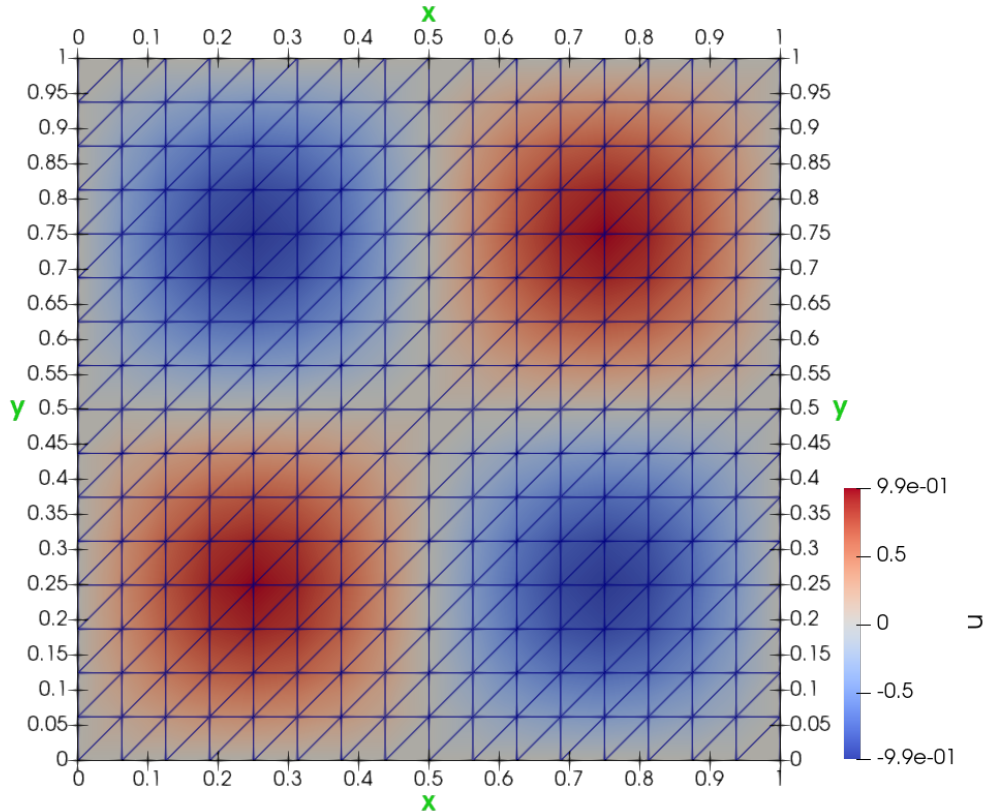


Figura 3.4: Distribuição da velocidade ao longo de Ω .

Na Figura 3.4 é possível visualizar os elementos triangulares que compõem a malha. O valor da solução u está representado através de cores, conforme legenda à direita da figura.

Para avaliar a qualidade numérica, comparamos a solução aproximada u_h com a solução analítica u através das normas $L^2(\Omega)$ e $H^1(\Omega)$ vistas em (2.1) e (2.3):

$$\|e\|_{L^2(\Omega)} = \|u - u_h\|_{L^2(\Omega)}, \quad \|e\|_{H^1(\Omega)} = \|u - u_h\|_{H^1(\Omega)}.$$

Primeiramente, optou-se por fixar a malha com $h = 0,1 \cdot \sqrt{2}m$ e avaliar a influência do grau do polinômio na aproximação por meio do erro absoluto em cada norma. Estes resultados, para aproximações polinomiais de grau 1, 2 e 3, são apresentados na Tabela 3.1.

Tabela 3.1: Relação da norma do erro absoluto com o aumento do grau polinomial.

Grau polinomial	$\ e\ _{H^1(\Omega)}$	$\ e\ _{L^2(\Omega)}$
grau 1	1,37e+00	5,23e-02
grau 2	1,72e-01	2,41e-03
grau 3	1,40e-02	1,39e-04

É possível observar que, mesmo para uma malha relativamente grossa, as normas do erro $\|e\|_{H^1(\Omega)}$ e $\|e\|_{L^2(\Omega)}$ decrescem significativamente à medida que se aumenta o grau do polinômio de aproximação.

A seguir, é realizada a análise da ordem de convergência, refinando a malha de maneira que $h_i = \frac{h_{i-1}}{2}$, para $i \in \{1, 2, \dots, 5\}$. A ordem de convergência (q) é calculada utilizando a fórmula:

$$q_V \approx \frac{\log\left(\frac{\|e_1\|_V}{\|e_2\|_V}\right)}{\log\left(\frac{h_1}{h_2}\right)}. \quad (3.4)$$

Realizamos esta análise para os espaços de aproximação P_1 , P_2 e P_3 .

Convergência para P_1 : Os resultados para aproximações com polinômios de grau 1 são apresentados na Tabela 3.2 (com valores de q aproximados com duas casas decimais).

Tabela 3.2: Norma do erro para P_1 .

i	h_i	$\ e\ _{H^1(\Omega)}$	$\ e\ _{L^2(\Omega)}$	q_{H^1}	q_{L^2}
0	$0,1\sqrt{2}$	1,37e+00	5,23e-02	-	-
1	$0,05\sqrt{2}$	6,95e-01	1,36e-02	0,98	1,95
2	$0,025\sqrt{2}$	3,49e-01	3,42e-03	1,00	1,99
3	$0,0125\sqrt{2}$	1,74e-01	8,56e-04	1,00	2,00
4	$0,00625\sqrt{2}$	8,72e-02	2,14e-04	1,00	2,00
5	$0,003125\sqrt{2}$	4,36e-02	5,40e-05	1,00	2,00

Os valores obtidos mostram que as ordens de convergência teórica para P_1 , $q_{H^1} = 1$ e $q_{L^2} = 2$, são confirmadas pelos experimentos numéricos. Estas ordens foram vistas através das estimativas (2.37) e (2.38).

Outra forma de analisar a convergência é gráfica utilizando linhas de referência que representam as ordens teóricas $O(h)$ e $O(h^2)$. Para isso, usamos a escala log-log e traçamos um gráfico onde o eixo horizontal corresponde ao espaçamento h , e o eixo vertical fornece a

norma do erro absoluto avaliadas em $H^1(\Omega)$ e $L^2(\Omega)$. A Figura 3.5 apresenta os resultados dessa análise.

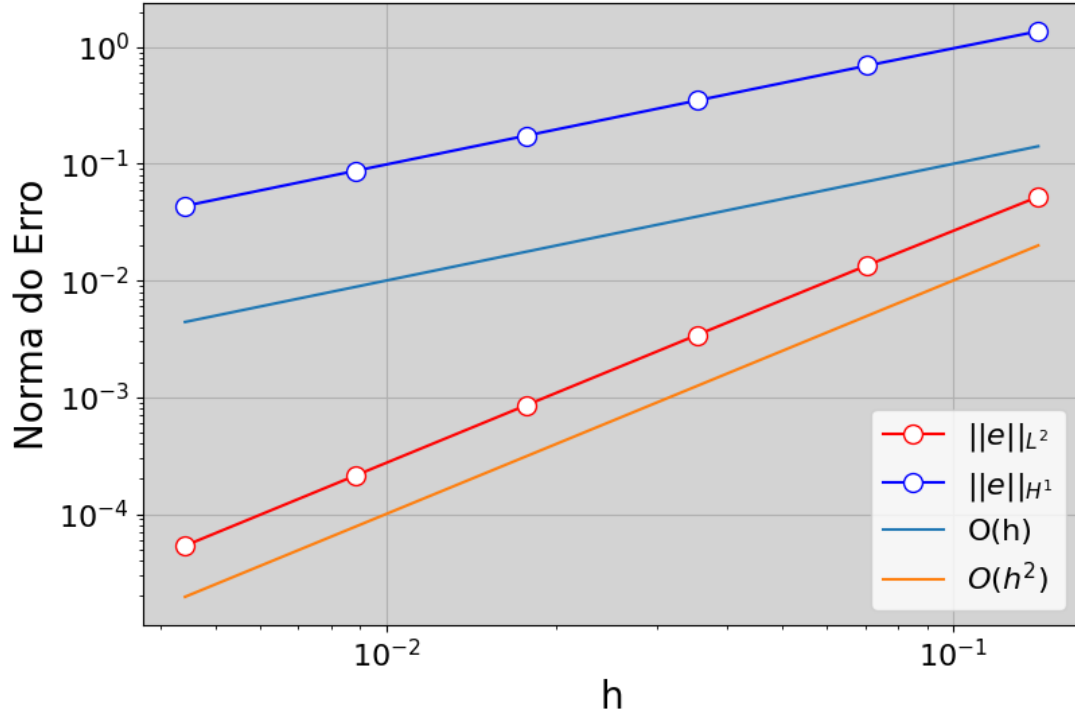


Figura 3.5: Ordem de convergência com aproximação polinomial de grau 1.

Perceba que, na escala log-log, o erro na norma $L^2(\Omega)$ decai com inclinação bem semelhante à linha de referência $O(h^2)$ e usando a norma $H^1(\Omega)$, o decaimento é da ordem $O(h)$, como era esperado pelas estimativas teóricas (2.37) e (2.38).

Convergência para P_2 : Os resultados para o espaço de aproximação P_2 , são apresentados na Tabela 3.3.

Tabela 3.3: Norma do erro para P_2 .

i	h_i	$\ e\ _{H^1(\Omega)}$	$\ e\ _{L^2(\Omega)}$	q_{H^1}	q_{L^2}
0	0,141421	1,72e-01	2,41e-03	-	-
1	0,070711	4,32e-02	2,87e-04	1,99	3,07
2	0,035355	1,08e-02	3,54e-05	2,00	3,02
3	0,017678	2,70e-03	4,41e-06	2,00	3,00
4	0,008839	6,76e-04	5,51e-07	2,00	3,00
5	0,004419	1,69e-04	6,88e-08	2,00	3,00

O gráfico de convergência na Figura 3.6 reforça o comportamento da ordem de convergência, comparando a inclinação das normas do erro com as linhas de referências $O(h^2)$ e $O(h^3)$.

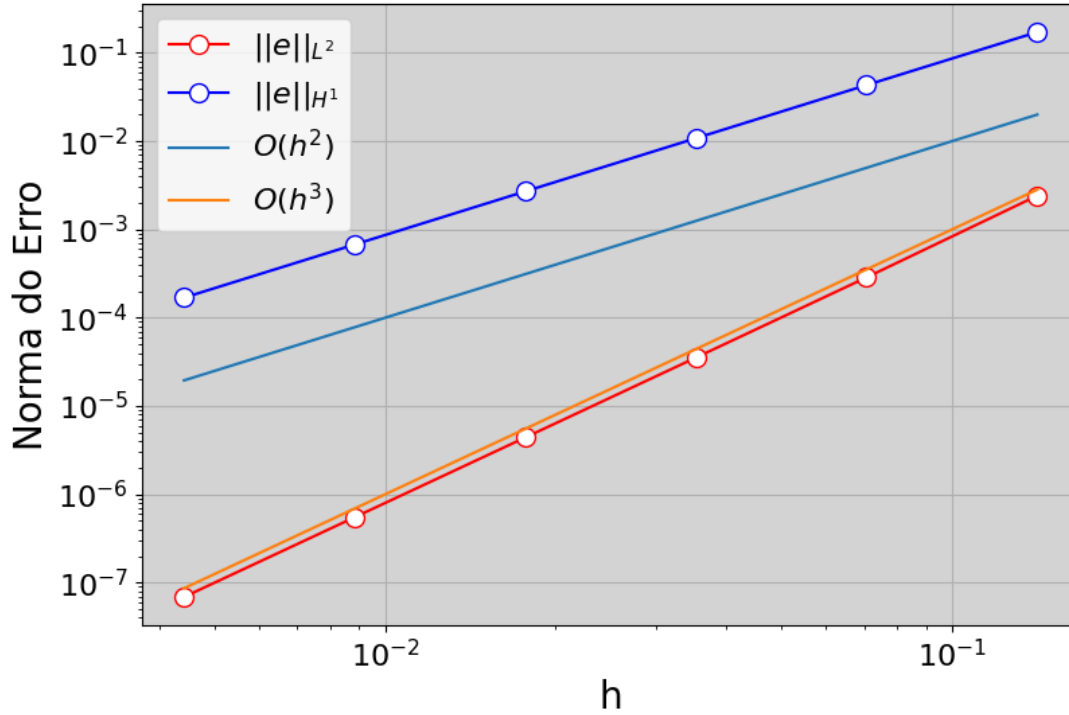


Figura 3.6: Convergência com aproximação polinomial de grau 2.

De acordo com os resultados apresentados na Figura 3.6 e Tabela 3.3 vê-se que ao adotar aproximação polinomial de grau 2, a ordem de convergência aumenta $q_{H^1} \approx 2$ e $q_{L^2} \approx 3$, o que condiz com a teoria.

Convergência para P_3 :

Procedendo de maneira análoga, utilizou-se também uma aproximação por polinômio de grau 3, e os resultados são mostrados na Tabela 3.4.

Tabela 3.4: Norma do erro para P_3 .

i	h_i	$\ e\ _{H^1(\Omega)}$	$\ e\ _{L^2(\Omega)}$	q_{H^1}	q_{L^2}
0	0,141421	1,40e-02	1,39e-04	-	-
1	0,070711	1,71e-03	8,17e-06	3,03	4,09
2	0,035355	2,11e-04	4,95e-07	3,02	4,04
3	0,017678	2,63e-05	3,06e-08	3,01	4,02
4	0,008839	3,28e-06	1,90e-09	3,00	4,01
5	0,004419	4,10e-07	1,19e-10	3,00	4,00

Os dados da Tabela 3.4 estão dispostos na Figura 3.7, na forma gráfica do sistema Cartesiano com escala log-log.

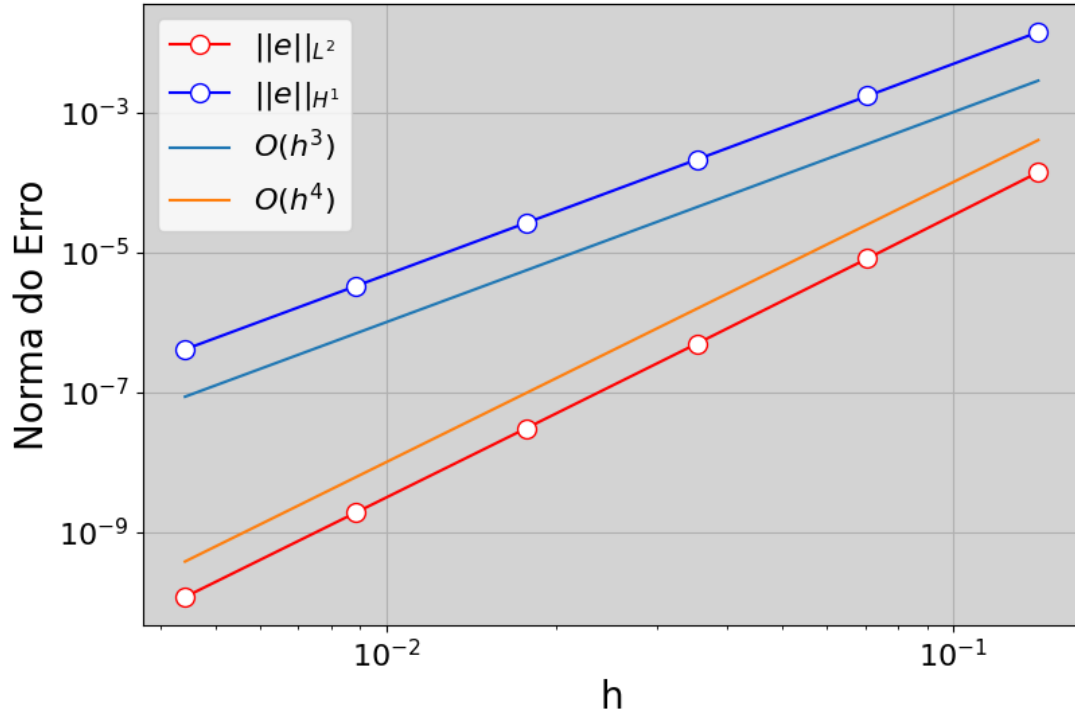


Figura 3.7: Convergência com aproximação polinomial de grau 3.

Novamente a convergência teórica é confirmada pelos experimentos, ou seja, utilizando polinômios de grau 3 para aproximar, os resultados apresentaram $q_{H^1} \approx 3$ e $q_{L^2} \approx 4$.

Uma observação importante sobre a solução analítica deste problema é que, caso ela fosse uma função polinomial de grau 2 por exemplo, a utilização de aproximações polinomiais de grau maior ou igual a dois resultaria em uma solução numérica idêntica à solução exata (desconsiderando os erros de arredondamento numérico). Essa coincidência poderia dificultar a análise de convergência do método para esses graus polinomiais. Por este motivo, a escolha de uma solução manufaturada baseada em uma função seno foi fundamental para garantir uma análise apropriada da convergência.

Os resultados apresentados nas tabelas indicam que, para reduzir a norma do erro, é possível refinar a malha, enquanto o aumento do grau polinomial utilizado na aproximação pode elevar a ordem de convergência, desde que a solução u seja regular, conforme visto no Capítulo 2.

3.1.2.2 Condição de Fronteira do tipo Dirichlet e Neumann

Quando o problema envolve as condições de fronteira de Dirichlet e Neumann simultaneamente, são necessários ajustes específicos para a sua resolução. Neste caso consideramos um problema de Poisson bidimensional com essas condições de fronteira, onde o objetivo é determinar u , tal que:

$$\begin{cases} -\nabla \cdot [\mu(\mathbf{x}) \nabla u(\mathbf{x})] = f(\mathbf{x}), & \mathbf{x} \in \Omega = [0, 1] \times [0, 1], \\ u(\mathbf{x}) = g_D(\mathbf{x}), & \mathbf{x} \in \Gamma_D = \{(0, y), (1, y) \mid y \in [0, 1]\}, \\ -\mu(\mathbf{x}) \nabla u(\mathbf{x}) \cdot \tilde{\mathbf{n}} = g_N(\mathbf{x}), & \mathbf{x} \in \Gamma_N = \{(x, 0), (x, 1) \mid x \in [0, 1]\}, \end{cases} \quad (3.5)$$

onde Γ_D é a fronteira onde se aplica a condição de Dirichlet e Γ_N corresponde à fronteira com a condição de Neumann.

A formulação variacional para este problema é construída conforme discutido no Capítulo 2.

A solução analítica do problema é dada por:

$$u(\mathbf{x}) = u_l + ((u_r - u_l)/Lx + 0.5 \cdot f \cdot Lx) \cdot x - 0.5 \cdot f \cdot x^2,$$

em que $\mathbf{x} = (x, y)$; $u_l = u(0, y)$ e $u_r = u(1, y)$, $\forall y \in [0, 1]$ são os valores das condições de Dirichlet; $g_N(x, y) = 0$, $\forall (x, y) \in \Gamma_N$ é a condição de Neumann; $Lx = 1m$ é o tamanho do domínio na direção x ; e $f(x, y) = f = 1N/m^3$.

Destaques da Implementação

O valor da condição de Neumann é armazenado na variável g_N , e essa condição é incorporada na construção da forma linear L :

```
1 g_N = fem.Constant(msh, 0.0)
2 L = (source * v) * dx - (g_N*v) * dx
```

O código completo para solucionar este problema encontra-se no Apêndice A.2.

Resultados e análises

Este problema está sendo abordado com foco na análise da norma do erro e da convergência do método.

Para iniciar esta análise, utilizou-se aproximação por polinômio de grau 1 (P_1). A Figura 3.8 apresenta os resultados dessa aproximação, evidenciando que o método converge para a solução analítica à medida que a malha é refinada. O processo de refinamento foi iniciado com $h_0 = 0,1\sqrt{2}m$, reduzindo o tamanho do elemento pela metade a cada etapa de refinamento. O menor erro calculado através da norma L^2 foi na ordem de $10e - 6$ com $h_5 = 0,003125\sqrt{2}m$.

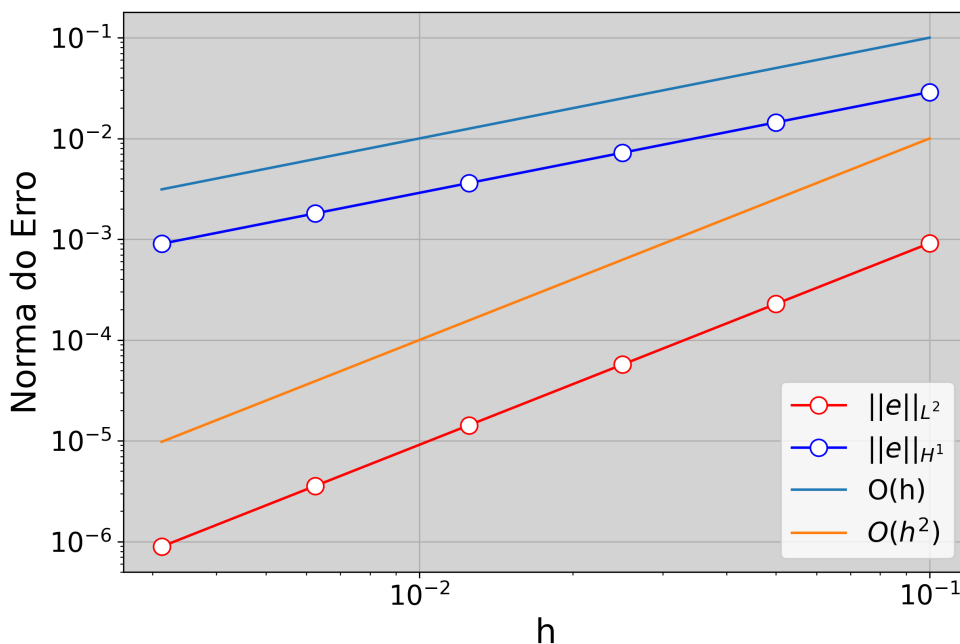


Figura 3.8: Análise de convergência da solução (P_1).

Ao utilizar aproximação por polinômio de grau 2 (P_2) e com $h = 0,5\sqrt{2}$, as normas do erro calculadas foram: $\|e\|_{H^1} = 2,47e - 13$ e $\|e\|_{L^2} = 6,62e - 14$. Esses valores indicam que o método já convergiu para a solução analítica, considerando que o *eps* da máquina é $2,22e - 16$ e que erros de arredondamento estejam presentes. Esse resultado é particularmente notável, pois foi obtido mesmo com um valor relativamente grande de h , ou seja, com uma malha composta por poucos elementos.

Esse comportamento é explicado pelo fato de que a solução analítica u , que é um polinômio de grau 2, pertence ao espaço de busca V_h . Assim, a melhor solução encontrada pelo método é a própria solução analítica u , resultando em erros extremamente pequenos que se aproximam do limite imposto pelo arredondamento numérico.

3.1.2.3 Malha não estruturada

O uso de malhas não estruturadas é particularmente vantajoso para problemas definidos em domínios irregulares, pois permite uma melhor adaptação da discretização à geometria do problema, resultando em aproximações mais precisas.

Neste tópico, consideramos o mesmo problema apresentado na Seção 3.1.2.1, mas agora utilizando uma malha não estruturada gerada com a biblioteca *gmsh*. Para isso, basta substituir a geração de malha no código original pelo trecho abaixo:

```

1 def GenerateMesh():
2     gmsh.initialize()
3     proc = MPI.COMM_WORLD.rank
4
5     if proc == 0:
6         center_x, center_y = 0, 0
7         background = gmsh.model.occ.addDisk(center_x, center_y, 0, radius,
8         radius)
9         gmsh.model.occ.synchronize()
10        pl = gmsh.model.addPhysicalGroup(2, [background], 1)
11
12        gmsh.model.mesh.setSize(gmsh.model.getEntities(0), h_malha)
13
14        if (cell_type == mesh.CellType.quadrilateral):
15            gmsh.model.mesh.setRecombine(2, pl)
16
17        gmsh.model.mesh.generate(2)
18
19        msh, _, _ = io.gmshio.model_to_mesh(gmsh.model, comm=MPI.COMM_WORLD, rank
20        =0, gdim=2)
21        return msh

```

Além de visualizar a solução diretamente no Google Colab, também é possível salvá-la nos formatos *xdmf* e *h5*, permitindo sua análise em softwares externos. O trecho do código à seguir ilustra esse procedimento:

```

1 #----- SALVANDO ARQUIVO -----
2 uh.name = "u" #nomeia a variável da solução uh
3 with io.XDMFFile(mpi4py.MPI.COMM_WORLD, "temp_malha_3_2_1_3.xdmf", "w") as
4     file:
5         file.write_mesh(msh) #armazena os dados da malha
6         file.write_function(uh) #armazena os dados da solução
7 files.download('temp_malha_3_2_1_3.xdmf') #efetua o download do arquivo em
8 xdmf
9 files.download('temp_malha_3_2_1_3.h5') #efetua o download do arquivo em x5

```

O código completo está disponível no Apêndice A.3.

A Figura 3.9 apresenta a solução para este problema (3.2)-(3.3) no espaço de aproximação P_1 , utilizando elementos triangulares com $h \approx 0.08186$, onde também se pode observar a estrutura da malha gerada:

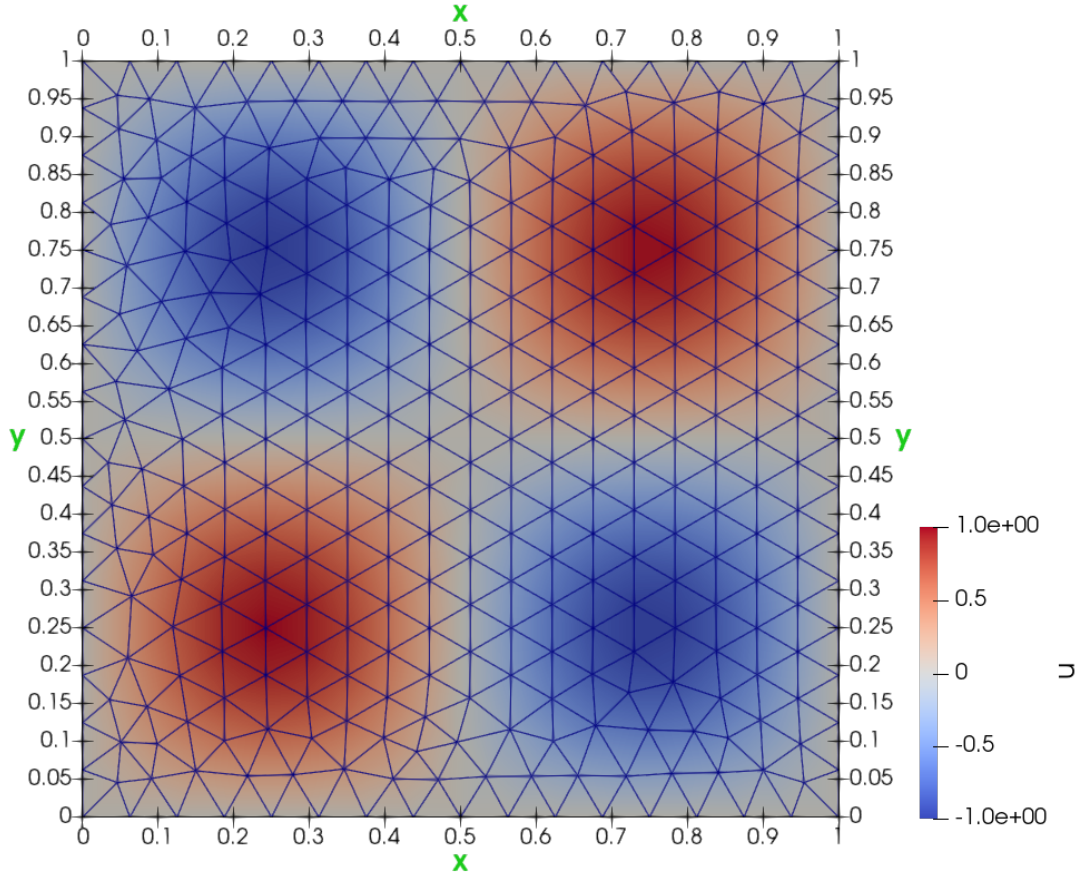


Figura 3.9: Distribuição da velocidade no espaço P_1 ao longo de Ω , com malha não estruturada.

A seguir, consideramos problemas com domínios irregulares.

3.1.2.4 Domínio irregular

3.1.2.4.1 Primeiro problema Este primeiro exemplo utiliza um domínio irregular representado por um círculo unitário (raio $r = 1m$), centralizado na origem $(0,0)$. Para resolver o problema, empregamos aproximação por polinômios de grau 1 e elementos triangulares em uma malha não estruturada. A formulação do problema segue:

$$\begin{cases} -\nabla \cdot [\mu(\mathbf{x}) \nabla u(\mathbf{x})] = f(\mathbf{x}) & \mathbf{x} \in \Omega, \\ u(\mathbf{x}) = g_D(\mathbf{x}) & \mathbf{x} \in \Gamma_D, \end{cases} \quad (3.6)$$

onde $\Omega = \{(x, y) \in \mathbb{R}^2 : x^2 + y^2 < 1\}$, $\Gamma_D = \{(x, y) \in \mathbb{R}^2 : x^2 + y^2 = 1\}$. Os dados do problema são:

$$\begin{aligned} \mu(\mathbf{x}) &= 1 \text{ Pa} \cdot s, & \mathbf{x} \in \Omega, \\ f(\mathbf{x}) &= -\nabla \cdot [\mu(\mathbf{x}) \nabla u(\mathbf{x})] \text{ N/m}^3, & \mathbf{x} \in \Omega, \\ u(\mathbf{x}) &= 0 \text{ m/s}, & \mathbf{x} \in \Gamma_D. \end{aligned} \quad (3.7)$$

A solução analítica para este problema é conhecida e dada por:

$$u(\mathbf{x}) = -\text{sen}(x^2) + \text{sen}(1 - y^2), \quad \mathbf{x} \in \Omega. \quad (3.8)$$

Destaques da Implementação

Para este exemplo, o domínio circular foi gerado utilizando o comando `gmsh.model.occ.addDisk(c_x, c_y, c_z, r_x, r_y)`, que permite definir c_x, c_y e c_z como sendo as coordenadas do centro do círculo (x, y, z) , e r_x e r_y como os raios, respectivamente, nas direções x e y . Essa abordagem possibilita também a criação de elipses ao modificar os valores dos raios. O trecho do código para implementar esta geometria é mostrado à seguir:

```
1 c_x, c_y = 0, 0 #coordenadas do centro do círculo
2 background = gmsh.model.occ.addDisk(c_x, c_y, 0, r_x, r_y)
3 gmsh.model.occ.synchronize()
4 pl = gmsh.model.addPhysicalGroup(2, [background], 1)
5 gmsh.model.mesh.setSize(gmsh.model.getEntities(0), h_malha)
```

A construção de uma função *boundary* permitiu a identificação dos graus de liberdade na fronteira do domínio:

```
1 #Identificação dos graus de liberdade para as condições de contorno
2 def boundary(x):
3     r = np.sqrt(x[0]**2 + x[1]**2)
4     return np.isclose(r, radius)
```

O código completo encontra-se no Apêndice A.4.

Resultados e análises

A Figura 3.10 apresenta a solução aproximada do problema (3.6)-(3.7) no espaço P_1 , gerada em uma malha não estruturada com $h \approx 0,29474 \text{ m}$. Observe que este tipo de malha permite melhor adaptação à borda do domínio, embora as arestas dos triângulos sejam secantes à circunferência, o que é uma característica inerente ao uso de elementos com arestas lineares.

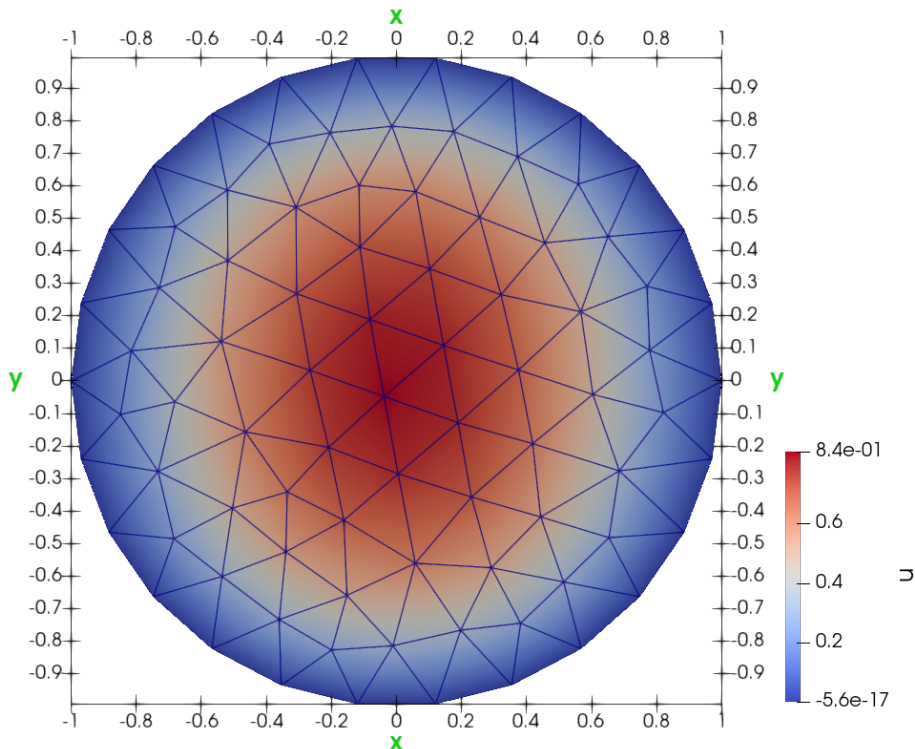


Figura 3.10: Distribuição da velocidade no espaço P_1 , com malha não estruturada.

Para verificar a acurácia do método em geometrias irregulares com malha não estruturada, realizamos uma análise de convergência. O refinamento da malha foi controlado pelo parâmetro h , correspondente ao maior diâmetro dos elementos triangulares. É importante destacar que devido à natureza não estruturada da malha, h não segue uma relação uniforme ao ser refinado. Além disso, os triângulos que compõem a malha não conseguem preencher perfeitamente a borda circular do domínio, pois suas arestas são secantes à circunferência. Quando realizamos o refinamento, uma nova malha é gerada do zero, resultando em elementos que se ajustam melhor ao formato do domínio. Esse processo não apenas diminui o tamanho dos elementos, como também contribui para uma *convergência geométrica* (o contorno do domínio é representado de maneira mais fiel a cada passo de refinamento), o que impacta positivamente na precisão da solução aproximada.

O erro foi avaliado nas normas H^1 e L^2 e seus valores apresentados na Tabela 3.5 mostram valores relativamente baixos, mesmo para as malhas mais grossas e reduzem ainda mais para as malhas mais finas. A ordem de convergência (q_{L^2}) foi calculada com a fórmula (3.4). Todos estes resultados estão apresentados na Tabela 3.5.

Tabela 3.5: Convergência do problema (3.6).

h	$\ e\ _{H^1}$	$\ e\ _{L^2}$	q_{L^2}
0,235690	1,83e-01	1,50e-02	-
0,129531	9,54e-02	3,97e-03	2,22
0,067823	4,84e-02	1,00e-03	2,13
0,031922	2,42e-02	2,51e-04	1,84
0,016612	1,21e-02	6,31e-05	2,12
0,008332	6,07e-03	1,58e-05	2,01
0,004349	3,04e-03	3,95e-06	2,13

Para uma visualização gráfica da Tabela 3.5, os dados foram dispostos em sistema gráfico, onde pode-se realizar a interpretação do comportamento geométrico destes dados.

A Figura 3.11 apresenta os resultados dessa análise. É possível observar que a inclinação da curva do erro absoluto através da norma L^2 se aproxima da linha de referência $O(h^2)$, o que confirma que a ordem de convergência $q_{L^2} \approx 2$, como já identificado na Tabela 3.5. De maneira análoga, o erro calculado pela norma H^1 apresenta inclinação compatível com a ordem $O(h)$, também esperado teoricamente.

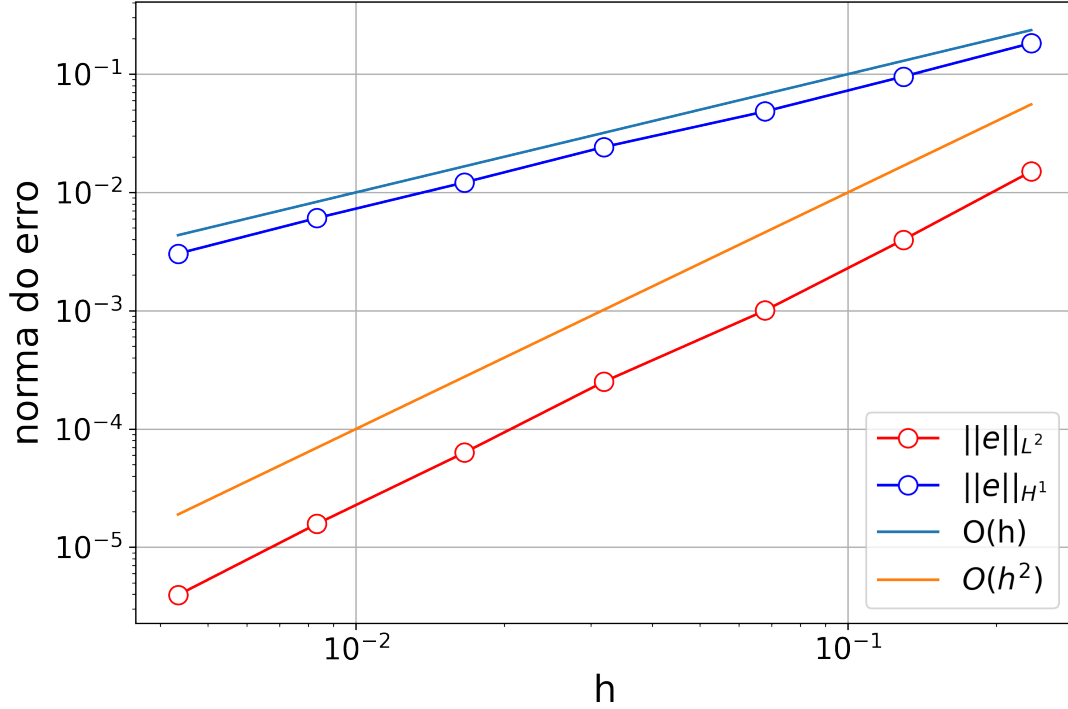


Figura 3.11: Decaimento da norma do erro absoluto em função de h , na escala log-log.

Os resultados apresentados na Figura 3.11 corroboram os valores obtidos na Tabela 3.5, mas sob uma perspectiva visual que facilita a interpretação do comportamento do erro. Essa abordagem permite observar mais claramente como a inclinação das curvas segue as ordens teóricas, mesmo em malhas não estruturadas. Portanto, a precisão do método é confirmada por estas duas análises.

3.1.2.4.2 Segundo problema Neste exemplo, consideramos um domínio quadrado $[0, 1] \times [0, 1]$ contendo um buraco no centro. Este buraco é um círculo de raio $r = 0,25 \text{ m}$, centrado em $(c_x, c_y) = \left(\frac{1}{2}, \frac{1}{2}\right) \text{ m} \times \text{m}$. O problema a ser resolvido sobre este domínio, que possui uma fronteira interna devido ao obstáculo, é modelado por:

$$\begin{cases} -\nabla \cdot [\mu(\mathbf{x}) \nabla u(\mathbf{x})] = f(\mathbf{x}), & \mathbf{x} \in \Omega, \\ u(\mathbf{x}) = u_D(\mathbf{x}), & \mathbf{x} \in \Gamma_D, \end{cases} \quad (3.9)$$

em que,

$$\begin{aligned} u_D(\mathbf{x}) &= \text{sen}(2\pi x_1) \text{sen}(2\pi x_2), \\ \mu(\mathbf{x}) &= 1 + \cos(2\pi x_1) \cos(2\pi x_2), \\ f(\mathbf{x}) &= -\nabla \cdot [\mu(\mathbf{x}) \nabla u(\mathbf{x})], \end{aligned} \quad (3.10)$$

onde Γ_D é a união da fronteira do retângulo com a fronteira da circunferência.

A solução analítica do problema (3.9)-(3.10) é dada por:

$$u(\mathbf{x}) = \text{sen}(2\pi x_1) \text{sen}(2\pi x_2), \quad \mathbf{x} \in \Omega.$$

Destaques da Implementação

Inicialmente definem-se separadamente o retângulo *background* e o círculo *hole*. Em seguida, a malha é gerada aplicando o comando `gmsh.model.occ.cut`, que remove a região correspondente ao círculo do domínio retangular, criando assim um buraco na geometria:

```

1 background = gmsh.model.occ.addRectangle(0, 0, 0, Lx, Ly)
2
3 center_hole_x, center_hole_y = Lx/2, Ly/2 #Coordenadas do centro do buraco.
4 hole = gmsh.model.occ.addDisk(center_hole_x, center_hole_y, 0, radius_hole,
    radius_hole)# Criação do buraco
5 gmsh.model.occ.cut([(2, background)], [(2, hole)]) #Subtraindo o buraco do
    retângulo
6
7 gmsh.model.occ.synchronize()
8 pl = gmsh.model.addPhysicalGroup(2, [background], 1)
9 pl_hole = gmsh.model.addPhysicalGroup(2, [hole], 2)

```

O código completo pode ser consultado no Apêndice A.5.

Resultados e análises

A Figura 3.12 apresenta a solução aproximada no espaço P_1 , ilustrada por um gráfico de cores, que representa os valores da função solução sobre a malha não estruturada. A simulação foi realizada utilizando elementos triangulares lineares e com $h \approx 0.07981 \text{ m}$.

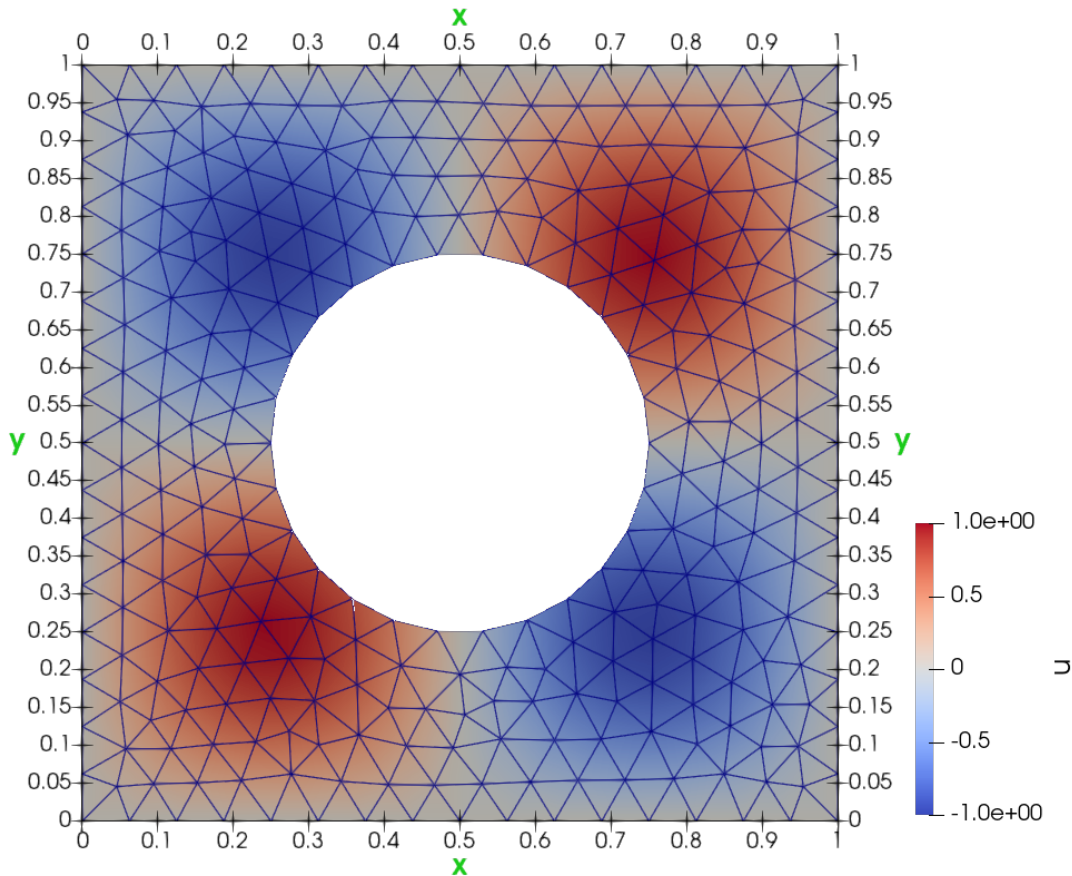


Figura 3.12: Distribuição da velocidade utilizando uma malha não estruturada.

Neste problema, as arestas dos elementos triangulares que se aproximam da fronteira circular são secantes ao contorno do buraco. Uma possível abordagem para reduzir esse erro seria o refinamento da malha (conforme realizado no exemplo anterior), o que, de fato, melhoraria a aproximação. No entanto, independentemente do nível de refinamento, a fronteira circular continuará sendo representada apenas de forma aproximada, além do próprio erro inerente ao método numérico.

Uma alternativa para melhorar a representação geométrica da fronteira seria o uso de elementos triangulares em que as arestas adjacentes ao buraco fossem curvadas, propor-

cionando uma aproximação mais fiel da fronteira circular. Entretanto, este tema não será abordado neste trabalho, mas os livros [23; 24] podem ser boas referências para cobrir este tópico em trabalhos futuros.

3.1.3 Problemas 3D

3.1.3.1 Paralelepípedo

Este problema modela o escoamento de um fluido em um domínio tridimensional no formato de um paralelepípedo com comprimento de 2 metros e seção transversal quadrada com área de 1 m^2 . As condições de contorno incluem fronteiras do tipo Dirichlet e Neumann, e o objetivo é determinar a velocidade u , que satisfaça:

$$\begin{cases} -\nabla \cdot [\mu(\mathbf{x}) \nabla u(\mathbf{x})] = f(\mathbf{x}), & \mathbf{x} \in \Omega, \\ u(\mathbf{x}) = u_D(\mathbf{x}), & \mathbf{x} \in \Gamma_D, \\ \frac{\partial u_D(\mathbf{x})}{\partial \eta} = 0, & \mathbf{x} \in \Gamma_N, \end{cases} \quad (3.11)$$

onde,

$$\begin{aligned} u_D(\mathbf{x}) &= \text{sen}(2\pi x) \cdot \text{sen}(2\pi y), & \mathbf{x} \in \Gamma_D, \\ \mu(\mathbf{x}) &= 1 + \cos(2\pi x) \cdot \cos(2\pi y), & \mathbf{x} \in \Omega. \end{aligned} \quad (3.12)$$

O domínio é definido como $\Omega = [0, 1] \times [0, 1] \times [0, 2]$, com $\mathbf{x} = (x, y, z)$. As condições de contorno do tipo Dirichlet foram impostas em $\Gamma_D = \{(0, y, z), (1, y, z), (x, 0, z), (x, 1, z) \mid x, y \in [0, 1] \text{ e } z \in [0, 2]\}$ e as de Neumann em $\Gamma_N = \{(x, y, 0), (x, y, 2) \mid x, y \in [0, 1]\}$. A solução analítica é dada por:

$$u(\mathbf{x}) = \text{sen}(2\pi x) \cdot \text{sen}(2\pi y), \mathbf{x} \in \Omega.$$

Destaques da Implementação

O problema tridimensional requer uma mudança no código. A construção da malha no formato de um paralelepípedo com elementos tetraédricos utiliza o comando `mesh.create_box`, de forma semelhante ao problema 2D:

```
1 msh = mesh.create_box(comm=mpi4py.MPI.COMM_WORLD, points=((0.0, 0.0, 0.0), (Lx, Ly, Lz)), n=(N, N, int((Lz/Lx)*N)), cell_type=mesh.CellType.tetrahedron)
```

O código completo pode ser encontrado no Apêndice A.6.

Resultados e análises

Para a geração da malha, foi utilizado o comando `mesh.create_box`, no qual são especificados os vértices extremos do paralelepípedo, a quantidade de elementos em cada direção e o tipo de elemento que, neste caso, são tetraedros.

A Figura 3.13 apresenta a solução aproximada do problema no espaço P_1 por meio de um gráfico de cores. As regiões de entrada e saída do fluido correspondem às fronteiras $\Gamma_{N1} = \{(x, y, 0) \in \mathbb{R}^3 \mid x, y \in [0, 1]\}$ e $\Gamma_{N2} = \{(x, y, 2) \in \mathbb{R}^3 \mid x, y \in [0, 1]\}$, respectivamente. A malha gerada é exibida, permitindo visualizar as faces triangulares dos elementos.

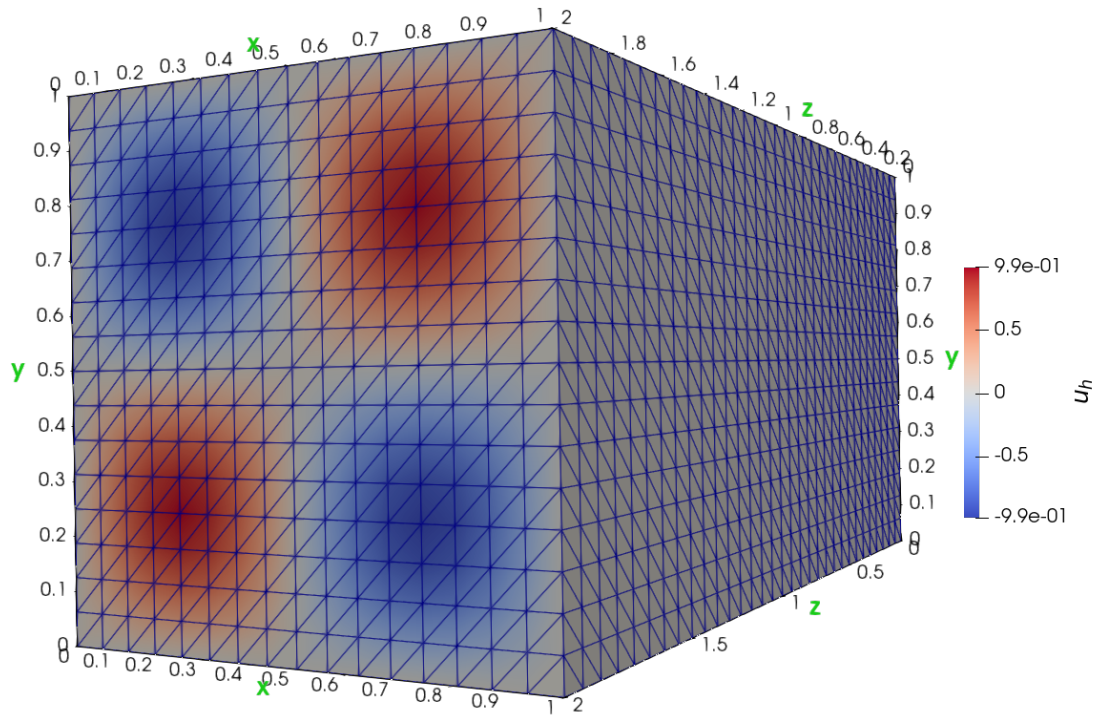


Figura 3.13: Distribuição da velocidade (solução aproximada) sobre a geometria de um paralelepípedo.

A solução analítica deste problema (3.11)-(3.12) é apresentada pela Figura 3.14.

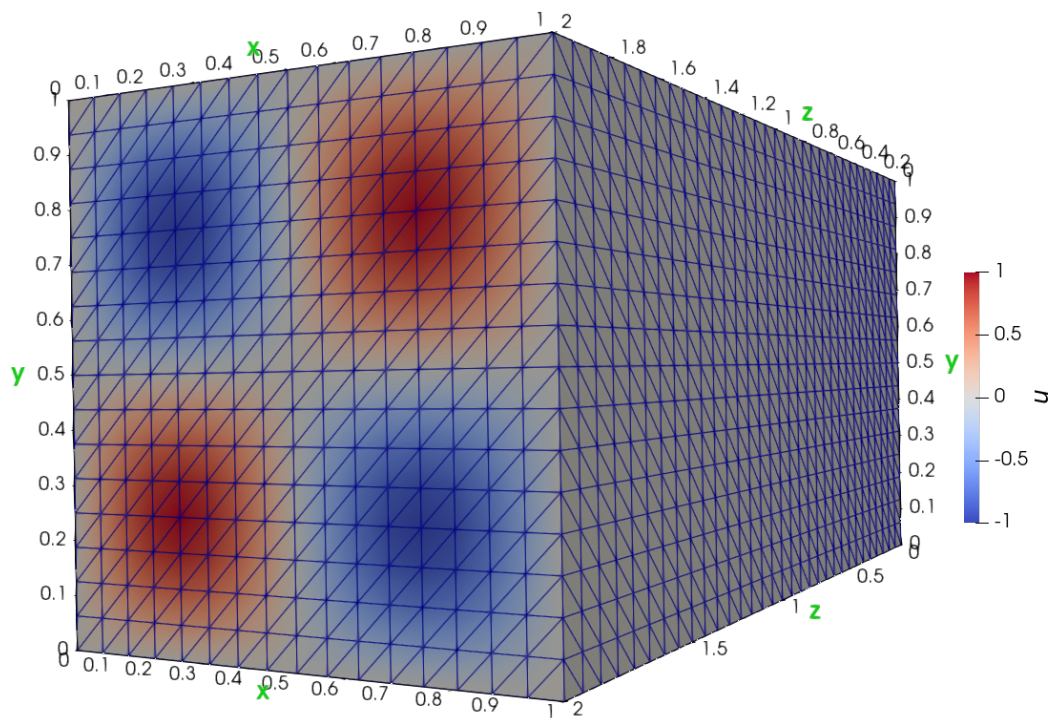


Figura 3.14: Distribuição da velocidade (solução analítica) sobre a geometria de um paralelepípedo.

Comparando as Figuras 3.13 e 3.14, observa-se que, qualitativamente, a solução aproximada obtida apresenta um comportamento muito semelhante à solução analítica. As soluções exibem padrões idênticos de variação representados pelo gradiente de cores.

A única diferença visual perceptível está na escala de valores indicada pela legenda das figuras. Enquanto a solução analítica apresenta valores máximos e mínimos exatamente iguais a 1 m/s e -1 m/s , respectivamente, a solução aproximada atinge valores ligeiramente inferiores, com um máximo de $0,99 \text{ m/s}$ e um mínimo de $-0,99 \text{ m/s}$.

Com base nessa análise qualitativa, observa-se que o método de elementos finitos resolve adequadamente problemas 3D, apresentando uma solução que reproduz bem as características da solução analítica.

Além da análise quantitativa, foi realizada uma análise quantitativa, na qual se calculou a norma do erro (L^2) para diferentes refinamentos de malha, avaliando o comportamento da convergência. Os valores da ordem de convergência foram arredondados para duas casas decimais, e os resultados estão apresentados na tabela 3.6.

Tabela 3.6: Análise de convergência do problema 3D com geometria de um paralelepípedo.

n	h	$\ e\ _{L^2}$	q_{L^2}
1500	0,346410	2,54e-01	-
12000	0,173205	7,41e-02	1,78
96000	0,086603	1,93e-02	1,94
768000	0,043301	4,86e-03	1,99

Nota-se que a norma do erro reduz à medida em que a malha é refinada, evidenciando a convergência do método. No primeiro refinamento, a ordem de convergência foi próxima de 1,78, um valor inferior do esperado. No entanto, à medida que a malha se torna mais fina, a ordem de convergência se aproxima de 2, o que está de acordo com a taxa teórica prevista para o método utilizado.

3.1.3.2 Cilindro

Este problema considera o escoamento de um fluido em um tubo cilíndrico de raio $r = 2$ metros e comprimento 4 metros. Adota-se a condição de contorno de Neumann na entrada e saída do tubo $\Gamma_N = \{(x, y, 0), (x, y, 4) \in \mathbb{R}^3 \mid x^2 + y^2 \leq 2^2\}$, onde a aceleração do fluido $\frac{\partial u(\mathbf{x})}{\partial \eta}$ é nula. Já na parede do tubo, impõe-se uma condição de contorno do tipo Dirichlet, com velocidade $u(\mathbf{x}) = 0$. A viscosidade do fluido é constante e igual a $1 \text{ Pa} \cdot \text{s}$. O problema, de domínio $\Omega = \{(x, y, z) \in \mathbb{R}^3 \mid x^2 + y^2 \leq 2^2; z \in [0, 4]\}$, é modelado pela seguinte equação:

$$\left\{ \begin{array}{ll} -\nabla \cdot [\mu(\mathbf{x}) \nabla u(\mathbf{x})] &= f(\mathbf{x}), \quad \mathbf{x} \in \Omega, \\ u(\mathbf{x}) &= u_D(\mathbf{x}), \quad \mathbf{x} \in \Gamma_D, \\ \frac{\partial u(\mathbf{x})}{\partial \eta} &= 0, \quad \mathbf{x} \in \Gamma_N, \end{array} \right. \quad (3.13)$$

com

$$\begin{aligned} u(\mathbf{x}) &= -\text{sen}(x^2) + \text{sen}(r^2 - y^2), \quad \mathbf{x} \in \Omega, \\ u_D(\mathbf{x}) &= -\text{sen}(x^2) + \text{sen}(r^2 - y^2) = 0, \quad \mathbf{x} \in \Gamma_D, \\ \mu(\mathbf{x}) &= 1, \quad \mathbf{x} \in \Omega, \end{aligned} \quad (3.14)$$

onde a condição de Dirichlet é imposta na parede do cilindro, definida por

$\Gamma_D = \{(x, y, z) \in \mathbb{R}^3 \mid x^2 + y^2 = 2^2 ; z \in [0, 4]\}$. A solução analítica do problema é dada por $u(\mathbf{x}) = -\text{sen}(x^2) + \text{sen}(r^2 - y^2)$, $\mathbf{x} \in \Omega$.

Destaques da Implementação

O cilindro é gerado pelo comando `gmsh.model.occ.addCylinder`, que requer as coordenadas dos extremos do eixo central e o raio:

```
1 background = gmsh.model.occ.addCylinder(c_x,c_y,c_z,
2                                           c_x,c_y, height, r=radius)
```

A geometria circular deste problema requer um tratamento cuidadoso na imposição da condição de contorno de Dirichlet. A superfície cilíndrica pode ser identificada pela equação $(x - c_x)^2 + (y - c_y)^2 = r^2$, onde *eixo_x* e *eixo_y* c_x e c_y representam as coordenadas do eixo central do cilindro, com $z \in [0, 4]$:

```
1 # ----- CONDIÇÕES DE CONTORNO -----
2 def boundary(x):
3     r = np.sqrt((x[0]-c_x)**2 + (x[1]-c_y)**2)
4     return np.isclose(r, radius, rtol=1e-4, atol=1e-6)
5
6 dofs_circle = fem.locate_dofs_geometrical(V, boundary)
7 bcs = [fem.dirichletbc(u_aux, dofs_circle)]
```

Para mais detalhes, consulte o código completo no Apêndice A.7.

Resultados e análises

A Figura 3.15 apresenta a solução aproximada no espaço P_1 , representada por um gráfico de cores. Destaca-se o uso de uma malha não estruturada, que permite um melhor ajuste próximo à fronteira circular.

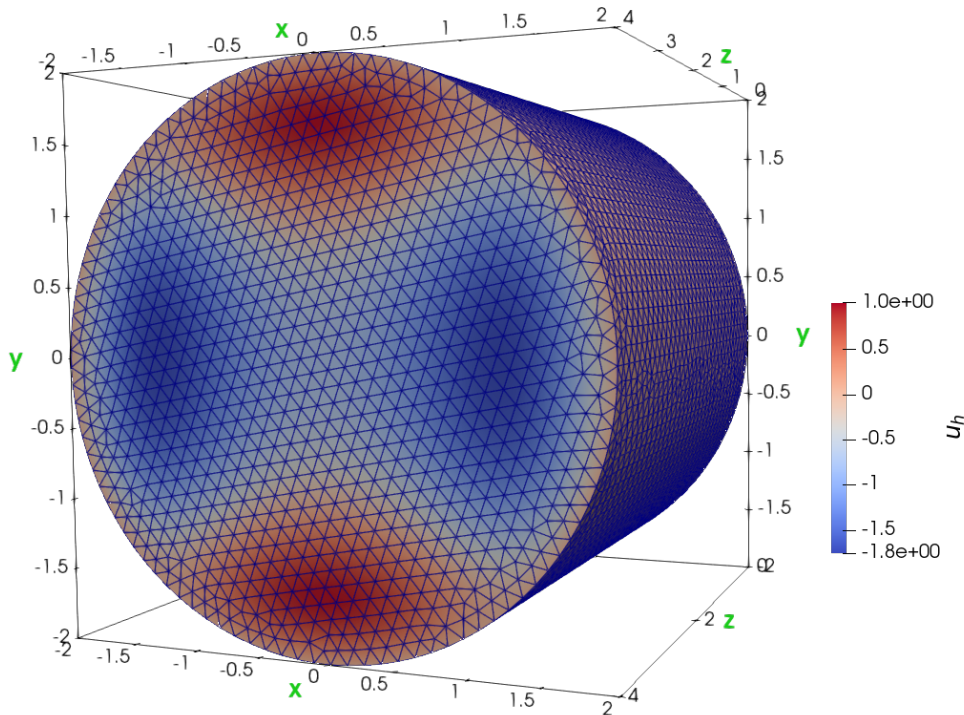


Figura 3.15: Distribuição da velocidade (solução aproximada) sobre uma geometria cilíndrica.

A solução analítica deste problema (3.13)-(3.14) é apresentada pela Figura 3.16.

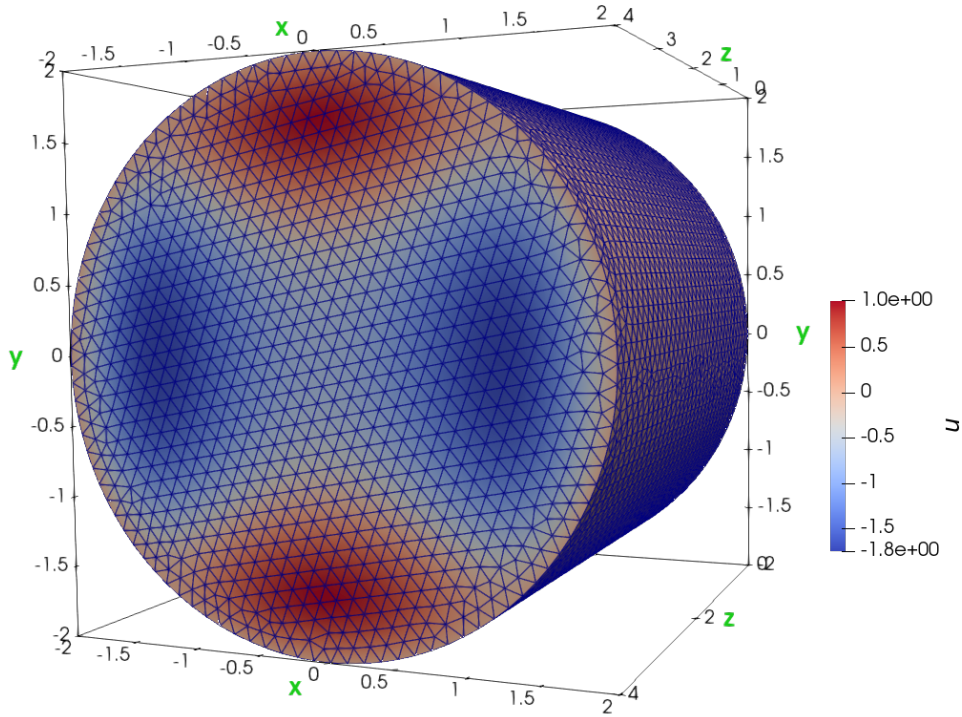


Figura 3.16: Distribuição da velocidade (solução analítica) sobre uma geometria cilíndrica.

Comparando as Figuras 3.15 e 3.16, observa-se que a solução aproximada mantém uma forte semelhança com a solução analítica, mesmo considerando a complexidade adicional imposta pela geometria cilíndrica. Essa análise qualitativa sugere que o método de elementos finitos se adapta bem a domínios tridimensionais com bordas curvas, capturando adequadamente o comportamento da solução.

A análise quantitativa foi realizada utilizando a norma L^2 para diferentes refinamentos de malha, avaliando o comportamento da convergência. Os valores da ordem de convergência foram arredondados para duas casas decimais, e os resultados estão apresentados na tabela 3.7.

Tabela 3.7: Análise de convergência do problema 3D com geometria cilíndrica.

n	h	$\ e\ _{L^2}$	q
3930	0,796386	9,17e-01	-
28960	0,413822	2,41e-01	1,93
225856	0,229124	5,92e-02	2,03
1788217	0,111945	1,47e-02	2,00

Os resultados apresentados na tabela mostram a redução da norma do erro à medida em que a malha é refinada, evidenciando a convergência do método. A ordem de convergência obtida é próxima de 2, o que está de acordo com a taxa teórica prevista para o método utilizado.

Os resultados de ambos os problemas 3D sugerem que a ferramenta FEniCSx, por meio do método de elementos finitos, obtém soluções precisas para problemas tridimensionais, mesmo em geometrias irregulares como no caso cilíndrico.

3.2 Equação do calor

Após abordar problemas estacionários na seção anterior, passamos agora à análise de problemas transientes, nos quais a solução T varia ao longo do tempo. O primeiro problema apresentado é modelado pela equação do calor, amplamente utilizada para descrever a difusão térmica em um domínio ao longo do tempo.

Considere o problema com domínio espacial $\Omega = [0, 1]^2$ e as fronteiras de Dirichlet e Neumann são definidas, respectivamente, como $\Gamma_D = \{(0, y, t), (1, y, t) \mid y, t \in [0, 1]\}$ e $\Gamma_N = \{(x, 0, t), (x, 1, t) \mid x, t \in [0, 1]\}$, descrito por

$$\left\{ \begin{array}{ll} \frac{\partial T(x, y, t)}{\partial t} - \nabla \cdot (\mu \nabla T(x, y, t)) = f(x, y, t), & (x, y, t) \in \Omega \times [0, t_f], \\ T(x, y, t) = 0, & (x, y, t) \in \Gamma_D \times [0, t_f], \\ -\mu(\mathbf{x}) \nabla T(x, y, t) \cdot \mathbf{\tilde{n}} = 0, & (x, y, t) \in \Gamma_N \times [0, t_f], \\ T(x, y, 0) = 0, & (x, y) \in \Omega \text{ (condição inicial)}, \end{array} \right. \quad (3.15)$$

onde a variável $\mathbf{x} = (x, y)$ representa as coordenadas espaciais, enquanto $T : \Omega \times [0, t_f] \rightarrow \mathbb{R}$ representa a temperatura. Neste exemplo, adotamos $f(x, y, t) = -x^2 + x + 2t\mu$, $\mu(x, y) = 1 \text{ Pa} \cdot \text{s}$ e consideramos a solução manufaturada $T(x, y, t) = -x(x-1)t$, a qual permite verificar a precisão do método numérico implementado.

Destaques da Implementação

Problemas transientes implicam em adaptações na implementação numérica, exigindo modificações nos códigos para refletir a evolução temporal da solução.

O tamanho do passo Δt é limitado pela condição de estabilidade (2.53), de modo que $\Delta t \leq \Delta t_{max}$, onde Δt_{max} depende de h_{min} . Para garantir a estabilidade mesmo com erros de arredondamento, adotou-se um fator de segurança, reduzindo Δt_{max} . Além disso, a dependência explícita de h_{min} assegura que, ao refinar a malha espacial, Δt também será ajustado, preservando a estabilidade do método:

```

1  #----- Calculo do valor de hmin -----
2  tdim = msh.topology.dim #dimensão topológica
3  num_cells = msh.topology.index_map(tdim).size_local + msh.topology.
   index_map(tdim).num_ghosts #número de elementos
4  msh.topology.create_connectivity(tdim-1, tdim)
5  h_e = msh.h(tdim-1, np.arange(num_cells))
6  h_min = min(h_e)
7  fator = 0.025
8
9  dt_value = fator * (2* h_min**2)/k_value #para problema 2D
10 dt = fem.Constant(msh, dt_value)

```

A formulação variacional depende do método de integração numérica adotado (Euler explícito e implícito), influenciando a forma como os termos bilinear e linear são tratados em cada caso:

```

1  if metodo=="explicito":
2      #----- FORMA BILINEAR -----
3      a = inner(u, v) * dx
4      #----- FORMA LINEAR -----
5      L = dt * inner(f, v) * dx + inner(u0, v) * dx - dt * inner(k*grad(u0),
   grad(v)) * dx
6

```

```

7  if metodo=="implicito":
8      #----- FORMA BILINEAR -----
9      a = inner(u, v) * dx + dt * inner(k*grad(u), grad(v)) * dx
10     #----- FORMA LINEAR -----
11     L = dt * inner(f, v) * dx + inner(u0, v) * dx
12
13     a = fem.form(a)
14     L = fem.form(L)

```

A evolução temporal requer o cálculo de u_h a cada passo no tempo. Como a matriz A independe do tempo, ela será calculada uma única vez da forma:

```

1  A = assemble_matrix(a, bcs=bcs)
2  A.assemble()

```

Define-se o solver que será utilizado para resolver o sistema linear:

```

1  #DEFININDO O SOLVER
2  solver = PETSc.KSP().create(mpi4py.MPI.COMM_WORLD)
3  solver.setOperators(A)
4  solver.setType(PETSc.KSP.Type.PREONLY)
5  pc = solver.getPC()
6  pc.setType("lu")
7  # https://www.mcs.anl.gov/petsc/petsc4py-current/docs/apiref/petsc4py.
   PETSc.PC.Type-class.html

```

Define-se a função uh , vetor b , tempo inicial $t = 0$ e tempo final $t_{max} = 1$:

```

1  uh = fem.Function(V)
2  uh.name = "temperatura"
3  b = create_vector(L)
4  t = 0
5  t_max = 1

```

A evolução temporal é implementada por meio de um loop *while*, que itera enquanto $t < t_{max}$. A cada passo no tempo, a função f , que depende de t , é atualizada, o lado direito da equação é montado, as condições de contorno são aplicadas, a solução do problema é computada e os resultados são armazenados em arquivo.

```

1  while t < t_max:
2      t += dt_value
3
4      #ATUALIZA f
5      if metodo=="explicito":
6          f.interpolate(partial(f_exp, t-dt_value))
7      if metodo=="implicito":
8          f.interpolate(partial(f_exp, t))
9
10     #MONTA O LADO DIREITO DA EQUAÇÃO (RHS)
11     with b.localForm() as loc_b:
12         loc_b.set(0)
13     fem.petsc.assemble_vector(b, L)
14
15     #APLICA A CONDIÇÃO DE CONTORNO
16     fem.petsc.apply_lifting(b, [a], [bcs])
17     b.ghostUpdate(addv=PETSc.InsertMode.ADD_VALUES, mode=PETSc.ScatterMode.
   REVERSE)
18     fem.petsc.set_bc(b, bcs)
19
20     #RESOLVE O SISTEMA LINEAR
21     uh.x.array[:] = 0
22     solver.solve(b, uh.x.petsc_vec)
23     uh.x.scatter_forward()

```

```

24
25 #ATUALIZA u_0
26 u0.x.array[:] = uh.x.array
27
28 #SALVA A SOLUÇÃO NO TEMPO t
29 file.write_function(uh, t=t)
30 file.close()

```

O código completo está disponível no Apêndice A.8.

Resultados e análises

A discretização espacial foi realizada com o método de elementos finitos, utilizando elementos triangulares lineares. Para a evolução temporal, empregou-se o método implícito, que é conhecido por sua estabilidade incondicional. A malha utilizada contém 200 elementos no total, e a implementação foi feita em PythonTM com a biblioteca FEniCSx.

Na Figura 3.17, apresentamos a solução aproximada no espaço P_1 obtida para diferentes instantes de tempo $t = 0\text{ s}$, $t = 0,7\text{ s}$, $t = 0,85\text{ s}$ e $t = 1,0\text{ s}$, representada através de gráficos de cores. Observa-se que, com o avanço do tempo, a temperatura no centro do domínio ($x = 0,5\text{ m}$) aumenta gradualmente, enquanto o calor se difunde em direção às bordas ($x = 0\text{ m}$ e $x = 1\text{ m}$), conforme esperado.

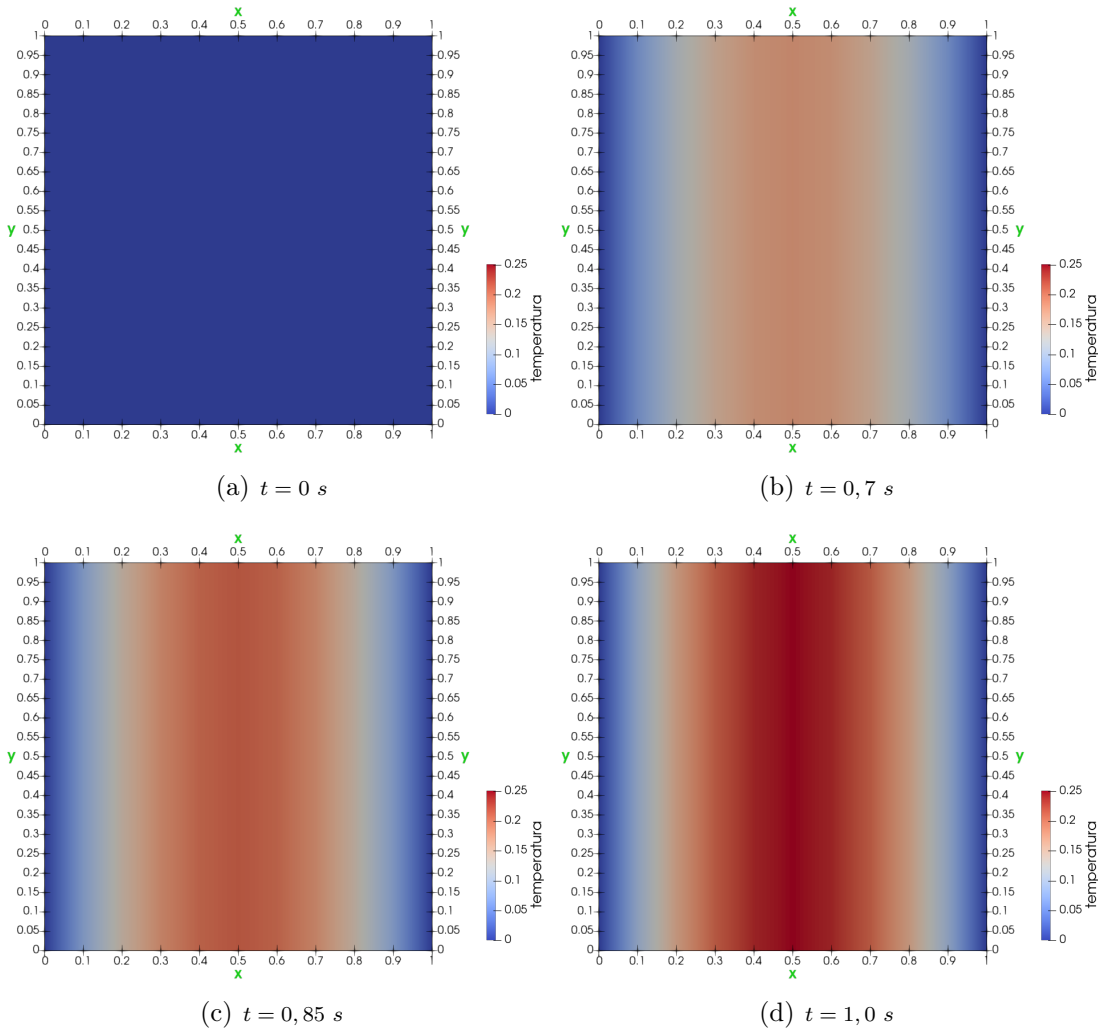


Figura 3.17: Distribuição da temperatura utilizando o método implícito em diferentes tempos.

A análise de convergência foi realizada a partir da verificação do erro utilizando a norma L^2 , calculada conforme (2.1), ou seja, $\|e\|_{L^2} = \|u - u_h\|_{L^2}$. A solução manufaturada adotada para este estudo é dada por $u(x, y, t) = -x(x-1)t$, a qual varia com o tempo t e, neste caso, não atinge um estado estacionário. Por este motivo, optou-se por calcular a norma do erro no instantes $t = 1$ s, por uma questão de conveniência, resultando na expressão:

$$\|e\|_{L^2} = \|u(x, y, 1) - u_h(x, y, 1)\|_{L^2}.$$

A discretização inicial da malha foi realizada com 50 triângulos, e o refinamento foi realizado de forma conjunta, refinando a malha espacial h e a temporal Δt , para cada simulação, sempre respeitando a condição de estabilidade indicada em (2.53), como requer o método explícito.

Os valores das normas do erro para os métodos implícito $\|e_I\|_{L^2}$ e explícito $\|e_E\|_{L^2}$ são apresentados na Tabela 3.8. Também foram calculadas as ordens de convergência espacial ($q_{L^2, h}$) e temporal ($q_{L^2, \Delta t}$), utilizando a fórmula (3.4), com h substituído por Δt para a análise temporal. O sistema linear resultante foi resolvido por meio da decomposição LU .

Tabela 3.8: Análise de convergência do problema envolvendo a equação do calor.

n	h	Δt	$\ e_E\ _{L^2}$	$\ e_I\ _{L^2}$	$q_{L^2, h}$	$q_{L^2, \Delta t}$
50	0,200	2,00e-03	7,30e-03	7,30e-03	-	-
200	0,100	5,00e-04	1,83e-03	1,83e-03	2,00	1,00
800	0,050	1,25e-04	4,56e-04	4,56e-04	2,00	1,00
3200	0,025	3,13e-05	1,14e-04	1,14e-04	2,00	1,00

Com o refinamento das malhas, observa-se que a norma do erro para ambos os métodos (explícito e implícito) diminui, indicando que ambos estão convergindo para a solução analítica. Os valores da norma do erro para os métodos são praticamente idênticos, quando considerados as casas decimais apresentadas na tabela, o que sugere que a escolha entre os métodos não afeta significativamente o erro calculado. Isso implica que a ordem de convergência é a mesma para ambos os métodos. As ordens de convergência espacial e temporal também podem ser analisadas comparando a inclinação das linhas gráficas que representam as normas do erro com linhas de referência de ordens 1 e 2, que utilizam a escala log-log. Observa-se o comportamento da norma do erro em função do espaço h (Figura 3.18) e do passo no tempo Δt (Figura 3.19).

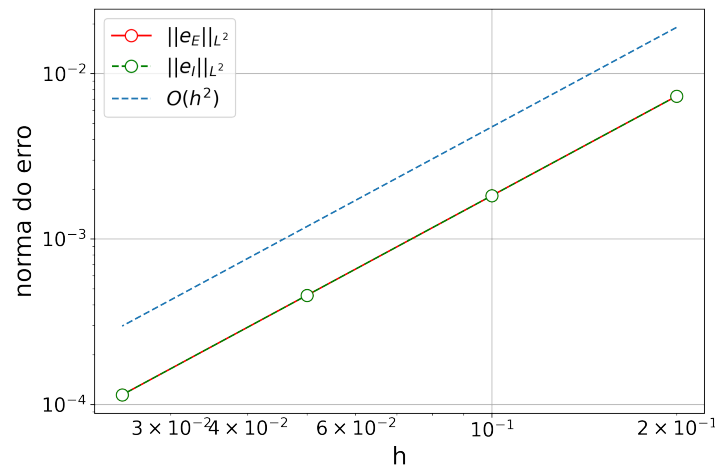


Figura 3.18: Decaimento da norma do erro dos métodos explícito e implícito em relação ao espaçamento h .

Os resultados apresentados na Figura 3.18 mostram que ambos os métodos (implícito e explícito) apresentam inclinações semelhantes a $O(h^2)$, sugerindo que a ordem de convergência espacial é aproximadamente 2.

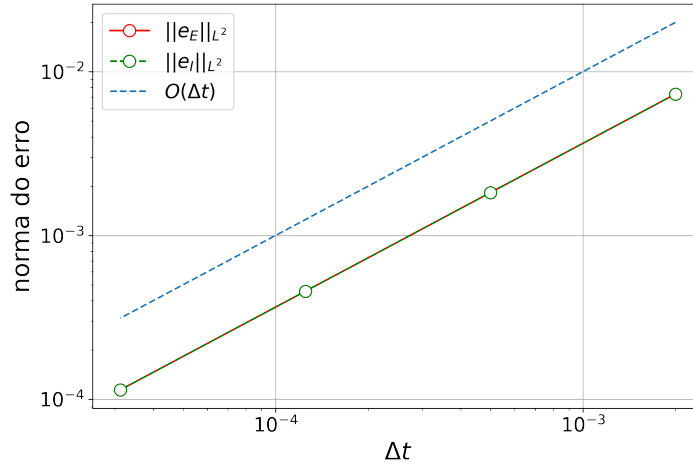


Figura 3.19: Decaimento da norma do erro dos métodos explícito e implícito em relação ao espaçamento Δt .

A Figura 3.19 apresenta a norma do erro de ambos os métodos em relação ao passo temporal Δt e neste caso a inclinação se assemelha com a linha $O(\Delta t)$, mostrando uma ordem de convergência temporal próxima de 1.

Os resultados obtidos confirmam o comportamento esperado para os métodos analisados. A convergência espacial manteve a ordem 2, mesmo no contexto de um problema transiente, enquanto a convergência temporal de ordem 1 está em plena concordância com a teoria, dado que tanto o método explícito, quanto o implícito possuem essa ordem.

3.3 Problema five-spot simplificado

A extração de petróleo é um dos principais exemplos de aplicação de problemas envolvendo fluidos. Uma das técnicas utilizadas nesse processo consiste na injeção de água em um poço, denominado poço injetor, com o objetivo de pressurizar o reservatório e deslocar o óleo em direção a outro poço, chamado poço produtor.

A Figura 3.20, à esquerda, ilustra um reservatório de petróleo onde ocorre esse processo. Os círculos em azul representam os *poços injetores*, enquanto os círculos pretos indicam os *poços produtores*. Esse método de extração é conhecido como *five-spot*, pois, ao tomar um poço injetor como referência, há quatro poços produtores dispostos ao seu redor.

Para fins de simplificação, considera-se uma representação reduzida do domínio, denominada *a quarter of the five-spot*, mostrada à direita da Figura 3.20. Esse modelo inclui um único poço injetor e um poço produtor, assumindo fluxo nulo nas bordas e valores prescritos para as pressões de entrada (no poço injetor) e saída (no poço produtor).

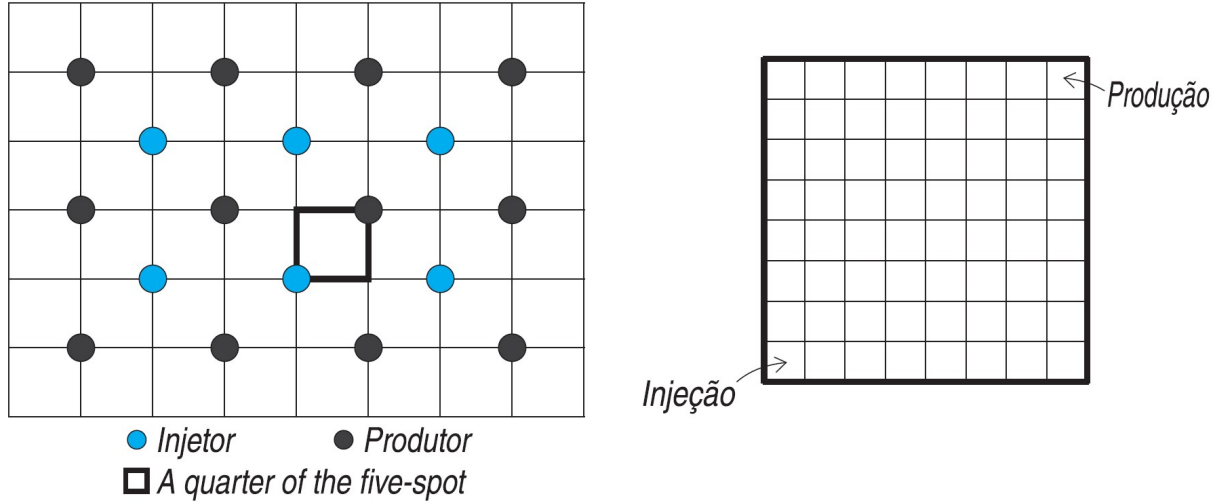


Figura 3.20: Domínio do tipo *a quarter of the five-spot*. Figura extraída do livro [1].

A modelagem deste problema foi baseada nos exemplos apresentados nos livros [1; 25]. Embora o problema *five-spot* seja comumente tratado em sua formulação mista para representar com maior precisão o escoamento em meios porosos, aqui adotamos uma abordagem simplificada, para enfatizar a implementação no FEniCSx. Considere o problema de domínio espacial $\Omega = [0, 1]^2$, descrito por

$$\left\{ \begin{array}{ll} \frac{\partial p(x, y, t)}{\partial t} - \nabla \cdot [\mathcal{A} \nabla p(x, y, t)] &= 0, \quad (x, y, t) \in \Omega \times [0, T], \\ p(0, 0, t) &= 1, \quad (x, y, t) \in \Gamma_D \times [0, T], \\ p(1, 1, t) &= 0, \quad (x, y, t) \in \Gamma_D \times [0, T], \\ p(x, y, t) &= 0, \quad (x, y, t) \in \Gamma_N \times [0, T], \\ p(x, y, 0) &= 0, \quad (x, y) \in \Omega \text{ (condição inicial)}, \end{array} \right. \quad (3.16)$$

onde $T = 1$ s é o tempo final, $\Gamma_N = \{(x, 0, t), (x, 1, t), (0, y, t), (1, y, t) \mid x, y, t \in [0, 1]\}$ é a fronteira de Neumann, $\mathcal{A} = \frac{K}{\varphi \mu \mathcal{C}} \text{ m}^2/\text{s}$ em que K, φ, μ e $\mathcal{C}$ representam respectivamente, os coeficientes de permeabilidade (m^2), porosidade, viscosidade ($\text{Pa} \cdot \text{s}$) e compressibilidade ($1/\text{Pa}$), que são propriedades do meio poroso e do fluido e neste caso, trata-se de um meio homogêneo, de forma que $\mathcal{A}$ é constante e $p : \Omega \times [0, 1] \rightarrow \mathbb{R}$ é a função incógnita.

Detalhes da Implementação

Os poços injetor e produtor exigem uma implementação diferenciada, pois impõem condições de Dirichlet nos pontos (0,0) e (1,1), ao invés de impor em arestas. A seguir, apresenta-se a abordagem utilizada para localizar esses pontos na malha:

```

1 #----- CONDIÇÕES DE CONTORNO -----
2 u_inflow = fem.Constant(msh, 1.0)
3 u_outflow = fem.Constant(msh, 0.0)
4
5 def locate_dof_at_point(msh, point):
6     coords = msh.geometry.x[:, :2]
7     idx = np.where(np.all(np.isclose(coords, point), axis=1))[0]
8     if idx.size > 0:
9         return [idx[0]] # Retorna o primeiro índice encontrado
10    return []
11
12 #LOCALIZANDO DoFs PARA OS PONTOS (0, 0) e (1, 1)
13 dofs_inflow = locate_dof_at_point(msh, [0.0, 0.0])

```

```

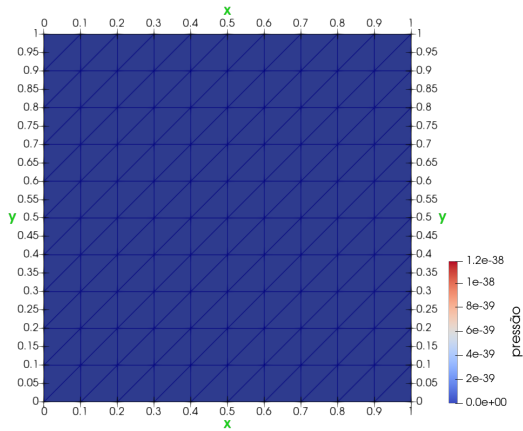
14  dofs_outflow = locate_dof_at_point(msh, [1.0, 1.0])
15
16  #VERIFICANDO SE OS DoFs FORAM ENCONTRADOS
17  if not dofs_inflow:
18      print("DoF para (0,0) não encontrado.")
19  if not dofs_outflow:
20      print("DoF para (1,1) não encontrado.")
21
22  dofs_inflow = np.array(dofs_inflow, dtype=np.int32)
23  dofs_outflow = np.array(dofs_outflow, dtype=np.int32)
24
25  bcs = []
26  if dofs_inflow:
27      bcs.append(fem.dirichletbc(u_inflow, dofs_inflow, V))
28  if dofs_outflow:
29      bcs.append(fem.dirichletbc(u_outflow, dofs_outflow, V))

```

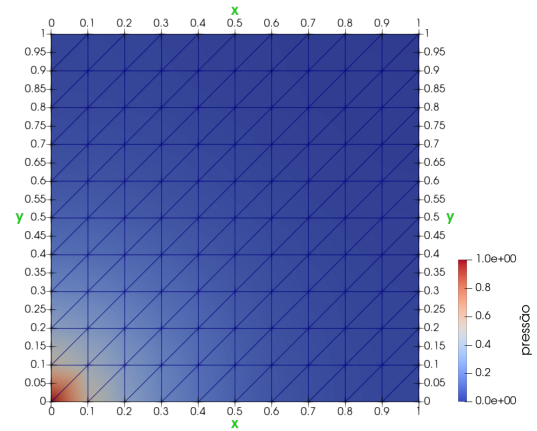
O código completo está disponível no Apêndice A.9.

Resultados e análises

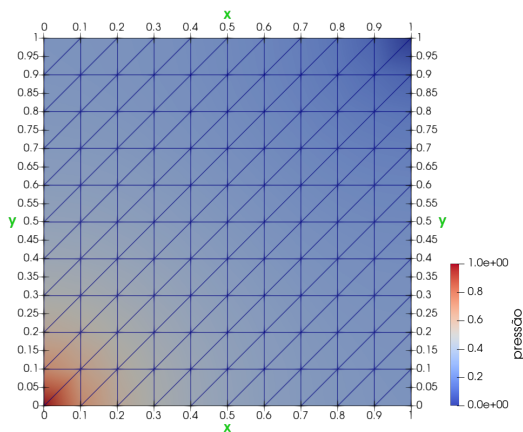
A solução aproximada no espaço P_1 foi obtida utilizando elementos finitos triangulares lineares e um esquema temporal implícito, considerando $n = 200$ elementos. A Figura 3.21 apresenta a solução aproximada em um gráfico de cores para diferentes instantes de tempo $t = 0\text{ s}$, $t = 0,3\text{ s}$, $t = 2\text{ s}$ e $t = 10\text{ s}$, escolhidos de modo a evidenciar a variação da pressão ao longo do domínio.



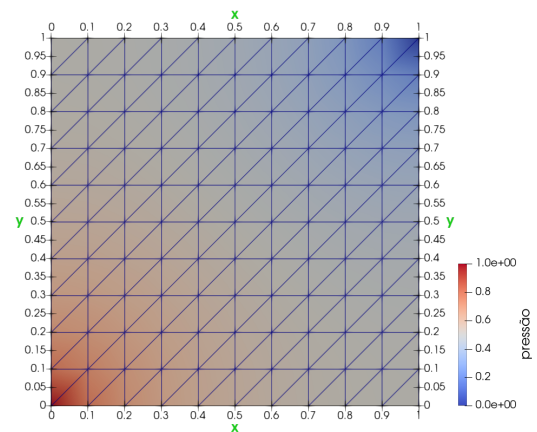
(a) $t = 0\text{ s}$



(b) $t = 0,3\text{ s}$



(c) $t = 2\text{ s}$



(d) $t = 10\text{ s}$

Figura 3.21: Campo da pressão utilizando o método implícito em diferentes tempos.

3.4 Equação de Advecção-Difusão

Considere o problema de advecção-difusão, de domínio espacial $\Omega = [0, 1]^2$, representado da forma:

$$\left\{ \begin{array}{ll} \frac{\partial T(x, y, t)}{\partial t} + \mathcal{U} \cdot \nabla T(x, y, t) - \nabla \cdot [\mu \nabla T(x, y, t)] & = 0, \quad (x, y, t) \in \Omega \times [0, t_f], \\ T(0, y, t) & = 0, \quad (x, y, t) \in \Gamma_D \times [0, t_f], \\ T(1, y, t) & = 1, \quad (x, y, t) \in \Gamma_D \times [0, t_f], \\ -\mu \nabla T(x, y, t) \cdot \mathbf{\tilde{n}} & = 0, \quad (x, y, t) \in \Gamma_N \times [0, t_f], \\ T(x, y, 0) & = 0, \quad (x, y) \in \Omega \text{ (condição inicial)}, \end{array} \right. \quad (3.17)$$

onde $t_f = 1$ s é o tempo final, $\Gamma_D = \{(0, y, t), (1, y, t) \mid y, t \in [0, 1]\}$ e $\Gamma_N = \{(x, 0, t), (x, 1, t) \mid x, t \in [0, 1]\}$ são, respectivamente, as fronteiras de Dirichlet e Neumann, $\mu = 1$ Pa · s é o coeficiente de difusão, $\mathcal{U} : \Omega \times [0, 1] \rightarrow \mathbb{R}^2$, $\mathcal{U}(x, y, t) = (1, 0)$ é um vetor velocidade com componentes em m/s (função coeficiente de advecção) e $T : \Omega \times [0, 1] \rightarrow \mathbb{R}$ é a temperatura ($^{\circ}\text{C}$). E consideramos a solução manufaturada $T(x, y, t) = \frac{e^x - 1}{e - 1}$, onde e é o número de Euler.

Detalhes da Implementação

A seguir, apresenta-se a implementação da formulação variacional do problema de advecção-difusão:

```

1  #----- FORMA LINEAR -----
2  a = inner(u, v) * dx + dt * inner(inner(alpha_vec, grad(u)), v) * dx + dt
   * inner(k*grad(u), grad(v)) * dx
3  #----- FORMA BILINEAR -----
4  L = dt * inner(f, v) * dx + inner(u0, v) * dx

```

O código completo está disponível no Apêndice A.10.

Resultados e análises

A solução aproximada foi obtida utilizando elementos finitos triangulares lineares e um esquema temporal implícito, considerando $n = 200$ elementos. A Figura 3.22 apresenta a solução aproximada no espaço P_1 em um gráfico de cores para diferentes instantes de tempo ($t = 0,001$ s, $t = 0,03$ s, $t = 0,1$ s e $t = 1$ s), escolhidos de modo a evidenciar a variação da temperatura ao longo do domínio.

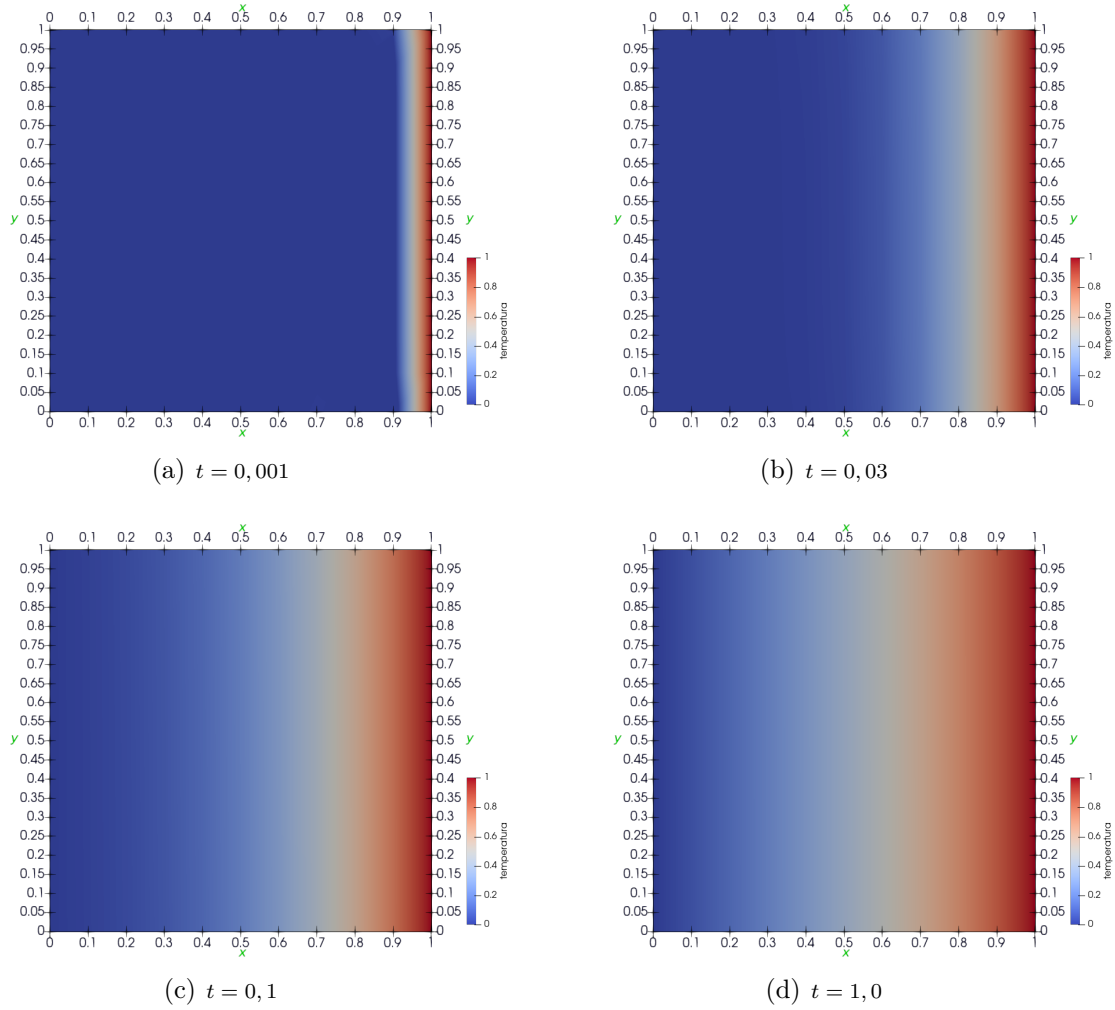


Figura 3.22: Evolução térmica no espaço P_1 , utilizando o método implícito.

A análise de convergência foi realizada a partir da verificação do erro utilizando a norma L^2 , no instante $t = 1s$ conforme expressão:

$$\| e \|_{L^2} = \| u(x, y, 1) - u_h(x, y, 1) \|_{L^2} .$$

As ordens de convergência espacial ($q_{L^2,h}$) e temporal ($q_{L^2,\Delta t}$) foram calculadas, utilizando a fórmula (3.4), com h substituído por Δt para a análise temporal. Para a discretização no tempo, utilizou-se o método implícito, e o sistema linear resultante foi resolvido por meio da decomposição LU .

Tabela 3.9: Análise de convergência do problema modelado pela equação de advecção-difusão.

n	h	Δt	$\ e\ _{L^2}$	$q_{L^2,h}$	$q_{L^2,\Delta t}$
50	0,200	2,00e-03	3,60e-03	-	-
200	0,100	5,00e-04	8,90e-04	2,01	1,01
800	0,050	1,25e-04	2,10e-04	2,05	1,03
3200	0,025	3,10e-05	4,70e-05	2,20	1,10

De acordo com a tabela 3.9, à medida que a malha é refinada, observa-se uma redução na norma do erro, indicando a convergência para a solução analítica. A convergência espacial de ordem 2 foi preservada mesmo no contexto de um problema transiente e na

formulação da equação de advecção-difusão. Além disso a convergência temporal de ordem 1 está em plena concordância com a teoria, uma vez que o método implícito possui essa precisão. Esses resultados sugerem que a ferramenta FEniCSx, por meio do método de elementos finitos, é capaz de reproduzir com ótima precisão comportamento da solução em problemas de advecção-difusão.

Escoamento Monofásico em Meios Porosos

O escoamento monofásico em meios porosos é descrito pelas equações que representam as leis de Darcy e da conservação de massa, que formam um problema misto, no qual a velocidade $\mathbf{v}$ e a pressão p devem ser calculadas simultaneamente. O espaço $H(\text{div}; \Omega)$ é amplamente utilizado para o estudo de métodos mistos e híbridos devido à sua capacidade de representar fluxos com divergência bem definida [26]. Os elementos Raviart-Thomas ($RT_k(\Omega_e)$), que são subespaços de $H(\text{div}; \Omega)$, são especialmente importantes para garantir a continuidade dos fluxos normais entre os elementos adjacentes, uma propriedade essencial para a conservação de massa global.

4.1 Espaços $H(\text{div}; \Omega)$

O espaço $H(\text{div}; \Omega)$ é definido como o conjunto de campos vetoriais cujas componentes pertencem ao espaço $L^2(\Omega)$ e cuja divergência também pertence [26; 27]. Sua representação matemática é escrita da forma:

$$H(\text{div}; \Omega) = \{\mathbf{v}, \mathbf{v} \in [L^2(\Omega)]^n : \nabla \cdot \mathbf{v} \in L^2(\Omega)\}, \quad (4.1)$$

com a norma apropriada:

$$\|\mathbf{v}\|_{\text{div}, \Omega} = \left(\int_{\Omega} \mathbf{v} \cdot \mathbf{v} \, d\Omega + \int_{\Omega} (\nabla \cdot \mathbf{v})^2 \, d\Omega \right)^{\frac{1}{2}} \quad (4.2)$$

4.2 Espaço $RT_k(\Omega_e)$

O espaço Raviart-Thomas $RT_k(\Omega_e)$ foi introduzido por [28] e é amplamente utilizado em métodos mistos, pois este espaço garante que as funções aproximadas possuam divergência bem definida e permite a conservação local da massa. Seja Ω_e um elemento triangular ou tetrahedral, este espaço (ver também [26; 27]) é definido por:

$$RT_k(\Omega_e) = (P_k(\Omega_e))^n + \mathbf{x}P_k(\Omega_e), \quad (4.3)$$

onde n é a dimensão do problema.

A dimensão de $RT_k(\Omega_e)$ pode ser facilmente calculada por:

$$\dim RT_k(\Omega_e) = \begin{cases} (k+1)(k+3), & \text{para } n = 2, \\ \frac{1}{2}(k+1)(k+2)(k+4), & \text{para } n = 3. \end{cases}$$

No caso planar mais simples ($k = 0$), o espaço $RT_0(\Omega_e)$ é definido como:

$$RT_0(\Omega_e) = [P_0(\Omega_e)]^2 + [x_1, x_2]^T P_0(\Omega_e),$$

em que as funções vetoriais possuem a forma:

$$\mathbf{v} = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} + b \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad (4.4)$$

com c_1 , c_2 e b sendo coeficientes. Os graus de liberdade, que em RT_0 são definidos como as componentes normais de $\mathbf{v}$ em Γ , estão representados na Figura 4.1.

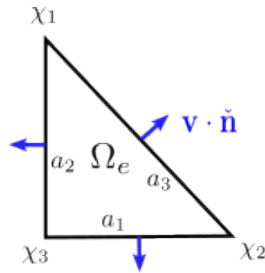


Figura 4.1: Representação dos graus de liberdade em RT_0 .

O espaço RT_0 possui dimensão 3 e cada uma das funções base ω_{ei} está associada a uma aresta a_i distinta do elemento Ω_e . Essas bases (ver [26]) podem ser expressas como:

$$\omega_{ei} = \frac{1}{h_i} ((x, y) - (x_i, y_i)), \quad i = 1, 2, 3,$$

onde (x_i, y_i) são as coordenadas do vértice χ_i , e h_i representa a altura do triângulo relativa à aresta a_i , ou seja, a distância perpendicular da aresta a_i ao vértice χ_i . As funções base em Ω_e são ilustradas através da Figura 4.2.

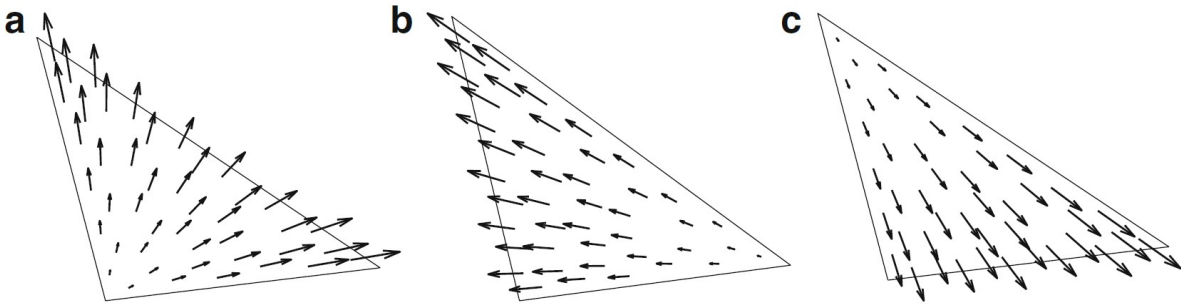


Figura 4.2: Representação das Funções base de RT_0 em Ω_e . Figura extraída do livro [2].

As funções bases são construídas de forma a garantir a continuidade da componente normal ao longo da aresta entre dois elementos, assegurando que as funções de RT_0 pertençam à $H(\text{div}; \Omega)$ [2]. Isso é fundamental para problemas onde a conservação de massa é exigida.

Os espaços Raviart-Thomas são utilizados em softwares para a resolução de problemas baseados em métodos mistos. No FEniCSx, por exemplo, eles são implementados para permitir uma formulação direta de problemas de fluxo em meios porosos. A biblioteca oferece suporte para o elemento de Raviart-Thomas.

4.3 Condição LBB

Ladyzhenskaya, Babuska e Brezzi determinaram uma condição de compatibilidade, conhecida como condição LBB (ou inf-sup) que os espaços contínuos e discretos devem satisfazer para garantir a estabilidade para os métodos mistos [29].

Esta condição afirma que os espaços da pressão e velocidade não podem ser escolhidos de forma arbitrária. Além disso, provar que um par pressão-velocidade satisfaz a condição LBB é, em geral, uma tarefa desafiadora. Os livros [26; 27; 30] apresentam diversos pares que são comprovadamente estáveis. No presente trabalho utilizaremos o par composto por P_0 (discreto e descontínuo entre os elementos) e RT_0 .

4.4 Formulação Variacional do problema misto

Problemas mistos envolvem duas ou mais variáveis desconhecidas e elas precisam ser determinadas simultaneamente. Um exemplo é o problema de fluxo em meios porosos, que envolve as variáveis desconhecidas $\mathbf{v}$ (velocidade em m/s) e p (pressão em Pa). A formulação deste problema é mostrada abaixo.

Seja $\Omega \subset \mathbb{R}^d$, com $d \in \{1, 2, 3\}$, o problema diferencial consiste em encontrar a velocidade $\mathbf{v} : \Omega \rightarrow \mathbb{R}^d$ e a pressão $p : \Omega \rightarrow \mathbb{R}$, que satisfaçam:

$$\begin{cases} \frac{\mu}{K} \mathbf{v} + \nabla p = 0, & \text{em } \Omega, \\ \nabla \cdot \mathbf{v} = f, & \text{em } \Omega, \\ p = g, & \text{em } \Gamma_D, \\ \mathbf{v} \cdot \mathbf{\check{n}} = h, & \text{em } \Gamma_N, \end{cases} \quad (4.5)$$

onde $\mu \in \mathbb{R}$ é a viscosidade do fluido ($Pa \cdot s$), $K \in \mathbb{R}$ a permeabilidade do meio (m^2), $f : \Omega \rightarrow \mathbb{R}$ o termo fonte e $g : \Gamma_D \rightarrow \mathbb{R}$ e $h : \Gamma_N \rightarrow \mathbb{R}$ as condições de contorno.

Para obter a formulação variacional, inicia-se multiplicando a primeira equação do sistema (4.5) por uma função peso $\mathbf{w}$, em que $W = \{\mathbf{w} \in H(div; \Omega) \mid \mathbf{w}|_{\Gamma_N} = 0\}$ e integrando sobre Ω .

$$\int_{\Omega} \frac{\mu}{K} \mathbf{v} \cdot \mathbf{w} \, dx + \int_{\Omega} \nabla p \cdot \mathbf{w} \, dx = 0.$$

Integrando por partes:

$$\int_{\Omega} \frac{\mu}{K} \mathbf{v} \cdot \mathbf{w} \, dx + \int_{\Omega} \nabla \cdot (p \mathbf{w}) \, dx - \int_{\Omega} p \nabla \cdot \mathbf{w} \, dx = 0.$$

Aplicando o Teorema de Gauss e separando as integrais na fronteira:

$$\int_{\Omega} \frac{\mu}{K} \mathbf{v} \cdot \mathbf{w} \, dx + \int_{\Gamma_D} \underbrace{p}_{=g} (\mathbf{w} \cdot \mathbf{\check{n}}) \, d\Gamma_D + \int_{\Gamma_N} p (\mathbf{w} \cdot \mathbf{\check{n}}) \, d\Gamma_N - \int_{\Omega} p \nabla \cdot \mathbf{w} \, dx = 0.$$

Resultando em:

$$\int_{\Omega} \frac{\mu}{K} \mathbf{v} \cdot \mathbf{w} \, dx - \int_{\Omega} p \nabla \cdot \mathbf{w} \, dx = - \int_{\Gamma_D} g (\mathbf{w} \cdot \mathbf{\check{n}}) \, d\Gamma_D. \quad (4.6)$$

Multiplicando a segunda equação do sistema (4.5) por uma função peso $q \in L^2(\Omega)$ e integrando sobre Ω :

$$\int_{\Omega} \nabla \cdot \mathbf{v} q \, dx = \int_{\Omega} f q \, dx. \quad (4.7)$$

Juntas, as equações (4.6) e (4.7) representam o sistema variacional (4.8) associado ao problema misto:

$$\begin{cases} \int_{\Omega} \frac{\mu}{K} \mathbf{v} \cdot \mathbf{w} \, dx - \int_{\Omega} p \nabla \cdot \mathbf{w} \, dx = - \int_{\Gamma_D} g(\mathbf{w} \cdot \check{\mathbf{n}}) \, d\Gamma_D, \\ - \int_{\Omega} q \nabla \cdot \mathbf{v} \, dx = - \int_{\Omega} f q \, dx. \end{cases} \quad (4.8)$$

4.5 Aproximação numérica e sistema linear

Para resolver numericamente o problema, as variáveis $\mathbf{v}$ e p são aproximadas por combinações lineares de funções de base, pertencentes a espaços finitos apropriados:

$$\mathbf{v} \approx \mathbf{v}_h = \sum_{j=1}^{N_v} V_j \boldsymbol{\omega}_j, \quad p \approx p_h = \sum_{j=1}^{N_p} P_j \sigma_j, \quad (4.9)$$

com $\boldsymbol{\omega}_j \in H(\text{div}, \Omega)$ e $\sigma_j \in L^2(\Omega)$. N_v e N_p representam as quantidades de graus de liberdade associadas, respectivamente, à velocidade e à pressão.

As funções-teste $\mathbf{w}$ e q também são escolhidas nos mesmos espaços, sendo:

$$\mathbf{w} = \boldsymbol{\omega}_i, \quad q = \sigma_i, \quad i = 1, 2, \dots, N_v \text{ ou } N_p. \quad (4.10)$$

A substituição dessas aproximações na formulação variacional resulta no sistema matricial:

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{B}^T & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{V} \\ \mathbf{P} \end{bmatrix} = \begin{bmatrix} \mathbf{F}_1 \\ \mathbf{F}_2 \end{bmatrix}, \quad (4.11)$$

onde $\mathbf{V} \in \mathbb{R}^{N_v \times 1}$ e $\mathbf{P} \in \mathbb{R}^{N_p \times 1}$ são vetores formados, respectivamente, pelos coeficientes das aproximações $\mathbf{v}_h$ e p_h . E as matrizes e vetores $\mathbf{A} \in \mathbb{R}^{N_v \times N_v}$, $\mathbf{B} \in \mathbb{R}^{N_v \times N_p}$, $\mathbf{B}^T \in \mathbb{R}^{N_p \times N_v}$, $\mathbf{0} \in \mathbb{R}^{N_p \times N_p}$, $\mathbf{F}_1 \in \mathbb{R}^{N_v \times 1}$, $\mathbf{F}_2 \in \mathbb{R}^{N_p \times 1}$ possuem seus elementos definidos por:

$$\begin{aligned} A_{ij} &= \int_{\Omega} \frac{\mu}{K} \boldsymbol{\omega}_j \cdot \boldsymbol{\omega}_i \, dx. \\ B_{ij} &= - \int_{\Omega} \sigma_j \nabla \cdot \boldsymbol{\omega}_i \, dx. \\ B_{ij}^T &= - \int_{\Omega} \sigma_i \nabla \cdot \boldsymbol{\omega}_j \, dx. \\ F_{1i} &= - \int_{\Gamma_D} g \boldsymbol{\omega}_i \cdot \check{\mathbf{n}} \, d\Gamma_D, \\ F_{2i} &= - \int_{\Omega} f \sigma_i \, dx. \end{aligned}$$

No caso bidimensional, o sistema matricial resultante pode ser resolvido com eficiência utilizando métodos diretos, como por exemplo a decomposição LU. Para problemas tridimensionais, o aumento significativo dos graus de liberdade torna os métodos diretos inviáveis, sendo recomendados métodos iterativos com preconditionadores. Discussões sobre a estrutura dos problemas mistos e métodos numéricos podem ser encontradas em [26].

4.6 Simulação do escoamento em um campo homogêneo

Considere o problema (4.5) definido no domínio $\Omega = [0, 1]^2$, com permeabilidade homogênea em todo o domínio, sendo $f(\mathbf{x}) = 2$ e $\frac{\mu}{K} = 1$. As condições de contorno são especificadas como $g(0, y) = g(1, y) = 0$ e $h(x, 0) = h(x, 1) = 0$.

Destaques da Implementação

A definição do espaço de função para problemas mistos envolvem uma implementação diferente:

```

1  el_u = element("RT", msh.basix_cell(), degree_u) #Raviart-Thomas
2  el_p = element("DG", msh.basix_cell(), degree_p) #Discontinuous Galerkin
3
4  #DEFININDO ELEMENTO MISTO
5  el_m = mixed_element([el_u, el_p])
6
7  #ESPAÇO DE FUNÇÃO
8  W = fem.functionspace(msh, el_m)
```

É importante destacar que os elementos de menor grau RT são referenciados como sendo de grau 0 por alguns autores como [2; 26; 27], convenção que também foi adotada neste trabalho. No entanto, no FEniCSx, esses elementos são referenciados como RT de grau 1 [13].

As definições das funções *trial* e *test* são realizadas:

```

1  #----- FUNÇÕES TRIAL E TEST -----
2  TrialF = TrialFunction(W)
3  TestF = TestFunction(W)
4  (u, p) = split(TrialF)
5  (v, q) = split(TestF)
```

A implementação das fronteiras pode ser feita enumerando-as a fim de identificá-las:

```

1  #----- CONDIÇÕES DE CONTORNO -----
2  # IDENTIFICANDO AS 4 FRONTEIRAS
3  boundaries = [(1, lambda x: np.isclose(x[0], 0)),
4               (2, lambda x: np.isclose(x[0], Lx)),
5               (3, lambda x: np.isclose(x[1], 0)),
6               (4, lambda x: np.isclose(x[1], Ly))]
7
8  facet_indices, facet_markers = [], []
9  fdim = msh.topology.dim - 1
10 for (marker, locator) in boundaries:
11     facets = mesh.locate_entities(msh, fdim, locator)
12     facet_indices.append(facets)
13     facet_markers.append(np.full_like(facets, marker))
14 facet_indices = np.hstack(facet_indices).astype(np.int32)
15 facet_markers = np.hstack(facet_markers).astype(np.int32)
16 sorted_facets = np.argsort(facet_indices)
17 facet_tag = mesh.meshtags(msh, fdim, facet_indices[sorted_facets],
18                             facet_markers[sorted_facets])
19
20 ds = Measure("ds", domain=msh, subdomain_data=facet_tag)
21 dx = ufl.dx
22
23 if (False):
24     one = fem.Constant(msh, 1.0)
25     for i in [1, 2, 3, 4]:
```

```

25     length_form = fem.form( one*ds(i) )
26     lengthside = fem.assemble_scalar(length_form)
27     print(lengthside)

```

Desta forma diferentes valores (condição de Dirichlet) podem ser implementados em suas respectivas fronteiras na forma bilinear:

```

1  n = FacetNormal(msh)
2  # ----- FORMA LINEAR -----
3  B = ((mu/k_value)*inner(u, v) - div(v)*p - div(u)*q) * dx
4  # ----- FORMA BILINEAR -----
5  L = - f * q * dx - pleft*dot(v,n)*ds(1) - pright*dot(v,n)*ds(2)

```

A solução do problema é armazenado na variável wh e então separa-se a solução da velocidade uh e da pressão ph .

```

1 (uh,ph) = wh.split()

```

O código completo está disponível no Apêndice A.11.

Resultados e análises

A solução aproximada deste problema foi obtida por meio da discretização utilizando o par de elementos finitos RT_0 para a velocidade e P_0 para a pressão, onde RT_0 é o espaço Raviart-Thomas de menor grau e P_0 é o espaço de aproximação para pressões descontínuas entre os elementos. A malha utilizada foi composta por $n = 12.800$ triângulos e os resultados apresentados nas Figuras 4.3 e 4.4, as quais ilustram, respectivamente, a magnitude da velocidade e o campo de pressão.

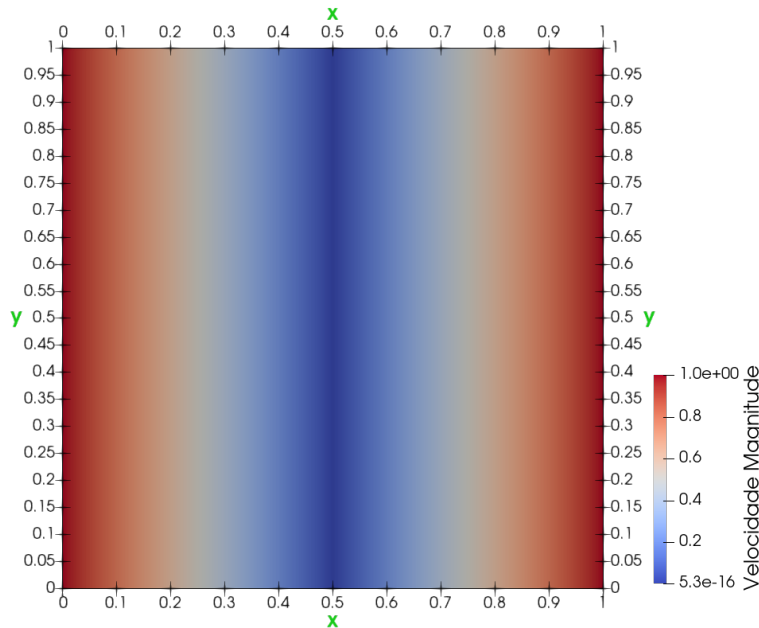


Figura 4.3: Distribuição da magnitude da velocidade.

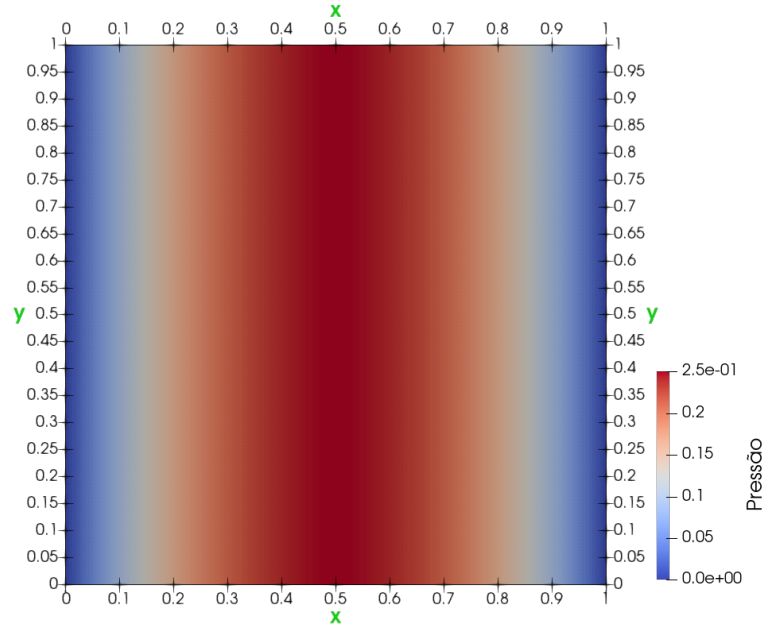


Figura 4.4: Campo de pressão.

Com o objetivo de mostrar a acurácia da ferramenta FEniCSx para problemas mistos, a seguir, serão apresentados os estudos de análise de convergência para ambas as variáveis, velocidade e pressão.

O erro para a pressão foi calculado como a diferença entre a solução analítica $p(x)$ e a solução aproximada p_h , ou seja, $e_p = p(x) - p_h$. O mesmo procedimento foi adotado para a velocidade, sendo $e_v = v(x) - v_h$. A norma do erro para a pressão foi calculada através da norma L^2 , conforme fórmula (2.1), e a norma do erro para a velocidade através da norma $H(\text{div}; \Omega)$ utilizando a fórmula (4.2). As ordens de convergência para a velocidade (q_v) e para a pressão (q_p) foram obtidas por meio da fórmula (3.4) e os resultados aproximados com 2 casas decimais. A Tabela 4.1 apresenta os resultados de 5 refinamentos sucessivos, iniciando com uma malha de 50 triângulos e quadruplicando sua quantidade à cada refinamento.

Tabela 4.1: Convergência do problema misto.

n	h_{min}	$\ e_p\ _{L^2}$	$\ e_v\ _{H(\text{div}; \Omega)}$	q_p	q_v
50	0,20000	2,76e-02	6,93e-02	-	-
200	0,10000	1,37e-02	2,52e-02	1,02	1,46
800	0,05000	6,81e-03	9,01e-03	1,00	1,48
3200	0,02500	3,40e-03	3,21e-03	1,00	1,49
12800	0,01250	1,70e-03	1,14e-03	1,00	1,50
51200	0,00625	8,50e-04	4,03e-04	1,00	1,50

Os resultados mostram que a norma do erro diminuiu tanto para a pressão, quanto para a velocidade. A ordem de convergência para a pressão foi aproximadamente 1, o que era esperado pela estimativa (2.38) quando utiliza-se o espaço P_0 ; enquanto para a velocidade foi cerca de 1,5. Esses resultados sugerem que a solução está convergindo à medida que a malha é refinada.

Transporte Passivo da Temperatura em Meios Porosos

Diante dos estudos apresentados até então, neste Capítulo já temos estrutura para resolver numericamente uma aplicação de transporte passivo da temperatura em meios porosos. Além disso, a implementação no FEniCSx demonstrou resultados precisos e alinhados com a teoria do método de elementos finitos. Em particular, as soluções exibiram convergência para a solução analítica em diversos cenários, incluindo problema de advecção-difusão e formulações mistas em meios porosos, que apresentam características similares ao problema tratado neste capítulo. Esses resultados reforçam a confiabilidade do código empregado para a resolução da aplicação proposta.

Primeiramente, o problema será definido em detalhes, incluindo as suas características, equações governantes e uma breve descrição da metodologia numérica. Em seguida, o Capítulo será finalizado com os resultados numéricos obtidos e suas principais considerações.

5.1 Descrição do problema

Este estudo investiga o comportamento térmico de um fluido incompressível em um meio poroso heterogêneo, como o encontrado em rochas porosas [7; 15; 18]. O problema envolve não apenas o escoamento do fluido através do meio poroso, mas também o transporte de calor nas duas fases do sistema (sólida e líquida). A fase sólida é representada pelas rochas, cujas propriedades térmicas, como densidade e capacidade calorífica, diferem das propriedades do fluido. O objetivo principal é determinar a distribuição da temperatura ($^{\circ}C$) ao longo do meio, levando em consideração as variações temporais e espaciais. Além disso, busca-se calcular a velocidade de Darcy (com componentes em m/s) e pressão do fluido (Pa), uma vez que essas variáveis desempenham um papel crucial no comportamento térmico e no escoamento pelo domínio.

Este problema é dado por duas equações, em que a fase sólida é dada por:

$$(1 - \varphi)\tilde{\rho}\tilde{c}_p\frac{\partial\tilde{T}}{\partial t} = (1 - \varphi)\nabla \cdot (\tilde{\kappa}\nabla\tilde{T}) - ah(\tilde{T} - T), \quad (5.1)$$

e a fase do fluido por:

$$\varphi\rho c_p\frac{\partial T}{\partial t} + \rho c_p\nabla T \cdot \mathbf{v} = \varphi\nabla \cdot (\kappa\nabla T) + ah(\tilde{T} - T), \quad (5.2)$$

onde todas as variáveis com til estão associadas à rocha e sem til para as variáveis relacionadas ao fluido. ρ representa a densidade; c_p a capacidade de calor; T a temperatura; t o tempo; κ a condutividade termal; $\mathbf{v}$ o fluxo de volume chamado de velocidade Darcy; a é a área de superfície específica; h é o coeficiente de transferência de calor e φ a porosidade efetiva que é dada por:

$$\varphi = \frac{V_p}{V_p + V_m},$$

onde V_p representa o volume dos poros que contribuem para o escoamento do fluido e V_m o volume de poros isolados e de regiões sem saída.

5.2 Características do problema

O problema envolve duas equações (5.1) e (5.2) de forma acoplada. Essas equações levam em consideração o comportamento térmico de cada fase. No entanto, a solução de um sistema acoplado de equações de energia, com variáveis dependentes do tempo e do espaço, torna o problema consideravelmente mais complexo.

Para simplificar a modelagem e a resolução numérica, assume-se o equilíbrio térmico local, ou seja, que a temperatura é a mesma para ambas as fases (sólida e líquida), portanto: $T = \tilde{T} = T_m$. Essa simplificação é frequentemente adotada em problemas semelhantes, especialmente quando o objetivo principal é lidar com o transporte de calor em meios porosos, onde o problema envolve não apenas a condução térmica, mas também o acoplamento com a Lei de Darcy e a conservação de massa.

A equação de energia simplificada é obtida pela soma das equações 5.1 e 5.2, obtendo assim:

$$\rho_m c_{p,m} \frac{\partial T_m}{\partial t} + \rho c_p \mathbf{v} \cdot \nabla T_m = \nabla \cdot (\kappa_m \nabla T_m), \quad (5.3)$$

onde

$$\rho_m c_{p,m} = \varphi \rho c_p + (1 - \varphi) \tilde{\rho} \tilde{c}_p, \quad (5.4)$$

e

$$\kappa_m = \varphi \kappa + (1 - \varphi) \tilde{\kappa}. \quad (5.5)$$

A Lei de Darcy descreve o escoamento de fluido através de um meio poroso [1; 7] sendo descrita por:

$$\mathbf{v} = -\frac{K}{\mu} (\nabla p - \rho g \nabla z),$$

onde K é a permeabilidade absoluta do meio (m^2), μ a viscosidade do fluido ($Pa \cdot s$), p a pressão do fluido, g a magnitude da aceleração da gravidade (m/s^2) e z um valor fixo (m). No presente estudo, o efeito da gravidade sobre o fluxo foi desconsiderado, simplificando a Lei de Darcy:

$$\mathbf{v} = -\frac{K}{\mu} \nabla p. \quad (5.6)$$

Outra equação fundamental para este problema está relacionada à conservação de massa, que pode ser expressa da seguinte forma:

$$\frac{d}{dt} \int_{\Omega} \varphi \rho \, d\mathbf{x} + \int_{\Gamma} \rho \mathbf{v} \cdot \check{\mathbf{n}} \, d\Gamma = \int_{\Omega} Q \, d\mathbf{x},$$

onde Q representa o termo fonte, Ω um volume elementar representativo e Γ a sua fronteira. Aplicando o teorema da divergência temos:

$$\int_{\Omega} \left(\frac{d}{dt}(\varphi \rho) + \nabla \cdot (\rho \mathbf{v}) \right) d\mathbf{x} = \int_{\Omega} Q \, d\mathbf{x}. \quad (5.7)$$

Como esta Equação (5.7) vale para qualquer volume elementar Ω , obtém-se:

$$\frac{d}{dt}(\varphi \rho) + \nabla \cdot (\rho \mathbf{v}) = Q.$$

O problema envolve um fluido incompressível, assumindo a porosidade independente do tempo e considerando que o problema não envolve termo fonte, a equação se reduz à:

$$\nabla \cdot \mathbf{v} = 0. \quad (5.8)$$

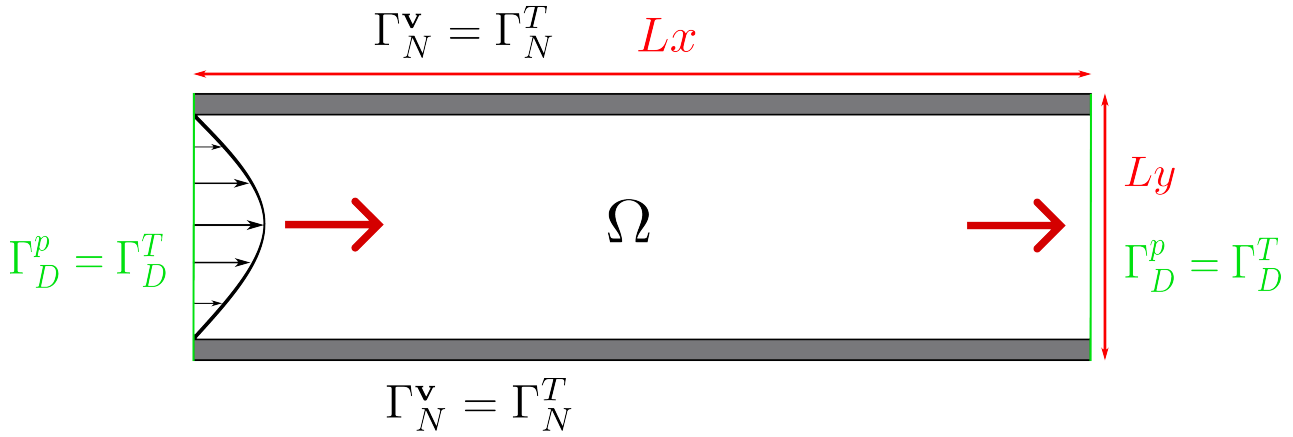
Portanto, ao combinar as equações (5.3), (5.6) e (5.8), e ao incorporar as condições de contorno e iniciais, o problema, que envolve as equações (5.9) e (5.10) de forma acoplada, pode ser formulado da seguinte maneira:

$$\left\{ \begin{array}{ll} \mathbf{v} + \frac{K}{\mu} \nabla p = 0, & \text{em } \Omega, \\ \nabla \cdot \mathbf{v} = 0, & \text{em } \Omega, \\ p = g, & \text{em } \Gamma_D^p, \\ \mathbf{v} \cdot \check{\mathbf{n}} = h, & \text{em } \Gamma_N^p, \end{array} \right. \quad (5.9)$$

$$\left\{ \begin{array}{ll} \rho_m c_{p,m} \frac{\partial T_m}{\partial t} + \rho c_p \mathbf{v} \cdot \nabla T_m - \nabla \cdot (\kappa_m \nabla T_m) = 0 & \text{em } \Omega \times (0, t_{\text{final}}], \\ T_m = T_D & \text{em } \Gamma_D^T, \\ -\kappa_m \nabla T_m \cdot \check{\mathbf{n}} = q_N & \text{em } \Gamma_N^T, \\ T_m(x, y, 0) = T_0(x, y) & \text{em } \Omega. \end{array} \right. \quad (5.10)$$

A Figura 5.1 ilustra a configuração da aplicação, onde as paredes estão representadas em cinza e a entrada e saída do fluido em verde. O domínio $\Omega = [0, Lx] \times [0, Ly]$, em que $Lx = 110m$ e $Ly = 30m$ e o tempo (em segundos) variando no intervalo $[0, 1000]$. $T_m : \Omega \rightarrow \mathbb{R}$ representa a temperatura média, $\mathbf{v} : \Omega \rightarrow \mathbb{R}^2$ a velocidade de Darcy e $p : \Omega \rightarrow \mathbb{R}$ a pressão. As fronteiras são descritas abaixo:

$$\begin{aligned} \Gamma_D^p &= \Gamma_D^T = \{(0, y) \text{ e } (Lx, y) : y \in [0, Ly]\}, \\ \Gamma_N^p &= \Gamma_N^T = \{(x, 0) \text{ e } (x, Ly) : x \in [0, Lx]\}. \end{aligned}$$

Figura 5.1: Representação do domínio Ω .

A região de estudo é delimitada pela entrada do fluido em $(0, y)$ e saída em (Lx, y) , e suas laterais são compostas por paredes com isolamento térmico, não havendo troca de calor com as laterais, ou seja, $q_N(x, 0) = q_N(x, Ly) = 0 \text{ W/m}^2$ e nem movimento do fluido na direção das paredes $h(x, 0) = h(x, Ly) = 0 \text{ m/s}$. A pressão e a temperatura do fluido são controladas na entrada $g(0, y) = 100 \text{ Pa}$, $T_D(0, y) = 80^\circ\text{C}$ e na saída $g(Lx, y) = 0 \text{ Pa}$, $T_D(Lx, y) = 20^\circ\text{C}$. A temperatura inicial em Ω é $T_0(x, y) = 20^\circ\text{C}$. A Figura 5.2 ilustra a permeabilidade K que é isotrópica e constante por partes. Os dados para sua representação foram obtidas pela camada 36 do projeto SPE10 (disponível em [3]). Esta camada apresenta uma variação abrupta e descontínua na distribuição dos valores da permeabilidade, destacando-se a existência de um canal de alta permeabilidade [1].

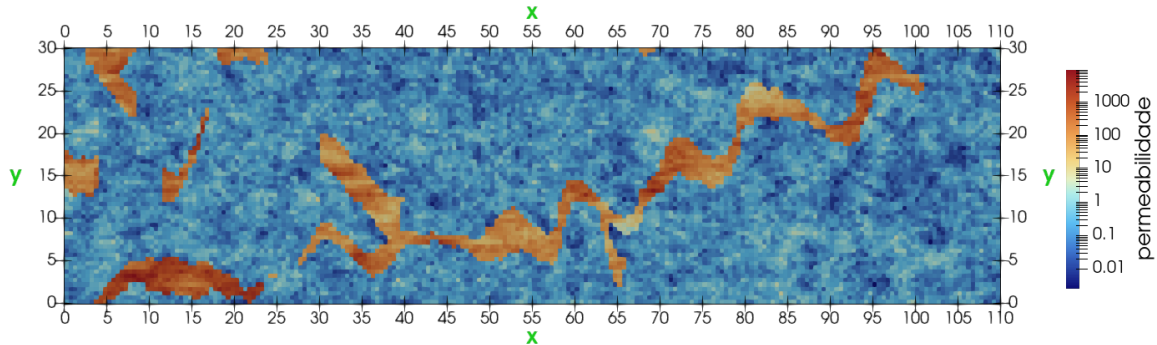


Figura 5.2: Campo de permeabilidade da camada 36 do projeto SPE 10 (disponível em [3]).

As demais características do fluido e da rocha, utilizadas na simulação, são apresentadas a seguir:

$$\begin{aligned}
 \rho &= 1,00 \text{ kg/m}^3 \\
 \tilde{\rho} &= 2,80 \text{ kg/m}^3 \\
 c_p &= 4,50 \text{ J/(kg} \cdot ^\circ\text{C)} \\
 \tilde{c}_p &= 1,00 \text{ J/(kg} \cdot ^\circ\text{C)} \\
 \kappa &= 0,58 \text{ W/(m} \cdot ^\circ\text{C)} \\
 \tilde{\kappa} &= 1,00 \text{ W/(m} \cdot ^\circ\text{C)} \\
 \mu &= 1,00 \text{ Pa} \cdot \text{s} \\
 \varphi &= 0,30
 \end{aligned}$$

Os dados para o fluido (água) e o sólido (rocha porosa) foram extraídos de [7]. Para simplificar a modelagem e facilitar a análise comparativa, a densidade do fluido foi normalizada para 1, enquanto a densidade da rocha foi ajustada para manter a proporção real em relação ao fluido ($\tilde{\rho} = 2,8 \cdot \rho$). De maneira análoga, as capacidades caloríficas foram ajustadas, preservando a relação física entre os materiais. Essa abordagem facilita os cálculos e permite analisar o comportamento térmico do escoamento, mantendo a consistência relativa entre as propriedades do fluido e da rocha.

5.3 Metodologia de resolução do problema

O objetivo do problema (5.9)-(5.10) é determinar as soluções aproximadas para a velocidade $\mathbf{v}$, a pressão p e a temperatura média T_m . Dada a natureza do problema, a Lei de Darcy e a equação da conservação de massa apresentadas em (5.9) indicam que as variáveis associadas ao escoamento do fluido, $\mathbf{v}$ e p , estão no estado estacionário, ou seja, são invariantes no tempo. Diante deste fato, essas variáveis são calculadas uma única vez, permitindo que o problema seja resolvido de forma desacoplada: inicialmente, resolve-se o sistema estacionário (5.9) e, em seguida, utiliza-se a solução obtida para calcular a temperatura T_m por meio do sistema transiente (5.10).

O processo de resolução inicia-se pelo problema misto (5.9), no qual a velocidade $\mathbf{v}$ e a pressão p são calculadas simultaneamente. A solução segue a formulação variacional descrita na seção 4.4, assim como o método de aproximação apresentado na seção 4.5 (usando os espaços RT_0 para a velocidade e P_0 para a pressão descontínua entre os elementos). Desta forma, obtém-se as soluções aproximadas para $\mathbf{v}_h$ e p_h .

A seguir, procede-se à resolução da equação de advecção-difusão (5.10). Inicialmente, calculam-se os valores médios da condutividade termal κ_m e da relação da densidade com a capacidade de calor $\rho_m c_{p,m}$, utilizando as expressões (5.5) e (5.4), respectivamente. Estes valores são inseridos no modelo (5.10). Como $\mathbf{v}_h$ já é uma variável conhecida, o problema é reduzido a uma equação transiente de advecção-difusão, resolvida com base na formulação variacional e nos métodos de discretização descritos na seção 2.14.3, com solução aproximada no espaço P_1 contínua entre os elementos.

A aproximação temporal é realizada utilizando o método de Euler implícito. O objetivo é calcular o vetor solução $\mathbf{T}_{k+1} \in \mathbb{R}^{N_T \times 1}$ a cada passo no tempo, utilizando a equação matricial:

$$(\mathbf{M} + \Delta t \mathbf{J} + \Delta t \mathbf{K}) \mathbf{T}_{k+1} = \mathbf{M} \mathbf{T}_k - \Delta t \mathbf{H}_{k+1}. \quad (5.11)$$

onde as matrizes e vetores $\mathbf{T}_k \in \mathbb{R}^{N_T \times 1}$, $\mathbf{M} \in \mathbb{R}^{N_T \times N_T}$, $\mathbf{J} \in \mathbb{R}^{N_T \times N_T}$, $\mathbf{K} \in \mathbb{R}^{N_T \times N_T}$, $\mathbf{H}_{k+1} \in \mathbb{R}^{N_T \times 1}$, possuem seus elementos definidos por:

$$\begin{aligned} M_{ij} &= \rho_m c_{p,m} \int_{\Omega} \phi_j(\mathbf{x}) \phi_i(\mathbf{x}) \, d\mathbf{x}, \\ J_{ij} &= \rho c_p \int_{\Omega} (\mathbf{v} \cdot \nabla \phi_j(x)) \phi_i(x) \, d\mathbf{x}, \\ K_{ij} &= \kappa_m \int_{\Omega} \nabla \phi_j(\mathbf{x}) \cdot \nabla \phi_i(\mathbf{x}) \, d\mathbf{x}, \\ H_i &= \int_{\Gamma_N} q_N \phi_i(\mathbf{x}) \, d\Gamma_N, \quad \mathbf{x} \in \Gamma_N. \end{aligned}$$

O processo iterativo prossegue até alcançar o tempo final desejado ($t = 1000s$), garantindo que a solução $T_h(x, y, t)$ seja calculada em toda a extensão do domínio e ao longo

do intervalo de tempo especificado. O código completo da implementação pode ser visto no Apêndice A.12.

5.4 Resultados numéricos e discussões

O domínio Ω foi discretizado em uma malha uniforme composta por 220×60 elementos triangulares, nas direções x e y , respectivamente. E o passo temporal foi definido como $\Delta t = 0,4576s$. As Figuras 5.3 e 5.4 apresentam, respectivamente, o campo de pressão e a magnitude da velocidade (na escala logarítmica), obtidos pela aproximação do problema (5.9)-(5.10).

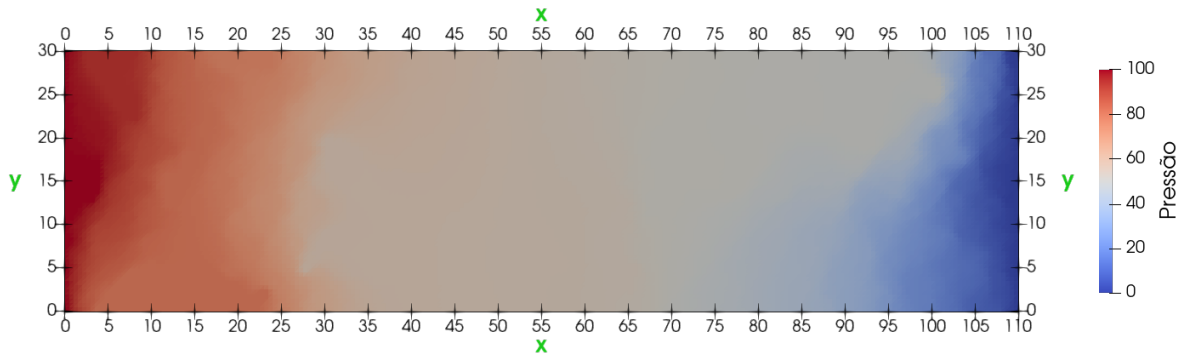


Figura 5.3: Campo da pressão.

Apesar da heterogeneidade do meio e dos altos contrastes na permeabilidade, a pressão apresentou uma distribuição relativamente suave. Isso sugere que as condições de contorno desempenham o papel mais significativo, estabelecendo um gradiente bem definido entre a entrada ($p(0, y) = 100Pa$) e a saída ($p(Lx, y) = 0Pa$).

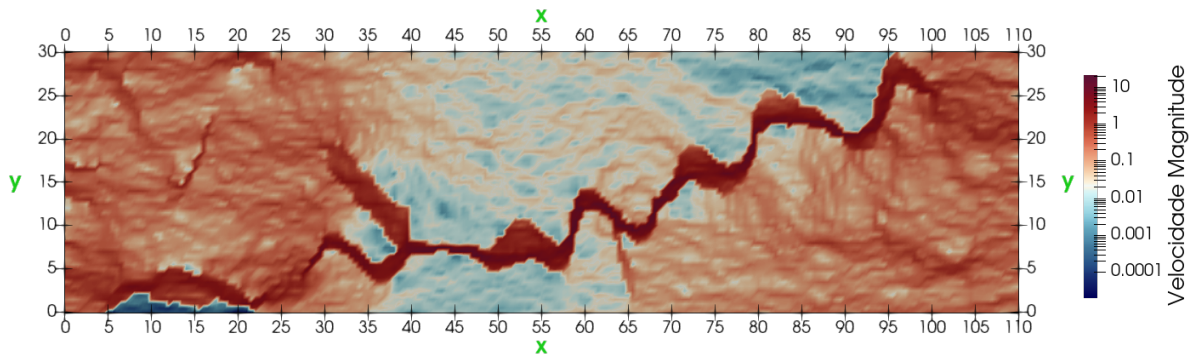


Figura 5.4: Distribuição da magnitude da velocidade em escala logarítmica.

A heterogeneidade na permeabilidade exerce um impacto significativo na magnitude da velocidade do escoamento. A distribuição dessa magnitude permite identificar as regiões preferenciais de percurso do fluido, evidenciando que ele tende a fluir majoritariamente pelas áreas de maior permeabilidade, com um fluxo mais intenso através do canal de alta permeabilidade presente no domínio.

Com o objetivo de compreender a evolução térmica no meio heterogêneo, os resultados para a temperatura foram analisados em quatro instantes diferentes no tempo. Essa abordagem permite avaliar como a temperatura se distribui e varia ao longo do domínio Ω , considerando as condições impostas no problema e as características heterogêneas do meio. A sequência de Figuras 5.5, 5.6, 5.7 e 5.8 destaca a evolução térmica, evidenciando as influências do escoamento e das propriedades do meio no transporte do calor.

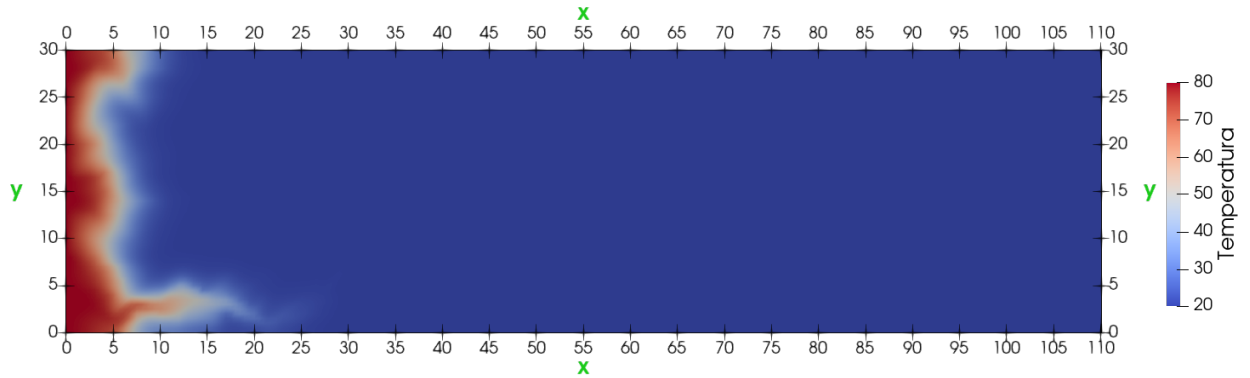


Figura 5.5: Aproximação da temperatura no tempo $t = 11,8976$ s.

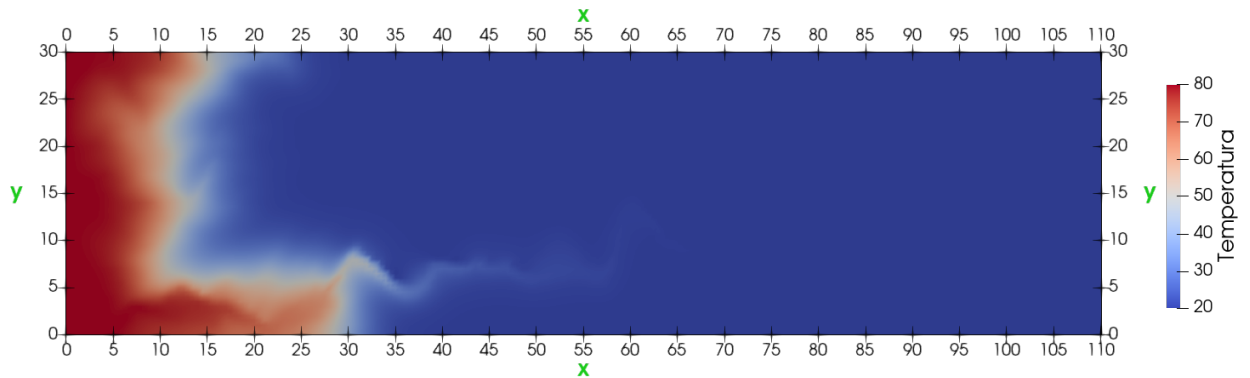


Figura 5.6: Aproximação da temperatura no tempo $t = 34,7776$ s.

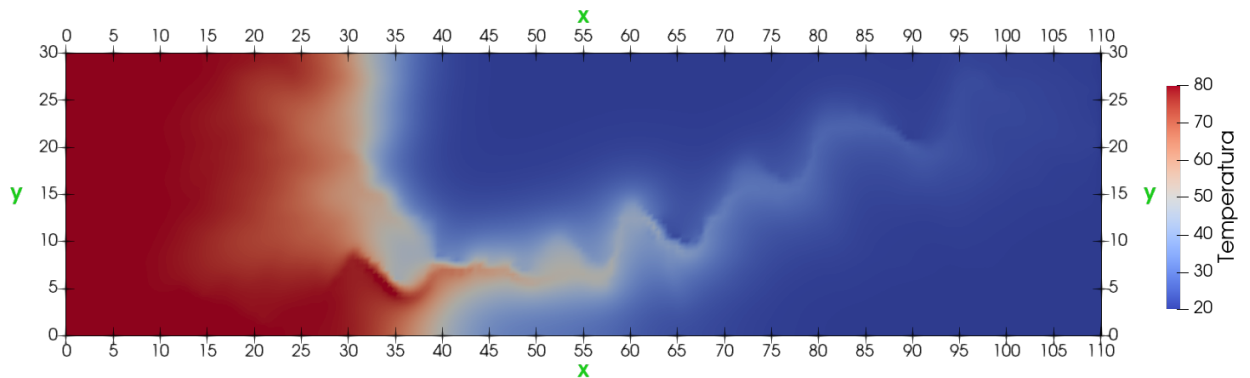


Figura 5.7: Aproximação da temperatura no tempo $t = 103,4176$ s.

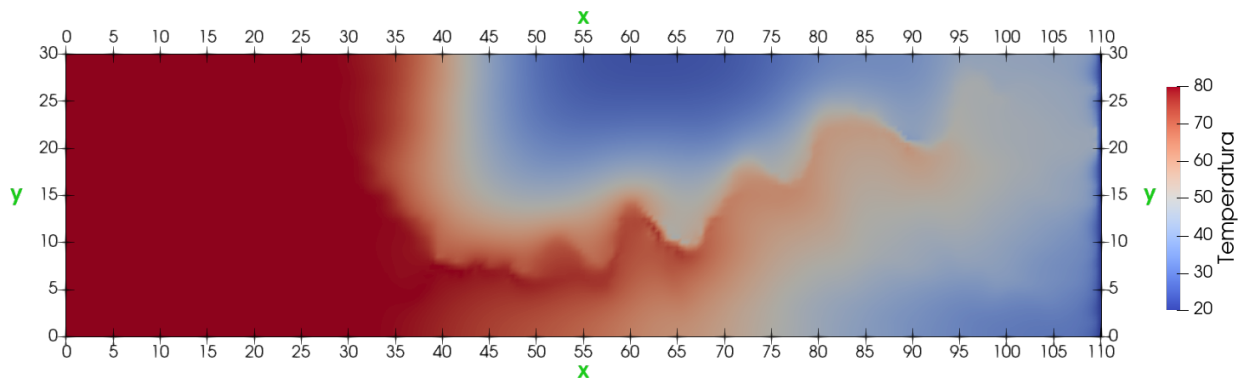


Figura 5.8: Aproximação da temperatura no tempo $t = 252,1376$ s.

Os resultados apresentam o aumento da temperatura de forma progressiva da entrada do fluido, onde $T_D(0, y) = 80^\circ C$, em direção à saída, onde $T_D(Lx, y) = 20^\circ C$. Ao longo desse percurso, o aquecimento é mais acentuado no canal de alta permeabilidade, sendo uma região que possui velocidade de escoamento elevada. Esse comportamento pode ser atribuído à maior advecção térmica nas regiões de alta velocidade, já que o transporte de calor é diretamente influenciado pela movimentação do fluido.

Os resultados obtidos estão em concordância com a teoria física do problema, refletindo adequadamente os efeitos da heterogeneidade do meio na distribuição da pressão, na magnitude da velocidade e no transporte de calor. A suavidade do campo de pressão, mesmo em um meio com altos contrastes de permeabilidade, confirma a dominância das condições de contorno na determinação do gradiente de pressão. Além disso, a concentração do escoamento nas regiões de maior permeabilidade e a influência da advecção térmica na distribuição da temperatura corroboram o comportamento esperado para problemas de advecção-difusão em meios porosos. Dessa forma, as soluções numéricas obtidas estão condizentes com a metodologia empregada e a precisão da implementação computacional utilizada.

Conclusão e Estudos Futuros

Os principais conceitos do método de elementos finitos foram abordados no Capítulo 2, incluindo um exemplo unidimensional detalhado que ilustra o processo de obtenção da solução aproximada. Foram discutidos tanto problemas no regime estacionário quanto transiente, explorando as abordagens de discretização temporal via os métodos de Euler explícito e implícito.

No Capítulo 3, o foco foi a aplicação prática da ferramenta FEniCSx para a resolução de problemas clássicos em diferentes dimensões. As soluções numéricas foram apresentadas por meio de gráficos e testes de convergência, demonstrando a flexibilidade e eficiência da ferramenta para problemas unidimensionais, bidimensionais e tridimensionais. Além disso, foi realizada a implementação de um problema misto utilizando elementos Raviart-Thomas RT_k aproveitando o suporte nativo do FEniCSx para esse tipo de elemento. As implementações foram disponibilizadas no Apêndice A, facilitando a reprodutibilidade e futuras extensões.

No Capítulo 4 apresentou-se conceitos do método de elementos finitos aplicados a problemas mistos, com destaque para os espaços $H(\text{div}; \Omega)$ e $RT_k(\Omega_e)$. A formulação mista permite a obtenção de soluções mais precisas para variáveis vetoriais, como a velocidade do fluido. Nesse contexto, a solução numérica de um problema misto foi analisada utilizando os elementos Raviart-Thomas, explorando as vantagens desse tipo de formulação na captura do comportamento do fluxo em meios heterogêneos.

O último Capítulo deste trabalho reuniu todos os conhecimentos teóricos e numéricos estudados isoladamente nos Capítulos anteriores, por meio de uma aplicação envolvendo o transporte passivo da temperatura em meios porosos. Esta aplicação foi modelada por um problema misto, para obtenção das variáveis velocidade-pressão, e um problema de advecção-difusão, para observar a evolução da temperatura média neste meio sendo influenciada pela solução do problema misto inicial. Os resultados numéricos evidenciam o impacto significativo da heterogeneidade do meio no comportamento do escoamento e na distribuição térmica. A pressão apresentou uma variação suave ao longo do domínio, enquanto a velocidade do fluido se mostrou mais elevada nas regiões de alta permeabilidade, revelando os canais preferenciais do fluxo. A distribuição da temperatura demonstrou um aquecimento mais rápido nas regiões de maior permeabilidade, comportamento atribuído à maior velocidade do fluido nessas regiões, que intensificam o transporte advectivo de calor. Diante das diversas características físicas envolvidas, como a heterogeneidade do meio, as condições de contorno impostas e a interação entre as variáveis pressão, velocidade e temperatura, este problema apresenta uma complexidade significativa. O estudo detalhado da influência das propriedades físicas é, portanto, essencial para compreender mais profundamente os fenômenos associados, permitindo não apenas interpretar os

resultados obtidos, mas também ampliar o entendimento sobre problemas similares em contextos reais.

Estudos futuros

Este trabalho pode ser ampliado em diversas direções. Algumas possibilidades incluem:

- **Estudo da recuperação terciária do Petróleo** Investigar estratégias numéricas para modelar e simular técnicas de *recuperação terciária* [1] em meios porosos, como a injeção de solventes (CO_2 , N_2 ou gás natural), surfactantes ou polímeros. O objetivo seria analisar como diferentes abordagens modificam a mobilidade do fluido e o impacto na eficiência da extração. Além disso, a modelagem de métodos térmicos, como a injeção de vapor poderia ser explorada para compreender sua influência na redução da viscosidade do óleo e na otimização do processo de extração.
- **Paralelização e eficiência computacional:** Explorar estratégias de paralelização para melhorar o desempenho computacional da simulação, aproveitando os recursos do FEniCSx para execução em arquiteturas paralelas. Isso permitiria a resolução de problemas mais robustos e com maior refinamento espacial e temporal.
- **Implementação do método MRCM no FEniCSx:** Explorar a aplicabilidade do Multiscale Robin Coupled Method (MRCM) [31] dentro da estrutura FEniCSx, avaliando sua eficiência na resolução problemas multiescala, como fluxo em meios altamente heterogêneos.

Referências

- [1] SOUSA, F. S.; ROCHA, F. F. **Métodos de volumes finitos para modelagem computacional de reservatórios de petróleo**. São Carlos, SP: SBMAC, 2023. E-book. (Notas em Matemática Aplicada, v. 96). Disponível em: <<https://www.sbmec.org.br/wp-content/uploads/2023/07/volume96.pdf>>. Acesso em: 27 out. 2024.
- [2] LARSON, M. G.; BENGTSON, F. **The finite element method: theory, implementation, and applications**. Heidelberg: Springer Science & Business Media, 2013. (Texts in Computational Science and Engineering, v. 10).
- [3] SPE International. **SPE comparative solution project**. Disponível em: <<https://www.spe.org/web/csp/datasets/set02.htm>>. Acesso em: 15 jan. 2025.
- [4] CUMINATO, J. A.; MENEGUETTE JUNIOR, M. **Discretização de equações diferenciais parciais: técnicas de diferenças finitas**. Rio de Janeiro: SBM, 2013.
- [5] BECKER E. B.; CAREY G. F.; ODEN, J. T. **Finite elements: an introduction**. v.1. Englewood Cliffs: Prentice Hall, 1981.
- [6] REDDY, J. N.; GARTLING, D. K. **The finite element method in heat transfer and fluid dynamics**. 3. ed. Boca Raton: CRC press, 2010.
- [7] LAMPE, V. **Modelling fluid flow and heat transport in fractured porous media**. 2013. Dissertação (Mestrado) — The University of Bergen, 2013. Disponível em: <<https://hdl.handle.net/1956/7509>>. Acesso em: 27 out. 2024.
- [8] FENICS PROJECT. **Legacy FEniCS**. Disponível em: <<https://fenicsproject.org/citing/legacy/>>. Acesso em: 7 nov. 2024.
- [9] HAM, D. A. et al. **Firedrake user manual**. 1. ed. [S.l.], 2023. Disponível em: <<https://doi.org/10.25561/104839>>. Acesso em: 28 out. 2024.
- [10] ALNÆS, M. S. et al. Unified form language: a domain-specific language for weak formulations of partial differential equations. **ACM Transactions on Mathematical Software**, Association for Computing Machinery, New York, NY, v. 40, n. 2, p. 1–37, 2014. Disponível em: <<https://doi.org/10.1145/2566630>>. Acesso em: 27 out. 2024.
- [11] SCROGGS, M. W. et al. Construction of arbitrary order finite element degree-of-freedom maps on polygonal and polyhedral cell meshes. **ACM Transactions on Mathematical Software**, Association for Computing Machinery, New York, v. 48, n. 2, p. 18:1–18:23, 2022. Disponível em: <<https://doi.org/10.1145/3524456>>. Acesso em: 27 out. 2024.

- [12] SCROGGS, M. W. et al. **Basix: a runtime finite element basis evaluation library.** *The Journal of Open Source Software*, The Open Journal, v. 7, n. 73, p. 3982, 2022. Disponível em: <<https://doi.org/10.21105/joss.03982>>. Acesso em: 27 out. 2024.
- [13] BARATTA, I. A. et al. **DOLFINx: the next generation FEniCS problem solving environment.** Zenodo, 2023. Disponível em: <<https://doi.org/10.5281/zenodo.10447666>>. Acesso em: 27 out. 2024.
- [14] LANGTANGEN, H. P.; LOGG, A. **Solving PDEs in python: the fenics tutorial I.** Cham, Switzerland: Springer Nature, 2016. E-book. (Simula SpringerBriefs on Computing, v. 3). Disponível em: <<https://link.springer.com/book/10.1007/978-3-319-52462-7>>. Acesso em: 27 out. 2024.
- [15] BARATTA, I. A.; AUSAS, R. F. **Mixed problem: Darcy's flow in porous media.** [S.l.], 2023. Este repositório contém os scripts python e os notebooks jupyter a serem utilizados no curso FEniCSx a ser realizado no ICMC-USP, São Carlos, SP de 30/01 a 03/02/2023. Disponível em: <https://github.com/IgorBaratta/FEniCSxCourse/blob/main/Problem6_Darcy/Darcy.ipynb>. Acesso em: 15 jan. 2025.
- [16] DOKKEN, J. S. **The FEniCSx tutorial.** [S.l.], 2023. Esta página da web é uma adaptação do tutorial FEniCS. Disponível em: <<https://jsdokken.com/dolfinx-tutorial/index.html>>. Acesso em: 26 jan. 2025.
- [17] CARVALHO, M. S.; VALÉRIO, J. V. **Introdução ao método de elementos finitos: aplicação em dinâmica dos fluidos.** São Carlos: SBMAC, 2012. E-book. (Notas em Matemática Aplicada, v. 61). Disponível em: <https://www.sbmac.org.br/wp-content/uploads/2022/08/livro_61.pdf>. Acesso em: 27 out. 2024.
- [18] AUSAS, R. F. **(Lecture Notes), Introduction to the finite element method: theory and applications with the fenics(x) platform.** São Carlos, SP: [s.n.], 2023.
- [19] LIMA, E. L. **Curso de análise.** v.2. Rio de Janeiro, RJ: IMPA, 1981.
- [20] BRENNER, S. C.; SCOTT, L. R. **The mathematical theory of finite element methods.** 3. ed. New York: Springer, 2008. (Texts in Applied Mathematics, v. 15).
- [21] REDDY, B. D. **Introductory functional analysis: with applications to boundary value problems and finite elements.** New York: Springer Science & Business Media, 1998. (Texts in Applied Mathematics, v. 27).
- [22] CARVALHO, P. G. S. **Métodos de elementos finitos hibridizados aplicados a problemas hidromecânicos= Hybridized finite element methods applied to hydro-mechanical problems.** 2021. Tese (Doutorado) — Unicamp, Campinas, 2021.
- [23] GOCKENBACH, M. S. **Understanding and implementing the finite element method.** Philadelphia: SIAM, 2006.
- [24] JOHNSON, C. **Numerical solution of partial differential equations by the finite element method.** Mineola, N.Y.: Dover Publications, INC, 2009.
- [25] CUNHA, M. C. C. **Métodos numéricos.** 2. ed. Campinas, SP: Editora da Unicamp, 2003.

- [26] BOFFI, D.; BREZZI, F.; FORTIN, M. **Mixed finite element methods and applications**. Heidelberg: Springer, 2013. (Springer series in computational mathematics, v. 44).
- [27] BREZZI, F.; FORTIN, M. **Mixed and hybrid finite element methods**. New York: Springer-Verlag, 1991. (Springer series in computational mathematics, v. 15).
- [28] RAVIART, P. A.; THOMAS, J. M. A mixed finite element method for 2-nd order elliptic problems. In: GALLIGANI, I.; MAGENES, E. (Ed.). **Mathematical Aspects of Finite Element Methods**. Berlin, Heidelberg: Springer Berlin Heidelberg, 1977. p. 292–315.
- [29] DONEA, J.; HUERTA, A. **Finite element methods for flow problems**. Chichester: John Wiley & Sons, 2003.
- [30] GIRAULT, V.; RAVIART, P. A. **Finite element methods for Navier-Stokes equations: theory and algorithms**. Berlin, Heidelberg: Springer Berlin Heidelberg, 1986. (Springer series in computational mathematics, v.5).
- [31] GUIRALDELLO, R. et al. Velocity postprocessing schemes for multiscale mixed methods applied to contaminant transport in subsurface flows. **Computational Geosciences**, v. 24, p. 1141–1161, 2020. Disponível em: <<https://doi.org/10.1007/s10596-019-09930-8>>. Acesso em: 01 fev. 2025.

Códigos gerados em FEniCSx

A.1 Problema 2D da subseção 3.1.2.1

```

1 def solucao_fem_2D(N, degree, Lx, Ly):
2     msh = mesh.create_rectangle(comm=mpi4py.MPI.COMM_WORLD, points=((0.0,0.0)
3         , (Lx, Ly)), n=(N,N), cell_type = mesh.CellType.triangle) #malha
4
5     #----- ESPAÇO DE FUNÇÕES CONTINUOUS GALERKING -----
6     V = fem.functionspace(msh, ("CG", degree))
7
8     x = SpatialCoordinate(msh)
9     u_aux = fem.Function(V)
10    u_aux.interpolate(lambda x: np.sin(2*np.pi*x[0]) * np.sin(2*np.pi*x[1]))
11
12    #----- CONDIÇÕES DE CONTORNO -----
13    dofsL = fem.locate_dofs_geometrical(V, lambda x: np.isclose(x[0], 0))
14    dofsR = fem.locate_dofs_geometrical(V, lambda x: np.isclose(x[0], Lx))
15    dofsD = fem.locate_dofs_geometrical(V, lambda x: np.isclose(x[1], 0))
16    dofsU = fem.locate_dofs_geometrical(V, lambda x: np.isclose(x[1], Ly))
17
18    x = SpatialCoordinate(msh)
19    bcs = [fem.dirichletbc(u_aux, dofsL), fem.dirichletbc(u_aux, dofsR), fem.
20        dirichletbc(u_aux, dofsD), fem.dirichletbc(u_aux, dofsU)]
21
22    #----- SOLUCAO ANALÍTICA USANDO LAMBDA -----
23    uex = lambda x: ufl.sin(2*ufl.pi*x[0])*ufl.sin(2*ufl.pi*x[1])
24    mu = lambda x: ufl.cos(2.0*ufl.pi*x[0])*ufl.cos(2.0*ufl.pi*x[1])+1.0
25
26    #----- FORMULAÇÃO VARIACIONAL -----
27
28    #----- FUNÇÕES TRIAL e TEST -----
29    u = ufl.TrialFunction(V)
30    v = ufl.TestFunction(V)
31
32    #----- FORMA BILINEAR -----
33    dx = ufl.dx
34    a = ufl.inner(mu(x)*ufl.grad(u), ufl.grad(v)) * dx
35
36    #----- FORMA LINEAR -----
37    source = fem.Function(V)
38    source = -ufl.div(mu(x)*ufl.grad(uex(x)))
39    L = (source * v) * dx

```

```

39
40 #----- SOLUCIONANDO O PROBLEMA -----
41 problem = fem.petsc.LinearProblem(a, L, bcs=bcs, petsc_options={"
    ksp_type": "preonly", "pc_type": "lu"})
42 uh = problem.solve()
43
44 #----- CALCULANDO NORMAS DO ERRO -----
45 eh = uh - uex(x)
46
47 #NORMA L2
48 eL2form = fem.form(eh**2*dx)
49 L2norm = np.sqrt(fem.assemble_scalar(eL2form))
50
51 #NORMA H1
52 eH1form = fem.form(inner(grad(eh), grad(eh))*dx)
53 H1norm = np.sqrt(fem.assemble_scalar(eL2form) + fem.assemble_scalar(
    eH1form))
54
55 #----- Calculo do valor de h (diâmetro do elemento) -----
56 tdim = msh.topology.dim #dimensão topológica
57 num_cells = msh.topology.index_map(tdim).size_local + msh.topology.
    index_map(tdim).num_ghosts #número de elementos
58 msh.topology.create_connectivity(tdim-1, tdim)
59 h_e = msh.h(tdim-1, np.arange(num_cells))
60 h_dim = max(h_e)
61 return uh, eh, L2norm, H1norm, h, h_dim

```

A.2 Problema 2D da subseção 3.1.2.2

```

1 def solucao_fem_2D(N, degree):
2     #----- DIMENSÃO DO DOMÍNIO -----
3     Lx = 1
4     Ly = 1
5
6     msh = mesh.create_rectangle(comm=mpi4py.MPI.COMM_WORLD, points=((0,0), (
7         Lx, Ly)), n=(N,N), cell_type = mesh.CellType.triangle)
8     V = fem.functionspace(msh, ("CG", degree))
9     f = fem.Function(V)
10
11     #----- CONDIÇÕES DE CONTORNO -----
12     uleft = 100
13     uright = 1
14
15     dofsL = fem.locate_dofs_geometrical(V, lambda x: np.isclose(x[0], 0))
16     dofsR = fem.locate_dofs_geometrical(V, lambda x: np.isclose(x[0], Lx))
17
18     bcs = [fem.dirichletbc(ScalarType(uleft), dofsL, V), fem.dirichletbc(
19         ScalarType(uright), dofsR, V)]
20
21     #----- FORMULAÇÃO VARIACIONAL -----
22     #----- FUNÇÕES TRIAL e TEST -----
23     u = ufl.TrialFunction(V)
24     v = ufl.TestFunction(V)
25
26     #----- FORMA BILINEAR -----
27     dx = ufl.dx
28     mu = fem.Constant(msh, 1.0)

```

```

28 a = ufl.inner(mu*ufl.grad(u), ufl.grad(v)) * dx
29
30 #----- FORMA LINEAR -----
31 f = 1.0
32 source = fem.Constant(msh, f)
33 g_N = fem.Constant(msh, 0.0)
34 L = (source * v) * dx - (g_N*v) * dx
35
36 #----- SOLUCIONANDO O PROBLEMA -----
37 problem = fem.petsc.LinearProblem(a, L, bcs=bcs, petsc_options={"
    ksp_type": "preonly", "pc_type": "lu"})
38 uh = problem.solve()
39
40 #----- SOLUÇÃO ANALÍTICA -----
41 uex_aux = lambda x: uleft + ((uright - uleft)/Lx + 0.5*f*Lx)*x[0] -
    0.5*f*x[0]**2
42 x = SpatialCoordinate(msh)
43
44 #----- PLOTAGEM -----
45 pv.OFF_SCREEN = True
46 u_topology, u_cell_types, u_geometry = dolfinx.plot.vtk_mesh(V)
47 u_grid = pv.UnstructuredGrid(u_topology, u_cell_types, u_geometry)
48 u_grid.point_data["u"] = uh.x.array.real
49 u_grid.set_active_scalars("u")
50
51 pl = pv.Plotter()
52 min_ = uh.x.array.min()
53 max_ = uh.x.array.max()
54 sargs = dict(height=0.6, vertical=True, position_x=0.85, position_y=0.15
55             , title_font_size=20,
56             label_font_size=16,
57             shadow=True,
58             italic=True,
59             fmt="%3f",
60             font_family="arial",)
61 pl.add_mesh(u_grid, show_edges=True, clim=[min_, max_], scalar_bar_args=
    sargs)
62 pl.view_xy()
63 labels = dict(zlabel='uh', xlabel='x', ylabel='y')
64 pl.show_grid(**labels)
65 pl.add_axes(**labels)
66 figure = pl.screenshot("sol_aprox_n%d.png"%N)
67 files.download("sol_aprox_n%d.png"%N)
68 pl.show()
69
70 #----- CALCULANDO NORMAS DO ERRO -----
71 eh = uh - uex_aux(x)
72
73 #NORMA L2
74 eL2form = fem.form(eh**2*dx)
75 L2norm = np.sqrt(fem.assemble_scalar(eL2form))
76
77 #NORMA H1
78 eH1form = fem.form(inner(grad(eh), grad(eh))*dx)
79 H1norm = np.sqrt(fem.assemble_scalar(eL2form) + fem.assemble_scalar(
    eH1form))
80
81 h = Lx/N
82 return uh, eh, L2norm, H1norm, h

```

A.3 Problema 2D com malha não estruturada da subseção 3.1.2.3

```

1 def solucao_fem_2D(N, degree, Lx, Ly):
2     # ----- CRIAÇÃO DA MALHA -----
3     cell_type = mesh.CellType.triangle
4     h_malha = Lx/N
5     simple_mesh = False
6
7     if (simple_mesh):
8         msh = mesh.create_rectangle(comm=MPI.COMM_WORLD,
9                                     points=((0.0, 0.0), (Lx, Ly)), n=(N, N),
10                                    cell_type=cell_type)
11     else:
12         def GenerateMesh():
13             gmsh.initialize()
14             proc = MPI.COMM_WORLD.rank
15
16             if proc == 0:
17                 background = gmsh.model.occ.addRectangle(0, 0, 0, Lx, Ly)
18                 gmsh.model.occ.synchronize()
19                 pl = gmsh.model.addPhysicalGroup(2, [background], 1)
20
21             gmsh.model.mesh.setSize(gmsh.model.getEntities(0), h_malha) #
22
23             if (cell_type == mesh.CellType.quadrilateral):
24                 gmsh.model.mesh.setRecombine(2, pl)
25
26             gmsh.model.mesh.generate(2)
27
28             msh, _, _ = io.gmshio.model_to_mesh(gmsh.model, comm=MPI.COMM_WORLD,
29                                                  rank=0, gdim=2)
30             gmsh.finalize()
31             return msh
32
33         msh = GenerateMesh()
34
35         #Criando o espaço das funções Continuous Galerkin
36         V = fem.functionspace(msh, ("CG", degree))
37
38         #criando uma função f que pertença ao espaço de funções V.
39         f = fem.Function(V)
40
41         x = SpatialCoordinate(msh)
42         u_aux = fem.Function(V)
43         u_aux.interpolate(lambda x: np.sin(2*np.pi*x[0]) * np.sin(2*np.pi*x[1]))
44
45         # ----- CONDIÇÃO DE CONTORNO -----
46         #Identificação dos graus de liberdade para as condições de contorno
47         dofsL = fem.locate_dofs_geometrical(V, lambda x: np.isclose(x[0], 0))
48         dofsR = fem.locate_dofs_geometrical(V, lambda x: np.isclose(x[0], Lx))
49         dofsD = fem.locate_dofs_geometrical(V, lambda x: np.isclose(x[1], 0))
50         dofsU = fem.locate_dofs_geometrical(V, lambda x: np.isclose(x[1], Ly))
51
52         #Criando uma lista 'bcs' especificando os valores e os graus de liberdade
53         #associados
54         bcs = [fem.dirichletbc(u_aux, dofsL), fem.dirichletbc(u_aux, dofsR), fem.
55               dirichletbc(u_aux, dofsD), fem.dirichletbc(u_aux, dofsU)]

```

```

53
54 #----- SOLUCAO ANALITICA USANDO LAMBDA -----
55 uex = lambda x: ufl.sin(2*ufl.pi*x[0])*ufl.sin(2*ufl.pi*x[1])
56 mu = lambda x: ufl.cos(2.0*ufl.pi*x[0])*ufl.cos(2.0*ufl.pi*x[1])+1.0
57
58 # Formulação variacional
59
60 # Funções Trial and test
61 u = ufl.TrialFunction(V)
62 v = ufl.TestFunction(V)
63
64 # Forma Bilinear
65 x = SpatialCoordinate(msh)
66 dx = ufl.dx
67 a = ufl.inner(mu(x)*ufl.grad(u), ufl.grad(v)) * dx
68
69 # Forma Linear
70 source = -ufl.div(mu(x)*ufl.grad(uex(x)))
71 L = (source * v) * dx
72
73 # Solucionando o problema
74 problem = fem.petsc.LinearProblem(a, L, bcs=bcs, petsc_options={"
    ksp_type": "preonly", "pc_type": "lu"})
75 uh = problem.solve()
76
77 #----- SALVANDO ARQUIVO -----
78 uh.name = "u" #nomeia a variável da solução uh
79 with io.XDMFFile(mpi4py.MPI.COMM_WORLD, "temp_malha_3_2_1_3.xdmf", "w")
    as file:
80     file.write_mesh(msh) #armazena os dados da malha
81     file.write_function(uh) #armazena os dados da solução
82 files.download('temp_malha_3_2_1_3.xdmf') #efetua o download do arquivo
    em xdmf
83 files.download('temp_malha_3_2_1_3.h5') #efetua o download do arquivo em
    x5
84
85
86 #----- PLOTAGEM -----
87 pv.OFF_SCREEN = True
88 u_topology, u_cell_types, u_geometry = dolfinx.plot.vtk_mesh(V)
89 u_grid = pv.UnstructuredGrid(u_topology, u_cell_types, u_geometry)
90 u_grid.point_data["u"] = uh.x.array.real #valores da solução nos vértices
    da malha
91 u_grid.set_active_scalars("u")
92
93 pl = pv.Plotter()
94 min_ = uh.x.array.min()
95 max_ = uh.x.array.max()
96 sargs = dict(height=0.6, vertical=True, position_x=0.85, position_y=0.15
97             ,title_font_size=20,
98             label_font_size=16,
99             shadow=True,
100             italic=True,
101             fmt="%0.3f", #3 casas decimais
102             font_family="arial",)
103 pl.add_mesh(u_grid, show_edges=True, clim=[min_, max_], scalar_bar_args=
    sargs)
104 pl.view_xy()
105 labels = dict(zlabel='uh', xlabel='x', ylabel='y')
106 pl.show_grid(**labels)

```

```

107 pl.add_axes(**labels)
108 figure = pl.screenshot("sol_aprox_n%d.png"%N)
109 files.download("sol_aprox_n%d.png"%N)
110
111 #----- Cálculo do valor de h (diâmetro do elemento) -----
112 tdim = msh.topology.dim #dimensão topológica
113 num_cells = msh.topology.index_map(tdim).size_local + msh.topology.
    index_map(tdim).num_ghosts #número de elementos
114 msh.topology.create_connectivity(tdim-1, tdim)
115 h_e = msh.h(tdim-1, np.arange(num_cells))
116 h_dim = max(h_e)
117
118 #CÁLCULO DO ERRO
119 eh = uh - uex(x)
120
121 #----- CALCULANDO NORMAS DO ERRO -----
122 #norma L2
123 eL2form = fem.form(eh**2*dx)
124 L2norm = np.sqrt(fem.assemble_scalar(eL2form))
125
126 #norma H1
127 eH1form = fem.form(inner(grad(eh), grad(eh))*dx)
128 H1norm = np.sqrt(fem.assemble_scalar(eL2form) + fem.assemble_scalar(
    eH1form))
129
130 h = Lx/N
131 return uh, eh, L2norm, H1norm, h, h_dim

```

A.4 Primeiro problema com domínio irregular da sub- seção 3.1.2.4.1

```

1 def solucao_fem_2D(N, degree, Lx, Ly):
2
3     #----- CRIAÇÃO DA MALHA -----
4     cell_type = mesh.CellType.triangle
5     h_malha = (2*Lx)/N
6     radius = Lx
7
8     def GenerateMesh():
9         gmsh.initialize()
10        proc = MPI.COMM_WORLD.rank
11
12        if proc == 0:
13            c_x, c_y = 0, 0 #coordenadas do centro do círculo
14            background = gmsh.model.occ.addDisk(c_x, c_y, 0, radius, radius)
15            gmsh.model.occ.synchronize()
16            pl = gmsh.model.addPhysicalGroup(2, [background], 1)
17
18            gmsh.model.mesh.setSize(gmsh.model.getEntities(0), h_malha)
19
20            if (cell_type == mesh.CellType.quadrilateral):
21                gmsh.model.mesh.setRecombine(2, pl)
22
23            gmsh.model.mesh.generate(2)
24

```

```

25     msh, _, _ = io.gmshio.model_to_mesh(gmsh.model, comm=MPI.COMM_WORLD,
26     rank=0, gdim=2)
27
28     gmsh.finalize()
29     return msh
30 msh = GenerateMesh()
31
32 V = fem.functionspace(msh, ("CG", degree))
33
34 f = fem.Function(V)
35
36 #----- CONDIÇÃO DE CONTORNO -----
37 u_circle = 0
38
39 #Identificação dos graus de liberdade para as condições de contorno
40 def boundary(x):
41     r = np.sqrt(x[0]**2 + x[1]**2)
42     return np.isclose(r, radius)
43
44 dofs_circle = fem.locate_dofs_geometrical(V, boundary)
45 bc_value = fem.Constant(msh, ScalarType(u_circle))
46
47 x = SpatialCoordinate(msh)
48 u_aux = fem.Function(V)
49 u_aux.interpolate(lambda x: -np.sin(x[0]**2)+np.sin(1-x[1]**2))
50
51 #Aplicando a condição de contorno de Dirichlet na fronteira do círculo
52 bcs = [fem.dirichletbc(u_aux, dofs_circle)]
53
54 # Formulação variacional
55
56 # Funções Trial and test
57 u = ufl.TrialFunction(V)
58 v = ufl.TestFunction(V)
59
60 ####----- SOLUÇÃO ANALÍTICA -----###
61 uex = lambda x: -ufl.sin(x[0]**2)+ufl.sin(1-x[1]**2)
62
63 # Forma Bilinear
64 dx = ufl.dx
65 mu = fem.Constant(msh, 1.0)
66 a = ufl.inner(mu*ufl.grad(u), ufl.grad(v)) * dx
67
68 # Forma Linear
69 source = fem.Function(V)
70 source = -ufl.div(mu*ufl.grad(uex(x)))
71 L = (source * v) * dx
72
73 # Soluciona o problema
74 problem = fem.petsc.LinearProblem(a, L, bcs=bcs, petsc_options={"
75     ksp_type": "preonly", "pc_type": "lu"})
76 uh = problem.solve()
77
78 #----- PLOTAGEM -----
79 pv.OFF_SCREEN = True
80 u_topology, u_cell_types, u_geometry = dolfinx.plot.vtk_mesh(V)
81 u_grid = pv.UnstructuredGrid(u_topology, u_cell_types, u_geometry)
82 u_grid.point_data["u"] = uh.x.array.real
83 u_grid.set_active_scalars("u")
84
85 pl = pv.Plotter()

```

```

83 min_ = uh.x.array.min()
84 max_ = uh.x.array.max()
85 sargs = dict(height=0.6, vertical=True, position_x=0.85, position_y=0.15,
86             title_font_size=20,
87             label_font_size=16,
88             shadow=True,
89             italic=True,
90             fmt="%3f",
91             font_family="arial",)
92 pl.add_mesh(u_grid, show_edges=True, clim=[min_, max_], scalar_bar_args=
93           sargs)
94 pl.view_xy()
95 labels = dict(zlabel='u', xlabel='x', ylabel='y')
96 pl.show_grid(**labels)
97 pl.add_axes(**labels)
98 figure = pl.screenshot("sol_aprox_n%d.png"%N)
99 files.download("sol_aprox_n%d.png"%N)
100
101 #----- SALVANDO ARQUIVO -----
102 uh.name = "u" #nomeia a variável da solução uh
103 with io.XDMFFile(mpi4py.MPI.COMM_WORLD, "temp_malha_3_1_2_4_1.xdmf", "w")
104   as file :
105     file.write_mesh(msh) #armazena os dados da malha
106     file.write_function(uh) #armazena os dados da solução
107 files.download('temp_malha_3_1_2_4_1.xdmf') #efetua o download do arquivo
108   em xdmf
109 files.download('temp_malha_3_1_2_4_1.h5') #efetua o download do arquivo
110   em x5
111
112 #----- Calculo do valor de h (diâmetro do elemento) -----
113 tdim = msh.topology.dim #dimensão topológica
114 num_cells = msh.topology.index_map(tdim).size_local + msh.topology.
115   index_map(tdim).num_ghosts #número de elementos
116 msh.topology.create_connectivity(tdim-1, tdim)
117 h_e = msh.h(tdim-1, np.arange(num_cells))
118 h_dim = max(h_e)
119
120 #CÁLCULO DO ERRO
121 eh = uh - uex(x)
122
123 #-----CALCULANDO NORMAS DO ERRO-----
124 #norma L2
125 eL2form = fem.form(eh**2*dx)
126 L2norm = np.sqrt(fem.assemble_scalar(eL2form))
127
128 #norma H1
129 eH1form = fem.form( inner(grad(eh), grad(eh))*dx)
130 H1norm = np.sqrt( fem.assemble_scalar(eL2form) + fem.assemble_scalar(
131   eH1form) )
132
133 h = h_malha
134 return uh, eh, L2norm, H1norm, h, h_dim

```

A.5 Segundo problema com domínio irregular da subseção 3.1.2.4.2

```

1 def solucao_fem_2D(N, degree, Lx, Ly):
2

```

```

3  #----- criando a malha
4  cell_type = mesh.CellType.triangle
5  h_malha = Lx/N
6  radius_hole = Lx/4  # Raio do buraco
7
8  def GenerateMesh():
9      gmsh.initialize()
10     proc = MPI.COMM_WORLD.rank
11
12     if proc == 0:
13         background = gmsh.model.occ.addRectangle(0, 0, 0, Lx, Ly)
14
15         center_hole_x, center_hole_y = Lx/2, Ly/2 #Coordenadas do centro do
16         buraco.
17         hole = gmsh.model.occ.addDisk(center_hole_x, center_hole_y, 0,
18         radius_hole, radius_hole)# Criação do buraco
19         gmsh.model.occ.cut([(2, background)], [(2, hole)]) #Subtraindo o
20         buraco do retângulo
21
22         gmsh.model.occ.synchronize()
23         pl = gmsh.model.addPhysicalGroup(2, [background], 1)
24         pl_hole = gmsh.model.addPhysicalGroup(2, [hole], 2)
25
26         gmsh.model.mesh.setSize(gmsh.model.getEntities(0), h_malha)
27
28         if (cell_type == mesh.CellType.quadrilateral):
29             gmsh.model.mesh.setRecombine(2, pl)
30
31         gmsh.model.mesh.generate(2)
32
33         msh, _, _ = io.gmshio.model_to_mesh(gmsh.model, comm=MPI.COMM_WORLD,
34         rank=0, gdim=2)
35         gmsh.finalize()
36         return msh
37
38 msh = GenerateMesh()
39
40 V = fem.functionspace(msh, ("CG", degree)) #Espaço de funções contínuas
41 por partes CG
42
43 f = fem.Function(V)
44 #----- Condição de contorno -----
45
46 #Identificação dos graus de liberdade para as condições de contorno
47 dofsL = fem.locate_dofs_geometrical(V, lambda x: np.isclose(x[0], 0))
48 dofsR = fem.locate_dofs_geometrical(V, lambda x: np.isclose(x[0], Lx))
49 dofsD = fem.locate_dofs_geometrical(V, lambda x: np.isclose(x[1], 0))
50 dofsU = fem.locate_dofs_geometrical(V, lambda x: np.isclose(x[1], Ly))
51
52 #----- Condição de contorno DIRICHLET COMO FUNÇÃO RETÂ
53 NGULO -----
54 x = SpatialCoordinate(msh)
55 u_aux = fem.Function(V)
56 u_aux.interpolate(lambda x: np.sin(2*np.pi*x[0]) * np.sin(2*np.pi*x[1]))
57
58 #valor da condicao de contorno do buraco
59 def boundary_hole(x):
60     r = np.sqrt((x[0]-Lx/2)**2 + (x[1]-Ly/2)**2)
61     return np.isclose(r, radius_hole, rtol=1e-2, atol=1e-4)

```

```

56
57 dofs_circle_hole = fem.locate_dofs_geometrical(V, boundary_hole)
58 x = SpatialCoordinate(msh)
59
60 #Aplicando a condição de contorno de Dirichlet na fronteira do retângulo
    e do buraco (hole)
61 bcs = [fem.dirichletbc(u_aux, dofsL),
62        fem.dirichletbc(u_aux, dofsR),
63        fem.dirichletbc(u_aux, dofsD),
64        fem.dirichletbc(u_aux, dofsU), fem.dirichletbc(u_aux,
        dofs_circle_hole)]
65
66 #----- SOLUCAO ANALITICA USANDO LAMBDA -----
67 uex = lambda x: ufl.sin(2*ufl.pi*x[0])*ufl.sin(2*ufl.pi*x[1])
68 mu = lambda x: ufl.cos(2.0*ufl.pi*x[0])*ufl.cos(2.0*ufl.pi*x[1])+1.0
69
70 # Formulação variacional
71
72 # Funções Trial and test
73 u = ufl.TrialFunction(V)
74 v = ufl.TestFunction(V)
75
76 # Forma Bilinear
77 dx = ufl.dx
78 a = ufl.inner(mu(x)*ufl.grad(u), ufl.grad(v)) * dx
79
80 # Forma Linear
81 source = fem.Function(V)
82 source = -ufl.div(mu(x)*ufl.grad(uex(x)))
83 L = (source * v) * dx
84
85 # Soluciona o problema
86 problem = fem.petsc.LinearProblem(a, L, bcs=bcs, petsc_options={"
    ksp_type": "preonly", "pc_type": "lu"})
87 uh = problem.solve()
88
89 #----- PLOTAGEM -----
90 pv.OFF_SCREEN = True
91 u_topology, u_cell_types, u_geometry = dolfinx.plot.vtk_mesh(V)
92 u_grid = pv.UnstructuredGrid(u_topology, u_cell_types, u_geometry)
93 u_grid.point_data["u"] = uh.x.array.real
94 u_grid.set_active_scalars("u")
95
96 pl = pv.Plotter()
97 min_ = uh.x.array.min()
98 max_ = uh.x.array.max()
99 sargs = dict(height=0.6, vertical=True, position_x=0.85, position_y=0.15
100              ,title_font_size=20,
101              label_font_size=16,
102              shadow=True,
103              italic=True,
104              fmt="%3f",
105              font_family="arial",)
106 pl.add_mesh(u_grid,show_edges=True, clim=[min_, max_], scalar_bar_args=
    sargs)
107 pl.view_xy()
108 labels = dict(zlabel='uh', xlabel='x', ylabel='y')
109 pl.show_grid(**labels)
110 pl.add_axes(**labels)
111 figure = pl.screenshot("sol_aprox_n%d.png"%N)

```

```

112 files.download("sol_aprox_n%d.png"%N)
113
114 #----- SALVANDO ARQUIVO -----
115 uh.name = "u"
116 with io.XDMFFile(mpi4py.MPI.COMM_WORLD, "temp_malha_3_1_2_4_2.xdmf", "w")
    as file:
117     file.write_mesh(msh)
118     file.write_function(uh)
119 files.download('temp_malha_3_1_2_4_2.xdmf')
120 files.download('temp_malha_3_1_2_4_2.h5')
121
122 #----- Calculo do valor de h (diâmetro do elemento) -----
123 tdim = msh.topology.dim #dimensão topológica
124 num_cells = msh.topology.index_map(tdim).size_local + msh.topology.
    index_map(tdim).num_ghosts #número de elementos
125 msh.topology.create_connectivity(tdim-1, tdim)
126 h_e = msh.h(tdim-1, np.arange(num_cells))
127 h_dim = max(h_e)
128
129 #CÁLCULO DO ERRO
130 eh = uh - uex(x)
131
132 #----- CALCULANDO NORMAS DO ERRO -----
133 #norma L2
134 eL2form = fem.form(eh**2*dx)
135 L2norm = np.sqrt(fem.assemble_scalar(eL2form))
136
137 #norma H1
138 eH1form = fem.form( inner(grad(eh), grad(eh))*dx)
139 H1norm = np.sqrt( fem.assemble_scalar(eL2form) + fem.assemble_scalar(
    eH1form) )
140
141 h = h_malha
142 return uh, eh, L2norm, H1norm, h, h_dim

```

A.6 Problema 3D num tubo paralelepípedo da seção 3.13

```

1 def solucao_fem_3D_paralelepipedo(N, degree, Lx, Ly, Lz):
2     h_malha = Lx/N
3     msh = mesh.create_box(comm=mpi4py.MPI.COMM_WORLD, points=((0.0,0.0,0.0), (
        Lx, Ly, Lz)), n=(N,N,int((Lz/Lx)*N)), cell_type=mesh.CellType.
        tetrahedron)
4
5     V = fem.functionspace(msh, ("CG", degree))
6
7     #----- FORMULAÇÃO VARIACIONAL -----
8
9     #----- FUNÇÕES TRIAL e TEST -----
10    u = ufl.TrialFunction(V)
11    v = ufl.TestFunction(V)
12    source = fem.Function(V)
13    dx = ufl.dx
14
15    x = SpatialCoordinate(msh)
16    u_aux = fem.Function(V)
17    u_aux.interpolate(lambda x: np.sin(2*np.pi*x[0]) * np.sin(2*np.pi*x[1]))

```

```

18 uex = lambda x: ufl.sin(2*ufl.pi*x[0])*ufl.sin(2*ufl.pi*x[1])
19 mu = lambda x: ufl.cos(2.0*ufl.pi*x[0])*ufl.cos(2.0*ufl.pi*x[1])+1.0
20
21 a = ufl.inner(mu(x)*ufl.grad(u), ufl.grad(v)) * dx
22 source = -ufl.div(mu(x)*ufl.grad(uex(x)))
23 L = (source * v) * dx
24
25 #----- CONDIÇÕES DE CONTORNO -----
26 dofsL = fem.locate_dofs_geometrical(V, lambda x: np.isclose(x[0], 0))
27 dofsR = fem.locate_dofs_geometrical(V, lambda x: np.isclose(x[0], Lx))
28 dofsD = fem.locate_dofs_geometrical(V, lambda x: np.isclose(x[1], 0))
29 dofsU = fem.locate_dofs_geometrical(V, lambda x: np.isclose(x[1], Ly))
30
31 bcs = [fem.dirichletbc(u_aux, dofsL), fem.dirichletbc(u_aux, dofsR), fem.
        dirichletbc(u_aux, dofsD), fem.dirichletbc(u_aux, dofsU)]
32
33 #----- SOLUCIONANDO O PROBLEMA -----
34 problem = fem.petsc.LinearProblem(a, L, bcs=bcs, petsc_options={"
        ksp_type": "preonly", "pc_type": "lu"})
35 uh = problem.solve()
36
37 #----- EXPORTAÇÃO PARA VISUALIZAR NO PARAVIEW -----
38 arquivo_xdmf="sol_teste_paralelepipedo.xdmf"
39 arquivo_h5="sol_teste_paralelepipedo.h5"
40
41 with io.XDMFFile(msh.comm, arquivo_xdmf, "w") as file:
42     file.write_mesh(msh)
43     file.write_function(uh)
44 from google.colab import files
45 files.download(arquivo_xdmf)
46 files.download(arquivo_h5)
47
48 #----- CALCULANDO NORMAS DO ERRO -----
49 eh = uh - uex(x)
50
51 #NORMA L2
52 eL2form = fem.form(eh**2*dx)
53 L2norm = np.sqrt(fem.assemble_scalar(eL2form))
54
55 #NORMA H1
56 eH1form = fem.form(inner(grad(eh), grad(eh))*dx)
57 H1norm = np.sqrt(fem.assemble_scalar(eL2form) + fem.assemble_scalar(
        eH1form))
58
59 return uh, eh, L2norm, H1norm, h_malha

```

A.7 Problema 3D num tubo cilíndrico da seção 3.1.3.2

```

1 def solucao_fem_3D_cilindro(N, degree, radius, height):
2     h_malha = radius/N
3     c_x, c_y, c_z = 0.0, 0.0, 0.0
4
5     def GenerateMesh():
6         gmsh.initialize()
7         proc = MPI.COMM_WORLD.rank
8
9         if proc == 0:
10             background = gmsh.model.occ.addCylinder(c_x, c_y, c_z,

```

```

11                                     c_x,c_y, height, r=radius)
12
13     gmsh.model.occ.synchronize()
14     pl = gmsh.model.addPhysicalGroup(3, [background], 3)
15
16     gmsh.model.mesh.setSize(gmsh.model.getEntities(0), h_malha)
17     gmsh.model.mesh.generate(3)
18
19     msh, _, _ = io.gmshio.model_to_mesh(gmsh.model, comm=MPI.COMM_WORLD, rank
20     =0, gdim=3)
21     gmsh.finalize()
22     return msh
23
24 msh = GenerateMesh()
25
26 #----- ESPAÇO DE FUNÇÕES -----
27 V = fem.functionspace(msh, ("CG", degree))
28
29 #----- FORMULAÇÃO VARIACIONAL -----
30
31 #----- FUNÇÕES TRIAL e TEST -----
32 u = ufl.TrialFunction(V)
33 v = ufl.TestFunction(V)
34 source = fem.Function(V)
35 dx = ufl.dx
36
37 x = SpatialCoordinate(msh)
38 u_aux = fem.Function(V)
39 u_aux.interpolate(lambda x: -np.sin(x[0]**2)+np.sin(radius**2-x[1]**2))
40 uex = lambda x: -ufl.sin(x[0]**2)+ufl.sin(radius**2-x[1]**2) #solução
41 analítica
42
43 mu = fem.Constant(msh, 1.0)
44 a = ufl.inner(mu*ufl.grad(u), ufl.grad(v)) * dx
45 source = -ufl.div(mu*ufl.grad(uex(x)))
46 L = (source * v) * dx
47
48 #----- CONDIÇÕES DE CONTORNO -----
49 def boundary(x):
50     r = np.sqrt((x[0]-c_x)**2 + (x[1]-c_y)**2)
51     return np.isclose(r, radius, rtol=1e-4, atol=1e-6)
52
53 dofs_circle = fem.locate_dofs_geometrical(V, boundary)
54 bcs = [fem.dirichletbc(u_aux, dofs_circle)]
55
56 #----- SOLUCIONANDO O PROBLEMA -----
57 problem = fem.petsc.LinearProblem(a, L, bcs=bcs,
58     petsc_options={"ksp_type": "preonly", "
59     pc_type": "lu"})
60 uh = problem.solve()
61
62 #----- EXPORTAÇÃO PARA VISUALIZAR NO PARAVIEW -----
63
64 #SOLUÇÃO APROXIMADA
65 arquivo_xdmf="sol_cilindro.xdmf"
66 arquivo_h5="sol_cilindro.h5"
67
68 uh.name = "$u_h$"
69 with io.XDMFFile(msh.comm, arquivo_xdmf, "w") as file:
70     file.write_mesh(msh)
71     file.write_function(uh)

```

```

68 from google.colab import files
69 files.download(arquivo_xdmf)
70 files.download(arquivo_h5)
71
72 #SOLUÇÃO ANALÍTICA
73 arquivo_analitica_xdmf="sol_cilindro_analitica.xdmf"
74 arquivo_analitica_h5="sol_cilindro_analitica.h5"
75
76 u_aux.name = "$u$"
77 with io.XDMFFile(msh.comm, arquivo_analitica_xdmf, "w") as file:
78     file.write_mesh(msh)
79     file.write_function(u_aux)
80 from google.colab import files
81 files.download(arquivo_analitica_xdmf)
82 files.download(arquivo_analitica_h5)
83
84 #————— Calculo do valor de h (diâmetro do elemento) —————
85 tdim = msh.topology.dim #dimensão topológica
86 num_cells = msh.topology.index_map(tdim).size_local + msh.topology.
87     index_map(tdim).num_ghosts #número de elementos
88 msh.topology.create_connectivity(tdim-1, tdim)
89 h_e = msh.h(tdim-1, np.arange(num_cells))
90 h_dim = max(h_e)
91
92 #————— CALCULANDO NORMAS DO ERRO —————
93 eh = uh - uex(x)
94
95 #NORMA L2
96 eL2form = fem.form(eh**2*dx)
97 L2norm = np.sqrt(fem.assemble_scalar(eL2form))
98
99 #NORMA H1
100 eH1form = fem.form(inner(grad(eh), grad(eh))*dx)
101 H1norm = np.sqrt(fem.assemble_scalar(eL2form) + fem.assemble_scalar(
102     eH1form))
103
104 return uh, eh, L2norm, H1norm, h_malha, h_dim

```

A.8 Problema de Poisson da seção 3.2

```

1 def solucao_fem(N, degree, metodo, dt_value, analise_matriz):
2     msh = create_unit_square(mpi4py.MPI.COMM_WORLD, N, N, CellType.triangle)
3     V = fem.functionspace(msh, ("CG", degree))
4     x = SpatialCoordinate(msh)
5
6     #————— CONECTIVIDADE ENTRE AS FACES E AS CÉLULAS —————
7     msh.topology.create_connectivity(msh.topology.dim - 1, msh.topology.dim)
8
9     #————— DADOS DO PROBLEMA —————
10    dt = fem.Constant(msh, dt_value)
11    k_value = 1.0
12    k = fem.Constant(msh, k_value)
13    def f_exp(t, x):
14        return -x[0]**2 + x[0] + 2*t*k_value
15    f = fem.Function(V)
16    f.interpolate(partial(f_exp, 0))
17
18    def expression(t, x):

```

```

19     return -x[0]*(x[0]-1)*t
20
21     u0 = fem.Function(V)
22     u0.interpolate(partial(expression,0))
23
24     #----- GRAVANDO DADOS DA TEMPERATURA INICIAL -----
25     u0.name = "Temperatura0"
26     with io.XDMFFile(mpi4py.MPI.COMM_WORLD, "temperatura0.xdmf", "w") as file
27         :
28         file.write_mesh(msh)
29         file.write_function(u0)
30
31     #----- FORMULAÇÃO VARIACIONAL -----
32
33     #----- FUNÇÕES TRIAL E TEST -----
34     u = TrialFunction(V)
35     v = TestFunction(V)
36
37     if metodo=="explicito":
38         #----- FORMA LINEAR -----
39         a = inner(u, v) * dx
40         #----- FORMA BILINEAR -----
41         L = dt * inner(f, v) * dx + inner(u0, v) * dx - dt * inner(k*grad(u0),
42             grad(v)) * dx
43
44     if metodo=="implicito":
45         #----- FORMA LINEAR -----
46         a = inner(u, v) * dx + dt * inner(k*grad(u), grad(v)) * dx
47         #----- FORMA BILINEAR -----
48         L = dt * inner(f, v) * dx + inner(u0, v) * dx
49
50     a = fem.form(a)
51     L = fem.form(L)
52
53     #----- CONDIÇÕES DE CONTORNO -----
54
55     u_left = 0.0
56     u_right = 0.0
57
58     sdim = msh.topology.dim
59     fdim = sdim - 1
60
61     # FRONTEIRA ESQUERDA
62     facets_left = mesh.locate_entities_boundary(msh, fdim, lambda x: np.
63         isclose(x[0], 0.0))
64     dofsL = fem.locate_dofs_topological(V, fdim, facets_left)
65
66     # FRONTEIRA DIREITA
67     facets_right = mesh.locate_entities_boundary(msh, fdim, lambda x: np.
68         isclose(x[0], 1.0))
69     dofsR = fem.locate_dofs_topological(V, fdim, facets_right)
70
71     bcs = [fem.dirichletbc(u_left, dofsL, V), fem.dirichletbc(u_right, dofsR,
72         V)]
73
74     A = assemble_matrix(a, bcs=bcs)
75     A.assemble()
76
77     # https://www.mcs.anl.gov/petsc/petsc4py-current/docs/apiref/petsc4py.
78         PETSc.Mat-class.html
79     info = A.getInfo()

```

```

73
74 #----- ANÁLISES DA MATRIZ A -----
75 A_array=A.getValues(range(0, A.getSize()[0]), range(0, A.getSize()[1]))
76 if analyse_matriz=="sim":
77     verificacao_matriz(A_array)
78
79 b = fem.Function(V)
80
81 #DEFININDO O SOLVER
82 solver = PETSc.KSP().create(mpi4py.MPI.COMM_WORLD)
83 solver.setOperators(A)
84 solver.setType(PETSc.KSP.Type.PREONLY)
85 pc = solver.getPC()
86 pc.setType("lu")
87 # https://www.mcs.anl.gov/petsc/petsc4py-current/docs/apiref/petsc4py.
  PETSc.PC.Type-class.html
88
89 #----- GRAVANDO DADOS NO ARQUIVO -----
90 file = io.XDMFFile(mpi4py.MPI.COMM_WORLD, "temperatura.xdmf", "w")
91 file.write_mesh(msh)
92
93 uh = fem.Function(V)
94 uh.name = "temperatura"
95
96 t = 0
97 t_max = 1
98 while t < t_max:
99     t += dt_value
100
101 #ATUALIZA f
102 if metodo=="explicito":
103     f.interpolate(partial(f_exp,t-dt_value))
104 if metodo=="implicito":
105     f.interpolate(partial(f_exp,t))
106
107 #MONTA O LADO DIREITO DA EQUAÇÃO (RHS)
108 b.x.array[:] = 0
109 fem.petsc.assemble_vector(b.vector, L)
110
111 #APLICA A CONDIÇÃO DE CONTORNO
112 fem.petsc.apply_lifting(b.vector, [a], [bcs])
113 b.x.scatter_reverse(la.InsertMode.add)
114 fem.petsc.set_bc(b.vector, bcs)
115
116 #RESOLVE O SISTEMA LINEAR
117 uh.x.array[:] = 0
118 solver.solve(b.vector, uh.vector)
119 uh.x.scatter_forward()
120
121 #ATUALIZA u_0
122 u0.x.array[:] = uh.x.array
123
124 #SALVA A SOLUÇÃO NO TEMPO t
125 file.write_function(uh, t=t)
126 file.close()
127
128 #DOWNLOAD DOS ARQUIVOS
129 files.download('temperatura.xdmf')
130 files.download('temperatura.h5')
131 files.download('temperatura0.xdmf')

```

```

132 files.download('temperatura0.h5')
133
134 #----- CALCULANDO NORMAS DO ERRO -----
135 eh = uh - expression(t, x)
136
137 #NORMA L2
138 eL2form = fem.form(eh**2*dx)
139 L2norm = np.sqrt(fem.assemble_scalar(eL2form))
140
141 #NORMA H1
142 eH1form = fem.form( inner(grad(eh), grad(eh))*dx)
143 H1norm = np.sqrt( fem.assemble_scalar(eL2form) + fem.assemble_scalar(
    eH1form) )
144 return uh, L2norm, H1norm, msh, V, t

```

A.9 Problema five-spot da seção 3.3

```

1 def solucao_fem(N, degree, dt_value, analise_matriz):
2     msh = create_unit_square(mpi4py.MPI.COMM_WORLD, N, N, CellType.triangle)
3     V = fem.functionspace(msh, ("CG", degree)) #Espaço de funções continuous
4     x = SpatialCoordinate(msh) #Definindo as coordenadas espaciais
5
6     #----- CONECTIVIDADE ENTRE AS FACES E AS CÉLULAS -----
7     msh.topology.create_connectivity(msh.topology.dim - 1, msh.topology.dim)
8
9     #----- DADOS DO PROBLEMA -----
10    dt = fem.Constant(msh, dt_value)
11    k_value = 0.5 #constante A
12    k = fem.Constant(msh, k_value)
13    def f_exp(t, x):
14        return -x[0]*0 + t*0
15    f = fem.Function(V)
16    f.interpolate(partial(f_exp, 0))
17
18    def expression(t, x):
19        return -x[0]*(x[0]-1)*t
20
21    u0 = fem.Function(V)
22    u0.interpolate(partial(expression, 0))
23
24    #----- GRAVANDO DADOS DA TEMPERATURA INICIAL -----
25    u0.name = "Temp0"
26    with io.XDMFFile(mpi4py.MPI.COMM_WORLD, "temp0.xdmf", "w") as file:
27        file.write_mesh(msh)
28        file.write_function(u0)
29
30    #----- FORMULAÇÃO VARIACIONAL -----
31
32    #----- FUNÇÕES TRIAL E TEST -----
33    u = TrialFunction(V)
34    v = TestFunction(V)
35
36    #----- FORMA LINEAR -----
37    a = inner(u, v) * dx + dt * inner(k*grad(u), grad(v)) * dx
38    #----- FORMA BILINEAR -----
39    L = inner(u0, v) * dx
40

```

```

41 a = fem.form(a)
42 L = fem.form(L)
43
44 #----- CONDIÇÕES DE CONTORNO -----
45 u_inflow = fem.Constant(msh, 1.0)
46 u_outflow = fem.Constant(msh, 0.0)
47
48 def locate_dof_at_point(msh, point):
49     coords = msh.geometry.x[:, :2]
50     idx = np.where(np.all(np.isclose(coords, point), axis=1))[0]
51     if idx.size > 0:
52         return [idx[0]] # Retorna o primeiro índice encontrado
53     return []
54
55 #LOCALIZANDO DoFs PARA OS PONTOS (0, 0) e (1, 1)
56 dofs_inflow = locate_dof_at_point(msh, [0.0, 0.0])
57 dofs_outflow = locate_dof_at_point(msh, [1.0, 1.0])
58
59 #VERIFICANDO SE OS DoFs FORAM ENCONTRADOS
60 if not dofs_inflow:
61     print("DoF para (0,0) não encontrado.")
62 if not dofs_outflow:
63     print("DoF para (1,1) não encontrado.")
64
65 dofs_inflow = np.array(dofs_inflow, dtype=np.int32)
66 dofs_outflow = np.array(dofs_outflow, dtype=np.int32)
67
68 bcs = []
69 if dofs_inflow:
70     bcs.append(fem.dirichletbc(u_inflow, dofs_inflow, V))
71 if dofs_outflow:
72     bcs.append(fem.dirichletbc(u_outflow, dofs_outflow, V))
73
74 A = assemble_matrix(a, bcs=bcs)
75 A.assemble()
76
77 # https://www.mcs.anl.gov/petsc/petsc4py-current/docs/apiref/petsc4py.PETSc.Mat-class.html
78 info = A.getInfo()
79 b = fem.Function(V)
80
81 #DEFININDO O SOLVER
82 solver = PETSc.KSP().create(mpi4py.MPI.COMM_WORLD)
83 solver.setOperators(A)
84 solver.setType(PETSc.KSP.Type.PREONLY)
85 pc = solver.getPC()
86 # https://www.mcs.anl.gov/petsc/petsc4py-current/docs/apiref/petsc4py.PETSc.PC.Type-class.html
87 pc.setType("lu")
88
89 #----- GRAVANDO DADOS NO ARQUIVO -----
90 file = io.XDMFFile(mpi4py.MPI.COMM_WORLD, "temp.xdmf", "w")
91 file.write_mesh(msh)
92
93 uh = fem.Function(V)
94 uh.name = "temperatura"
95
96 t = 0
97 t_max = 10
98 while t < t_max:

```

```

99     t += dt_value
100
101     #MONTA O LADO DIREITO DA EQUAÇÃO (RHS)
102     b.x.array[:] = 0
103     fem.petsc.assemble_vector(b.vector, L)
104
105     #APLICA A CONDIÇÃO DE CONTORNO
106     fem.petsc.apply_lifting(b.vector, [a], [bcs])
107     b.x.scatter_reverse(la.InsertMode.add)
108     fem.petsc.set_bc(b.vector, bcs)
109
110     #RESOLVE O SISTEMA LINEAR
111     uh.x.array[:] = 0
112     solver.solve(b.vector, uh.vector)
113     uh.x.scatter_forward()
114
115     #ATUALIZA u_0
116     u0.x.array[:] = uh.x.array
117
118     #SALVA A SOLUÇÃO NO TEMPO t
119     file.write_function(uh, t=t)
120     file.close()
121
122     #DOWNLOAD DOS ARQUIVOS
123     files.download('temp.xdmf')
124     files.download('temp.h5')
125     files.download('temp0.xdmf')
126     files.download('temp0.h5')
127
128     #----- CALCULANDO NORMAS DO ERRO -----
129     eh = uh - expression(t, x) #Calculo do erro (subtração da sol. aproximada
130                                #com a analítica)
131
132     #NORMA L2
133     eL2form = fem.form(eh**2*dx)
134     L2norm = np.sqrt(fem.assemble_scalar(eL2form))
135
136     #NORMA H1
137     eH1form = fem.form(inner(grad(eh), grad(eh))*dx)
138     H1norm = np.sqrt(fem.assemble_scalar(eL2form) + fem.assemble_scalar(
139         eH1form))
140     return uh, L2norm, H1norm, msh, V, t

```

A.10 Problema de Advecção-Difusão da seção 3.4

```

1 def solucao_fem(N, degree, Lx, dt_value, download_arquivos):
2     msh = create_unit_square(mpi4py.MPI.COMM_WORLD, N, N, CellType.triangle)
3     V = fem.functionspace(msh, ("CG", degree))
4
5     #----- CONECTIVIDADE ENTRE AS FACES E AS CÉLULAS -----
6     msh.topology.create_connectivity(msh.topology.dim - 1, msh.topology.dim)
7
8     #----- DADOS DO PROBLEMA -----
9     dt = fem.Constant(msh, dt_value)
10    k = fem.Constant(msh, 1.0)
11    f = fem.Constant(msh, 0.0)
12    Pe = 1.0
13    x = SpatialCoordinate(msh)

```

```

14
15 alpha = np.array([1.0, 0.0])
16 alpha_vec = as_vector(alpha)
17
18 def expression(x):
19     return (ufl.exp(Pe*x[0]) - 1) / (ufl.exp(Pe) - 1)
20
21 u0 = fem.Function(V)
22 u0.interpolate(lambda x: 0+x[0]*0)
23
24 #----- FORMULAÇÃO VARIACIONAL -----
25
26 #----- FUNÇÕES TRIAL E TEST -----
27 u = TrialFunction(V)
28 v = TestFunction(V)
29
30 if metodo=="explicito":
31     #----- FORMA LINEAR -----
32     a = inner(u, v) * dx
33     #----- FORMA BILINEAR -----
34     L = dt * inner(f, v) * dx + inner(u0, v) * dx - dt * inner(inner(
        alpha_vec, grad(u0)), v) * dx - dt * inner(k*grad(u0), grad(v)) * dx
35
36 if metodo=="implicito":
37     #----- FORMA LINEAR -----
38     a = inner(u, v) * dx + dt * inner(inner(alpha_vec, grad(u)), v) * dx +
        dt * inner(k*grad(u), grad(v)) * dx
39     #----- FORMA BILINEAR -----
40     L = dt * inner(f, v) * dx + inner(u0, v) * dx
41
42 a = fem.form(a)
43 L = fem.form(L)
44
45 #----- CONDIÇÕES DE CONTORNO -----
46 uleft = 0
47 uright = 1
48
49 dofsL = fem.locate_dofs_geometrical(V, lambda x: np.isclose(x[0], 0))
50 dofsR = fem.locate_dofs_geometrical(V, lambda x: np.isclose(x[0], Lx))
51
52 x = SpatialCoordinate(msh)
53 bcs = [fem.dirichletbc(ScalarType(uleft), dofsL, V), fem.dirichletbc(
    ScalarType(uright), dofsR, V)]
54
55 A = assemble_matrix(a, bcs=bcs)
56 A.assemble()
57
58 # https://www.mcs.anl.gov/petsc/petsc4py-current/docs/apiref/petsc4py.
    PETSc.Mat-class.html
59 info = A.getInfo()
60
61 b = fem.Function(V)
62
63 #DEFININDO O SOLVER
64 solver = PETSc.KSP().create(mpi4py.MPI.COMM_WORLD)
65 solver.setOperators(A)
66 solver.setType(PETSc.KSP.Type.PREONLY)
67 pc = solver.getPC()
68 # https://www.mcs.anl.gov/petsc/petsc4py-current/docs/apiref/petsc4py.
    PETSc.PC.Type-class.html

```

```

69 pc.setType("lu")
70
71 #----- GRAVANDO DADOS NO ARQUIVO -----
72 file = io.XDMFFile(mpi4py.MPI.COMM_WORLD, "temperatura.xdmf", "w")
73 file.write_mesh(msh)
74
75 u = fem.Function(V)
76 u.name = "temperatura"
77 t = 0
78 t_max= 1.0
79 while t < t_max:
80     t += dt_value
81
82 #MONTA O LADO DIREITO DA EQUAÇÃO (RHS)
83 b.x.array[:] = 0
84 fem.petsc.assemble_vector(b.vector, L)
85
86 #APLICA A CONDIÇÃO DE CONTORNO
87 fem.petsc.apply_lifting(b.vector, [a], [bcs])
88 b.x.scatter_reverse(la.InsertMode.add)
89 fem.petsc.set_bc(b.vector, bcs)
90
91 #RESOLVE O SISTEMA LINEAR
92 u.x.array[:] = 0
93 solver.solve(b.vector, u.vector)
94 u.x.scatter_forward()
95
96 #ATUALIZA u_0
97 u0.x.array[:] = u.x.array
98
99 #SALVA A SOLUÇÃO NO TEMPO t
100 file.write_function(u, t=t)
101 file.close()
102
103 #DOWNLOAD DOS ARQUIVOS
104 if download_arquivos=="sim":
105     files.download('temperatura.xdmf')
106     files.download('temperatura.h5')
107
108 #----- CALCULANDO NORMAS DO ERRO -----
109 eh = u - expression(x)
110
111 #NORMA L2
112 eL2form = fem.form(eh**2*dx)
113 L2norm = np.sqrt(fem.assemble_scalar(eL2form))
114
115 #NORMA H1
116 eH1form = fem.form(inner(grad(eh), grad(eh))*dx)
117 H1norm = np.sqrt(fem.assemble_scalar(eL2form) + fem.assemble_scalar(
    eH1form))
118
119 return uh, L2norm, H1norm, msh, V, t, h_min, dt_value, num_cells

```

A.11 Simulação do escoamento em um campo homogêneo da seção 4.6

```

1 def solucao_fem(N, degree_u, degree_p, download_arquivos):

```

```

2  Lx = 1.0
3  Ly = 1.0
4
5  cell_type = CellType.triangle
6  msh = create_unit_square(mpi4py.MPI.COMM_WORLD, N, N, cell_type)
7  el_u = element("RT", msh.basix_cell(), degree_u) #Raviart-Thomas
8  el_p = element("DG", msh.basix_cell(), degree_p) #Discontinuous Galerkin
9
10 #DEFININDO ELEMENTO MISTO
11 el_m = mixed_element([el_u, el_p])
12
13 #ESPAÇO DE FUNÇÃO
14 W = fem.functionspace(msh, el_m)
15
16 x = SpatialCoordinate(msh) #Definindo as coordenadas espaciais
17
18 #----- CONECTIVIDADE ENTRE AS FACES E AS CÉLULAS -----
19 msh.topology.create_connectivity(msh.topology.dim - 1, msh.topology.dim)
20
21 #----- DADOS DO PROBLEMA -----
22 #----- CALCULO DO VALOR DE h MÍNIMO -----
23 k_value = 1
24 mu = 1
25 tdim = msh.topology.dim #dimensão topológica
26 num_cells = msh.topology.index_map(tdim).size_local + msh.topology.
    index_map(tdim).num_ghosts #número de elementos
27 msh.topology.create_connectivity(tdim-1, tdim)
28 h_list = msh.h(tdim-1, np.arange(num_cells))
29 h_min = min(h_list)
30
31 #----- CONDIÇÕES DE CONTORNO -----
32 # IDENTIFICANDO AS 4 FRONTEIRAS
33 boundaries = [(1, lambda x: np.isclose(x[0], 0)),
34               (2, lambda x: np.isclose(x[0], Lx)),
35               (3, lambda x: np.isclose(x[1], 0)),
36               (4, lambda x: np.isclose(x[1], Ly))]
37
38 facet_indices, facet_markers = [], []
39 fdim = msh.topology.dim - 1
40 for (marker, locator) in boundaries:
41     facets = mesh.locate_entities(msh, fdim, locator)
42     facet_indices.append(facets)
43     facet_markers.append(np.full_like(facets, marker))
44 facet_indices = np.hstack(facet_indices).astype(np.int32)
45 facet_markers = np.hstack(facet_markers).astype(np.int32)
46 sorted_facets = np.argsort(facet_indices)
47 facet_tag = mesh.meshtags(msh, fdim, facet_indices[sorted_facets],
    facet_markers[sorted_facets])
48
49 ds = Measure("ds", domain=msh, subdomain_data=facet_tag)
50 dx = ufl.dx
51
52 if (False):
53     one = fem.Constant(msh, 1.0)
54     for i in [1, 2, 3, 4]:
55         length_form = fem.form(one*ds(i))
56         lengthside = fem.assemble_scalar(length_form)
57         print(lengthside)
58
59 #----- FORMULAÇÃO VARIACIONAL -----

```

```

60 # TERMO FONTE
61 f = fem.Constant(msh, 2.0)
62
63 #CONDIÇÕES DE CONTO RNO NATURAL
64 pleft = fem.Constant(msh, 0.0)
65 pright = fem.Constant(msh, 0.0)
66
67 #----- FUNÇÕES TRIAL E TEST -----
68 TrialF = TrialFunction(W)
69 TestF = TestFunction(W)
70 (u, p) = split(TrialF)
71 (v, q) = split(TestF)
72
73 n = FacetNormal(msh)
74 #----- FORMA LINEAR -----
75 B = ((mu/k_value)*inner(u, v) - div(v)*p - div(u)*q) * dx
76 #----- FORMA BILINEAR -----
77 L = - f * q * dx - pleft*dot(v,n)*ds(1) - pright*dot(v,n)*ds(2)
78
79 #----- CONDIÇÃO DE CONTO RNO ESSENCIAL -----
80
81 #Define algumas funções, como g.n = v
82 def g(x, n):
83     return (np.array(2*x[0] - 1), np.zeros_like(x[0]))
84
85 W0, U_to_W = W.sub(0).collapse()
86 w_bc = fem.Function(W0)
87
88 sdim = msh.topology.dim
89 top_facets = mesh.locate_entities_boundary(msh, sdim-1, lambda x: np.
    isclose(x[1], Ly))
90 top_cells = mesh.compute_incident_entities(msh.topology, top_facets,
    sdim-1, sdim)
91 w_bc.interpolate(lambda x: g(x, (0,1)), top_cells)
92
93 bottom_facets = mesh.locate_entities_boundary(msh, sdim-1, lambda x: np.
    isclose(x[1], 0))
94 bottom_cells = mesh.compute_incident_entities(msh.topology,
    bottom_facets, sdim-1, sdim)
95 w_bc.interpolate(lambda x: g(x, (0,1)), bottom_cells)
96
97 w_bc.x.scatter_forward()
98
99 bc_dofs = fem.locate_dofs_topological((W.sub(0), W0), sdim-1, np.hstack([
    top_facets, bottom_facets]))
100 bcs = [fem.dirichletbc(w_bc, bc_dofs, W.sub(0))]
101
102 #----- SOLUÇÃO DO PROBLEMA -----
103 wh = fem.Function(W)
104 petsc_opts={"ksp_type": "preonly", "pc_type": "lu", "
    pc_factor_mat_solver_type": "superlu", "ksp_monitor": None}
105 problem = LinearProblem(B, L, bcs=bcs, petsc_options=petsc_opts)
106 wh = problem.solve()
107 (uh, ph) = wh.split()
108
109 # VERIFICAÇÃO
110 inflow = fem.assemble_scalar(fem.form(dot(uh,n)*ds(1)))
111 outflow = fem.assemble_scalar(fem.form(dot(uh,n)*ds(2)))
112 bottomflow = fem.assemble_scalar(fem.form(dot(uh,n)*ds(3)))
113 topflow = fem.assemble_scalar(fem.form(dot(uh,n)*ds(4)))

```

```

114 print("\nInflow =%f\nOutflow=%f\nTopflow=%f\nBotflow=%f" %(inflow ,
    outflow , topflow , bottomflow))
115
116 #----- PLOTAGEM -----
117 BDM_out = fem.functionspace(msh, ("CG", 1, (msh.geometry.dim,)))
118
119 v_aprox = fem.Function(BDM_out)
120 v_aprox.interpolate(uh)
121 v_aprox.name = "Velocidade"
122 with io.XDMFFile(msh.comm, "vel.xdmf", "w") as xdmf:
123     xdmf.write_mesh(msh)
124     xdmf.write_function(v_aprox)
125
126 ph.name = "Pressão"
127 with io.XDMFFile(msh.comm, "pres.xdmf", "w") as xdmf:
128     xdmf.write_mesh(msh)
129     xdmf.write_function(ph)
130
131 if download_arquivos=="sim":
132     from google.colab import files
133     files.download('vel.xdmf')
134     files.download('vel.h5')
135     files.download('pres.xdmf')
136     files.download('pres.h5')
137
138 #----- NORMA DO ERRO (PRESSÃO) -----
139 #SOLUÇÃO ANALÍTICA DA PRESSÃO 'p'
140 p_ex = lambda x: x[0]*(1-x[0])
141
142 #INTERPOLANDO
143 Vp = fem.functionspace(msh, ("DG", 0))
144 ph_aprox = fem.Function(Vp)
145 ph_aprox.interpolate(ph)
146
147 #ERRO DE p
148 e_p = ph_aprox - p_ex(x)
149
150 #NORMA L2
151 dx = ufl.dx
152 eL2form = fem.form(e_p**2*dx)
153 L2_norm = np.sqrt(fem.assemble_scalar(eL2form))
154
155 #----- NORMA DO ERRO (VELOCIDADE) -----
156 #SOLUÇÃO ANALÍTICA DA VELOCIDADE 'v'
157 def v_aux(x):
158     return (np.array(2*x[0] - 1), np.zeros_like(x[0]))
159
160 #INTERPOLANDO
161 v_ex = fem.Function(BDM_out)
162 v_ex.interpolate(v_aux)
163
164 #ERRO DE v
165 e_v = v_aprox - v_ex(x)
166 Hdiv_form = fem.form( inner(e_v,e_v)*dx + inner(div(e_v),div(e_v))*dx)
167 Hdiv_norm = np.sqrt(fem.assemble_scalar(Hdiv_form))
168
169 return uh, ph, msh, h_min, L2_norm, Hdiv_norm

```

A.12 Problema de Transporte Passivo da Temperatura em Meios Porosos do capítulo 5

```

1 def DARCY(degree_u, degree_p):
2     # DIMENSÕES
3     Lx, Ly = 110, 30
4     Nx, Ny = 220, 60
5     Dx, Dy = Lx / Nx, Ly / Ny
6
7     cell_type = CellType.triangle
8     msh = create_rectangle(comm=MPI.COMM_WORLD,
9                             points=((0.0, 0.0), (Lx, Ly)), n=(Nx, Ny),
10                                cell_type=cell_type)
11     el_u = element("RT", msh.basix_cell(), degree_u) #Raviart–Thomas
12     el_p = element("DG", msh.basix_cell(), degree_p) #Discontinuous Galerkin
13
14     #DEFININDO O ELEMENTO MISTO
15     el_m = mixed_element([el_u, el_p])
16
17     #ESPAÇO GLOBAL CORRESPONDENTE
18     W = fem.functionspace(msh, el_m)
19
20     x = SpatialCoordinate(msh) #Definindo as coordenadas espaciais
21
22     #————— CONECTIVIDADE ENTRE AS FACES E AS CÉLULAS —————
23     msh.topology.create_connectivity(msh.topology.dim - 1, msh.topology.dim)
24
25     #————— DADOS DO PROBLEMA —————
26     #————— CONDIÇÕES DE CONTORNO —————
27     # IDENTIFICANDO AS 4 FRONTEIRAS
28     boundaries = [(1, lambda x: np.isclose(x[0], 0)),
29                  (2, lambda x: np.isclose(x[0], Lx)),
30                  (3, lambda x: np.isclose(x[1], 0)),
31                  (4, lambda x: np.isclose(x[1], Ly))] #default: atol = e-08.
32     rtol = e-05
33
34     facet_indices, facet_markers = [], []
35     fdim = msh.topology.dim - 1
36     for (marker, locator) in boundaries:
37         facets = mesh.locate_entities(msh, fdim, locator)
38         facet_indices.append(facets)
39         facet_markers.append(np.full_like(facets, marker))
40     facet_indices = np.hstack(facet_indices).astype(np.int32)
41     facet_markers = np.hstack(facet_markers).astype(np.int32)
42     sorted_facets = np.argsort(facet_indices)
43     facet_tag = mesh.meshtags(msh, fdim, facet_indices[sorted_facets],
44                               facet_markers[sorted_facets])
45
46     #DEFINE A MEDIDA PARA CALCULAR A INTEGRAL SOBRE AS FRONTEIRAS
47     ds = Measure("ds", domain=msh, subdomain_data=facet_tag)
48     if (False):
49         one = fem.Constant(msh, 1.0)
50         for i in [1, 2, 3, 4]:
51             length_form = fem.form(one*ds(i))
52             lengthside = fem.assemble_scalar(length_form)
53             print(lengthside)
54
55     #————— PERMEABILIDADE HETEROGÊNEA —————
56     el_kappa = element("DG", msh.basix_cell(), 0)

```

```

55 K = fem.functionspace(msh, el_kappa)
56 kappax = fem.Function(K)
57 invkappax = fem.Function(K)
58
59 #LEITURA DOS DADOS
60 table = open('perm_field.dat', 'r')
61 lista = table.readlines()
62 table.close()
63
64 kx_array = np.zeros(shape=(Nx, Ny))
65 for i in range(Nx):
66     for j in range(Ny):
67         k = Nx * j + i
68         kx_array[i, j] = float(lista[k].split()[0])
69
70 #----- PLOTAGEM -----
71 import matplotlib.pyplot as plt
72 plt.figure(figsize=(33/3, 3))
73 plt.imshow(np.log(kx_array[:, :].T), origin="lower", cmap='gist_rainbow')
74 plt.colorbar()
75 plt.show()
76
77 def EvalKappa(x):
78     icellf = 0.99999999*x[0]/Dx
79     jcellf = 0.99999999*x[1]/Dy
80     icell = icellf.astype(int)
81     jcell = jcellf.astype(int)
82     return kx_array[icell, jcell]
83
84 kappax.interpolate(EvalKappa)
85
86 # VISUALIZAÇÃO NO PARAVIEW
87 kappax.name = "permeabilidade"
88 with io.XDMFFile(msh.comm, "PRINCIPAL_perm.xdmf", "w") as xdmf:
89     xdmf.write_mesh(msh)
90     xdmf.write_function(kappax)
91
92 from google.colab import files
93 files.download('PRINCIPAL_perm.xdmf')
94 files.download('PRINCIPAL_perm.h5')
95
96 #----- FORMULAÇÃO VARIACIONAL -----
97 f = fem.Constant(msh, 0.0) # TERMO FONTE
98 mu = 1.0
99
100 #----- CONDIÇÃO DE CONTORNO NATURAL -----
101 pleft = fem.Constant(msh, 100.0)
102 pright = fem.Constant(msh, 0.0)
103
104 #----- FUNÇÕES TRIAL E TEST -----
105 TrialF = TrialFunction(W)
106 TestF = TestFunction(W)
107 (u, p) = split(TrialF)
108 (v, q) = split(TestF)
109
110 n = FacetNormal(msh)
111 #----- FORMA LINEAR -----
112 B = ((mu/kappax)*inner(u, v) - div(v)*p - div(u)*q) * dx
113 #----- FORMA BILINEAR -----
114 L = - f * q * dx - pleft*dot(v, n)*ds(1) - pright*dot(v, n)*ds(2)

```

```

115
116 #----- CONDIÇÃO DE CONTORNO ESSENCIAL -----
117 # DEFINE ALGUMAS FUNÇÕES, COMO g.n = v
118 def g(x, n):
119     return (np.zeros_like(x[0]), np.zeros_like(x[0]))
120
121 W0, U_to_W = W.sub(0).collapse()
122 w_bc = fem.Function(W0)
123
124 sdim = msh.topology.dim
125 top_facets = mesh.locate_entities_boundary(msh, sdim-1, lambda x: np.
126     isclose(x[1], Ly))
127 top_cells = mesh.compute_incident_entities(msh.topology, top_facets, sdim
128     -1, sdim)
129 w_bc.interpolate(lambda x: g(x, (0,1)), top_cells)
130
131 bottom_facets = mesh.locate_entities_boundary(msh, sdim-1, lambda x: np.
132     isclose(x[1], 0))
133 bottom_cells = mesh.compute_incident_entities(msh.topology, bottom_facets
134     , sdim-1, sdim)
135 w_bc.interpolate(lambda x: g(x, (0,1)), bottom_cells)
136
137 w_bc.x.scatter_forward()
138
139 bc_dofs = fem.locate_dofs_topological((W.sub(0), W0), sdim-1, np.hstack([
140     top_facets, bottom_facets]))
141 bcs = [fem.dirichletbc(w_bc, bc_dofs, W.sub(0))]
142
143
144 #----- SOLUÇÃO DO PROBLEMA -----
145 wh = fem.Function(W)
146 petsc_opts={"ksp_type": "preonly", "pc_type": "lu", "
147     pc_factor_mat_solver_type": "superlu", "ksp_monitor": None}
148 problem = LinearProblem(B, L, bcs=bcs, petsc_options=petsc_opts)
149 wh = problem.solve()
150 (uh,ph) = wh.split()
151
152 # VERIFICAÇÃO
153 inflow = fem.assemble_scalar(fem.form(dot(uh,n)*ds(1)))
154 outflow = fem.assemble_scalar(fem.form(dot(uh,n)*ds(2)))
155 bottomflow = fem.assemble_scalar(fem.form(dot(uh,n)*ds(3)))
156 topflow = fem.assemble_scalar(fem.form(dot(uh,n)*ds(4)))
157 print("\nInflow=%f\nOutflow=%f\nTopflow=%f\nBotflow=%f" %(inflow,
158     outflow, topflow, bottomflow))
159
160 #----- PLOTAGEM -----
161 BDM_out = fem.functionspace(msh, ("Lagrange", 1, (msh.geometry.dim,)))
162
163 u_out = fem.Function(BDM_out)
164 u_out.interpolate(uh)
165 u_out.name = "Velocidade"
166 with io.XDMFFile(msh.comm, "PRINCIPAL_vel.xdmf", "w") as xdmf:
167     xdmf.write_mesh(msh)
168     xdmf.write_function(u_out)
169
170 ph.name = "Pressão"
171 with io.XDMFFile(msh.comm, "PRINCIPAL_pres.xdmf", "w") as xdmf:
172     xdmf.write_mesh(msh)
173     xdmf.write_function(ph)
174
175 from google.colab import files

```

```

168 files.download('PRINCIPAL_vel.xdmf')
169 files.download('PRINCIPAL_vel.h5')
170 files.download('PRINCIPAL_pres.xdmf')
171 files.download('PRINCIPAL_pres.h5')
172 return uh, u_out, ph, msh, Lx, Ly
173
174 #-----#
175 #-----#
176
177 def TRANSIENTE_ADV_DIF(u_out, degree_T, Lx, Ly, msh):
178     V = fem.functionspace(msh, ("CG", degree_T))
179
180     #----- CONECTIVIDADE ENTRE AS FACES E AS CÉLULAS -----#
181     msh.topology.create_connectivity(msh.topology.dim - 1, msh.topology.dim)
182
183     #----- DADOS DO PROBLEMA -----#
184     #----- CALCULO DO VALOR DO hmin -----#
185     tdim = msh.topology.dim #dimensão topológica
186     num_cells = msh.topology.index_map(tdim).size_local + msh.topology.
187         index_map(tdim).num_ghosts #número de elementos
188     msh.topology.create_connectivity(tdim-1, tdim)
189     h_list = msh.h(tdim-1, np.arange(num_cells))
190     h_min = min(h_list)
191     fator = 0.8 #fator de segurança
192
193     varphi = 0.3
194     rho_c_fluido = 1.0 * 4.5 #rho * c_p
195     rho_c_solido = 2.8 * 1.0 #rho~ * c_p~
196     rho_c_m = varphi*rho_c_fluido + (1-varphi)*rho_c_solido #rho_m * c_m
197     k_m = varphi*0.58 + (1-varphi)*1.0
198
199     dt_calc = fator * (2* h_min**2)/k_m #calculo do dt
200     dt_value = np.floor(dt_calc*(10**4))/(10**4) #arredondando dt para baixo
201     # com 4 casas decimais.
202     dt = fem.Constant(msh, dt_value)
203     k = fem.Constant(msh, k_m)
204     am = fem.Constant(msh, rho_c_m)
205     a_1 = fem.Constant(msh, rho_c_fluido*dt_value) #rho * c * dt
206     x = SpatialCoordinate(msh)
207
208     T0 = fem.Function(V)
209     T0.interpolate(lambda x: 20+x[0]*0)
210
211     #----- FORMULAÇÃO VARIACIONAL -----#
212
213     #----- FUNÇÕES TRIAL E TEST -----#
214
215     T = TrialFunction(V)
216     v = TestFunction(V)
217
218     #----- FORMA LINEAR -----#
219     a = am*inner(T, v) * dx + a_1 * inner(inner(u_out, grad(T)), v) * dx + dt
220         * inner(k*grad(T), grad(v)) * dx
221     #----- FORMA BILINEAR -----#
222     L = am*inner(T0, v) * dx
223
224     a = fem.form(a)
225     L = fem.form(L)
226
227     #----- CONDIÇÕES DE CONTORNO -----#
228     Tleft = 80

```

```

225 Tright = 20
226
227 dofsL = fem.locate_dofs_geometrical(V, lambda x: np.isclose(x[0], 0))
228 dofsR = fem.locate_dofs_geometrical(V, lambda x: np.isclose(x[0], Lx))
229
230 x = SpatialCoordinate(msh)
231 bcs = [fem.dirichletbc(ScalarType(Tleft), dofsL, V), fem.dirichletbc(
    ScalarType(Tright), dofsR, V)]
232
233 A = assemble_matrix(a, bcs=bcs)
234 A.assemble()
235
236 # https://www.mcs.anl.gov/petsc/petsc4py-current/docs/apiref/petsc4py.
    PETSc.Mat-class.html
237 info = A.getInfo()
238
239 b = create_vector(L)
240
241 #DEFININDO O SOLVER
242 solver = PETSc.KSP().create(mpi4py.MPI.COMM_WORLD)
243 solver.setOperators(A)
244 solver.setType(PETSc.KSP.Type.PREONLY)
245 pc = solver.getPC()
246 # https://www.mcs.anl.gov/petsc/petsc4py-current/docs/apiref/petsc4py.
    PETSc.PC.Type-class.html
247 pc.setType("lu")
248
249 #----- GRAVANDO DADOS NO ARQUIVO -----
250 file = io.XDMFFile(mpi4py.MPI.COMM_WORLD, "PRINCIPAL_Temp.xdmf", "w")
251 file.write_mesh(msh)
252
253 Th = fem.Function(V)
254 Th.name = "Temperatura"
255 t = 0
256 t_max= 1000.0
257 dt_plot = 25
258 contador = 0
259 contador_plot = 0
260 file.write_function(Th, t=0) #SALVANDO VALOR PARA t=0
261 print('Plot com t', contador_plot, ' = ', t)
262
263 while t < t_max:
264     t += dt_value
265
266 #MONTA O LADO DIREITO DA EQUAÇÃO (RHS)
267 with b.localForm() as loc_b:
268     loc_b.set(0)
269     fem.petsc.assemble_vector(b, L)
270
271 #APLICA A CONDIÇÃO DE CONTORNO
272 fem.petsc.apply_lifting(b, [a], [bcs])
273 b.ghostUpdate(addv=PETSc.InsertMode.ADD_VALUES, mode=PETSc.ScatterMode.
    REVERSE)
274 fem.petsc.set_bc(b, bcs)
275
276 #RESOLVE O SISTEMA LINEAR
277 Th.x.array[:] = 0
278 solver.solve(b, Th.x.petsc_vec)
279 Th.x.scatter_forward()
280

```

```
281 #ATUALIZA u_0
282 T0.x.array[:] = Th.x.array
283
284 #SALVA A SOLUÇÃO NO TEMPO t (A CADA 'dt_plot' ITERAÇÕES)
285 if contador % dt_plot == 0:
286     contador_plot += 1
287     print('Plot com t', contador_plot, ' = ', t)
288     file.write_function(Th, t=t)
289     contador += 1
290 file.write_function(Th, t=t)
291 print('Plot com t', contador_plot+1, ' = ', t)
292 file.close()
293
294 #DOWNLOAD DOS ARQUIVOS
295 files.download('PRINCIPAL_Temp.xdmf')
296 files.download('PRINCIPAL_Temp.h5')
297
298 print('')
299 print('Resultado para t = ', t, ' s')
300 print('t_max = ', t_max)
301 print('fator = ', fator)
302 print('')
303
304 return Th, h_min, dt_value
```