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

Explorando Devito: uma Abordagem para Otimizar a Resolução de Escoamentos Viscosos por Diferenças Finitas

Thiago Burgo Ikeda

Orientadora: Prof[a]. Dr[a]. Gilcilene Sanchez de Paulo

Programa: Matemática Aplicada e Computacional

Presidente Prudente, Maio de 2026

UNIVERSIDADE ESTADUAL PAULISTA

Faculdade de Ciências e Tecnologia de Presidente Prudente

Programa de Pós-Graduação em Matemática Aplicada e Computacional

Explorando Devito: uma Abordagem para Otimizar a Resolução de Escoamentos Viscosos por Diferenças Finitas

Thiago Burgo Ikeda

Orientadora: Prof[a]. Dr[a]. 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 obteção do título de Mestre em Matemática Aplicada e Computacional.

Presidente Prudente, Maio de 2026

I26e

Ikeda, Thiago Burgo
Explorando Devito : uma abordagem para otimizar a resolução de escoamentos viscosos por diferenças finitas / Thiago Burgo Ikeda. -- Presidente Prudente, 2026
84 f. : tabs.

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

1. Devito. 2. Equações de Navier-Stokes. 3. CFD. 4. Diferenças Finitas. 5. PETSc. I. Título.

Sistema de geração automática de fichas catalográficas da Unesp. Dados fornecidos pelo autor(a).

UNIVERSIDADE ESTADUAL PAULISTA

Câmpus de Presidente Prudente

CERTIFICADO DE APROVAÇÃO

TÍTULO DA DISSERTAÇÃO: Explorando Devito: uma Abordagem para Otimizar a Resolução de Escoamentos Viscosos por Diferenças Finitas

AUTOR: THIAGO BURGO IKEDA
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 (Participaçao Presencial)
Departamento de Matemática e Computação / UNESP / Câmpus de Presidente Prudente - FCT

Prof. Dr. IRINEU LOPES PALHARES JUNIOR (Participaçao Presencial)
Departamento de Matemática e Computação / UNESP / Câmpus de Presidente Prudente - FCT

Prof. Dr. MANOEL SILVINO BATALHA DE ARAÚJO (Participaçao Virtual)
Faculdade de Matemática - Instituto de Ciências Exatas e Naturais/Universidade Federal do Pará - UFPA

Presidente Prudente, 29 de maio de 2026.

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

Agradecimentos

Dirijo meus mais sinceros agradecimentos à minha família, pelo apoio inabalável e pela presença constante, que foram essenciais para meu encorajamento ao longo deste caminho. À minha orientadora, Prof[a]. Dr[a]. Gilcilene Sanchez de Paulo, manifesto minha profunda gratidão por sua sábia orientação e pelas reuniões tão proveitosas, que ampliaram significativamente minha percepção sobre o tema. À minha namorada, agradeço pelo carinho, paciência e apoio ao longo de todo este período. Não poderia deixar de estender este agradecimento aos demais professores e colegas, cujas contribuições, das mais variadas formas, foram indispensáveis para o meu desenvolvimento acadêmico.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

"Sempre há algo novo a se aprender,
mesmo para um mestre."
Mestre Shifu

Resumo

Este trabalho investiga a aplicabilidade da plataforma Devito na resolução numérica de escoamentos viscosos incompressíveis por diferenças finitas, utilizando a formulação corrente-vorticidade ($\psi - \omega$) das equações de NavierStokes. Inicialmente, são realizados testes de verificação em problemas clássicos com soluções manufaturadas, permitindo avaliar a robustez das discretizações implementadas.

Em seguida, a formulação $\psi - \omega$ é aplicada ao problema da cavidade *lid-driven* para números de Reynolds $Re = 100$, 400 e 10000. Os resultados numéricos obtidos são comparados com soluções clássicas e consolidadas da literatura, evidenciando a capacidade da metodologia em reproduzir adequadamente as principais estruturas do escoamento.

O estudo também analisa aspectos computacionais da plataforma, identificando limitações relacionadas à resolução de sistemas lineares por meio de métodos explícitos de *pseudo-timestepping*, cujo custo computacional se torna proibitivo em malhas refinadas. Como parte da análise computacional realizada neste trabalho, é explorada a integração experimental entre o Devito e a biblioteca PETSc, proposta recentemente pelos desenvolvedores da plataforma. Os testes realizados neste trabalho indicam aceleração superior a 1200 vezes na solução da equação de Poisson.

Por fim, documenta-se uma outra limitação no Devito que compromete a implementação correta de esquemas *upwind* com seleção dinâmica de direção. Embora essa limitação restrinja determinadas abordagens numéricas, ela também evidencia oportunidades para o desenvolvimento de novas estratégias de discretização e implementação adaptadas à arquitetura da plataforma. Apesar das limitações observadas, os resultados indicam que o Devito constitui uma ferramenta promissora baseada em diferenças finitas para aplicações em Dinâmica dos Fluidos Computacional, especialmente quando combinado com *solvers* externos especializados, como o PETSc.

Palavras-Chave: *Devito, Equações de Navier-Stokes, CFD, Diferenças Finitas, PETSc.*

Abstract

This work investigates the applicability of the Devito platform for the numerical solution of incompressible viscous flows using finite differences, employing the streamfunction-vorticity ($\psi - \omega$) formulation of the NavierStokes equations. Initially, verification tests are performed on classical problems with manufactured solutions, allowing an assessment of the robustness of the implemented discretizations.

Next, the $\psi - \omega$ formulation is applied to the lid-driven cavity problem for Reynolds numbers $Re = 100$, 400, and 10000. The obtained numerical results are compared with classical and well-established solutions from the literature, demonstrating the methodology's ability to adequately reproduce the main flow structures.

The study also analyzes computational aspects of the platform, identifying limitations related to solving linear systems using explicit pseudo-timestepping methods, whose computational cost becomes prohibitive on refined meshes. As part of the computational analysis carried out in this work, the experimental integration between Devito and the PETSc library, recently proposed by the platform's developers, is explored. The tests conducted in this work indicate an acceleration of over 1200 times in the solution of the Poisson equation.

Finally, another limitation in Devito is documented, which compromises the correct implementation of upwind schemes with dynamic directional selection. Although this limitation restricts certain numerical approaches, it also highlights opportunities for the development of new discretization and implementation strategies tailored to the platform's architecture. Despite the observed limitations, the results indicate that Devito constitutes a promising finite-difference-based tool for Computational Fluid Dynamics applications, especially when combined with specialized external solvers such as PETSc.

Keywords: *Devito, Navier-Stokes Equations, CFD, Finite Difference, PETSc.*

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo

1

Introdução

O estudo e a resolução de problemas modelados por equações diferenciais desempenham um papel fundamental em diversas áreas da ciência e engenharia [26]. A técnica de diferenças finitas é uma abordagem comum para resolver equações diferenciais ordinárias (EDOs) e parciais (EDPs), envolvendo a discretização do domínio do problema e a aproximação por meio de equações de diferenças para calcular as derivadas das funções no espaço e no tempo [6, 7, 10, 18].

Embora eficaz, a implementação manual de códigos de diferenças finitas pode ser trabalhosa, suscetível a erros e exigir conhecimento técnico substancial de programação sofisticada. Outro ponto importante a ser considerado nesta área é a complexidade dos problemas numéricos de grande porte e com riqueza de detalhes a serem resolvidos atualmente [4, 25]. Essas simulações consomem uma quantidade significativa de recursos computacionais e tempo de processamento, muitas vezes exigindo supercomputadores para serem concluídas em um prazo razoável, o que torna esses problemas desafiadores em termos de escalabilidade e eficiência computacional.

Neste contexto, surge o Devito, uma ferramenta de código aberto desenvolvida em Python™, projetada para automatizar a geração e otimização de códigos de diferenças finitas [19, 20]. O Devito foi originalmente desenvolvido para resolver aplicações sísmicas, incluindo a propagação de ondas sísmicas em diferentes tipos de materiais geológicos. Porém, o *framework* também foi testado para resolver problemas clássicos de CFD, demonstrando sua flexibilidade em outros domínios de aplicação, como a equação de convecção, a equação de Burgers, a equação de Poisson, a equação de águas rasas, a equação de Darcy, bem como uma formulação inicial para as equações de Navier-Stokes que resolve o problema da cavidade.

Apesar dessas potencialidades, a formulação atualmente implementada no Devito para as equações de Navier-Stokes baseia-se no cálculo da pressão por meio de uma equação de Poisson que, na maioria dos problemas, não admite uma condição de contorno natural fisicamente aceitável. Adicionalmente, o Devito tem utilizado predominantemente métodos explícitos para a solução de sistemas lineares como o método de Jacobi para a equação de Poisson cuja convergência é computacionalmente lenta, conforme amplamente documentado na teoria de métodos numéricos. Não há na literatura, até o momento, uma implementação sistemática e validada da formulação corrente-vorticidade ($\psi - \omega$) [1, 11, 13, 22] utilizando o Devito, que permitiria contornar a primeira limitação. No caso da segunda limitação, os desenvolvedores do Devito tem trabalhado na integração da biblioteca PETSc ao Devito para resolver mais eficientemente a equação de Poisson.

Diante desse cenário, este trabalho preenche essa lacuna ao apresentar três contribuições originais:

- **Implementação e validação da formulação $\psi - \omega$ no Devito** para escoamentos viscosos incompressíveis, aplicada ao problema clássico da cavidade *lid-driven*. Esta formulação elimina a necessidade de cálculo explícito da pressão, satisfazendo automaticamente a equação da continuidade;

- **Análise sistemática de precisão numérica** utilizando o método das soluções manufaturadas para equações elípticas, parabólicas e hiperbólicas. Ao longo do Capítulo 3, são explicitamente distinguidos os componentes implementados originalmente neste trabalho (termos fonte arbitrários) das funcionalidades nativas do Devito;

- **Investigação experimental da integração Devito-PETSc**, quantificando o ganho de desempenho obtido ao substituir o pseudo-transiente por *solvers* iterativos implícitos. Os resultados demonstram aceleração de até três ordens de grandeza para malhas bidimensionais moderadamente refinadas, estabelecendo um caminho promissor para simulações de maior escala.

Além dessas contribuições, o caráter didático deste trabalho deve ser ressaltado: a implementação progressiva de problemas clássicos, da equação de Poisson à cavidade *lid-driven*, fornece um roteiro claro para pesquisadores que desejam adotar o Devito em seus próprios problemas de CFD, distinguindo o que é nativo do que requer implementação personalizada.

Este trabalho está organizado em cinco capítulos. O Capítulo 2 apresenta os conceitos básicos do Devito, incluindo sua *API* (*Application Programming Interface* - uma interface de programação de aplicação), definição de domínios, funções e derivadas. O Capítulo 3 aborda a implementação de problemas clássicos com EDPs elípticas, hiperbólicas e parabólicas [2, 3, 8, 21, 24, 27], servindo como validação da metodologia e familiarização com a ferramenta. O Capítulo 4 apresenta uma investigação experimental da integração do Devito com a biblioteca PETSc, comparando desempenho e precisão com a abordagem nativa de *pseudo-timestepping* (pseudo-transiente). O Capítulo 5 detalha a implementação da formulação corrente-vorticidade para as equações de Navier-Stokes, aplicada ao problema da cavidade *lid-driven* [12, 23, 28], incluindo a derivação e implementação das condições de contorno de alta ordem para a vorticidade. Por fim, o Capítulo 6 sintetiza as contribuições do trabalho, discute suas limitações e aponta direções para pesquisas futuras, com ênfase na consolidação da integração Devito-PETSc [15, 16, 17].

Ao longo desta dissertação, trechos de código serão apresentados e comentados com o objetivo de ilustrar a implementação passo a passo das formulações numéricas no Devito, desde a definição de malhas e funções até a construção de operadores e condições de contorno. Essa abordagem didática visa não apenas documentar a metodologia empregada, mas também facilitar a compreensão e a replicação dos experimentos por outros pesquisadores. Além disso, todos os códigos-fonte completos desenvolvidos neste trabalho, incluindo as implementações dos problemas clássicos do Capítulo 3, a formulação corrente-vorticidade para a cavidade *lid-driven* (Capítulo 5) e os *scripts* de integração com PETSc (Capítulo 4), estão disponíveis publicamente em um repositório do GitHub. O acesso pode ser obtido no seguinte endereço: `https://github.com/ikeda-thiago/Mestrado` [14]. Todo o material é fornecido sob licença aberta, incentivando o uso, a adaptação e o avanço da pesquisa em CFD com ferramentas de geração automática de código.

Capítulo 2

Conceitos básicos do Devito

O Devito é um pacote Python para implementar cálculos de estênceis de diferenças finitas otimizados a partir de definições simbólicas de problemas de alto nível. O Devito é construído sobre o SymPy (*Symbolic Python* ou Python Simbólico) e emprega geração automatizada de código e compilação *just-in-time* – uma técnica de programação que traduz o código de um programa em código de máquina (que a CPU entende) sob demanda, durante a execução do programa, em vez de fazer isso tudo antes – para executar núcleos computacionais otimizados em várias plataformas de computação, incluindo CPUs, GPUs e clusters. Projetado principalmente para criar núcleos de propagação de ondas para uso em problemas de inversão sísmica. Algumas de suas características principais:

- Linguagem funcional para expressar operadores de diferenças finitas;
- Mecanismos diretos para ajustar a discretização;
- Ferramentas para definir operadores esparsos (por exemplo, interpolação), operadores lineares tradicionais (como convoluções) e operações de contração entre tensores;
- Suporte integrado para condições de contorno e operadores adjuntos;
- *API* (Interface de Programação de Aplicações) flexível para definir estênceis personalizados, subdomínios, subamostragem e malhas desencontradas;
- Geração de código paralelo altamente otimizado (vetorização SIMD, paralelismo de CPU e GPU via OpenMP e OpenACC, paralelismo multi-nó via *MPI* (Interface de Passagem de Mensagens), bloqueio, transformações simbólicas agressivas para redução de FLOP, etc.);
- *Arrays* NumPy distribuídos sobre decomposições de domínio multi-nó (*MPI*);
- Inspeção e personalização do código gerado;
- *Framework* de *autotuning* para facilitar o ajuste de desempenho;
- Integração suave com pacotes Python populares como NumPy, SymPy, Dask e SciPy, bem como *frameworks* de aprendizado de máquina como TensorFlow e PyTorch.

Instalação do Devito

Nesse trabalho, o Devito será utilizada em um ambiente virtual, *Google Colaboratory*. Para tanto, o Devito está disponível como um `pip package` em PyPI. Após criado o ambiente virtual, basta instalar o Devito:

```
pip install devito
# ...ou para instalar outras dependências adicionais:
# pip install devito[extras,mpi,nvidia,tests]
```

E a seguir são as funções do Devito utilizadas nesse trabalho:

```
from devito import Grid, Function, TimeFunction, Operator, Eq, solve,
    configuration
from examples.cfd import plot_field, init_hat
from devito.logger import warning
from devito import VectorTimeFunction, grad, laplace
```

2.1 *API*

A *API* do Devito foi projetada para fornecer uma interface intuitiva e matemática para a resolução de equações diferenciais parciais. A filosofia da estrutura do Devito segue o princípio da separação entre a formulação matemática e a implementação computacional. O usuário define o problema em notação matemática simbólica, enquanto o *framework* gerencia automaticamente aspectos de baixo nível como discretização, otimização e paralelização.

2.1.1 O domínio no Devito

Apresenta-se uma visão geral da linguagem simbólica Devito, utilizada para expressar e discretizar operadores, em particular equações diferenciais parciais (EDPs).

O principal objetivo é demonstrar como o Devito e sua *API* simbólica baseada em SymPy podem ser usados para resolver equações diferenciais parciais usando o método das diferenças finitas com estênceis altamente otimizados em poucas linhas de Python. É mostrado como estênceis computacionais podem ser derivados diretamente da equação de forma automatizada e como o Devito pode ser usado para gerar e executar, em tempo de execução, o esquema numérico desejado na forma de código C otimizado.

Definindo o Domínio Físico

Antes de começar a criar estênceis de diferenças finitas, é preciso fornecer ao Devito detalhes sobre o domínio computacional dentro do qual se deseja resolver um problema. Para este propósito, cria-se um objeto `Grid` que armazena a extensão física (o tamanho) do domínio e sabe quantos pontos é usado em cada dimensão para discretizar nossos dados.

O `Grid` define a malha computacional onde as equações serão resolvidas. Ele é composto por:

- Extensão física (`extent`): as dimensões reais do domínio em unidades físicas.
- Formato (`shape`): o número de pontos de discretização em cada dimensão.
- Dimensões (`dimensions`): os eixos do espaço (ex.: x, y, t).

Essa discretização é essencial para transformar um problema contínuo (descrito por EDPs) em um problema discreto que pode ser resolvido numericamente. A Figura 2.1 ilustra um exemplo de malha bidimensional.

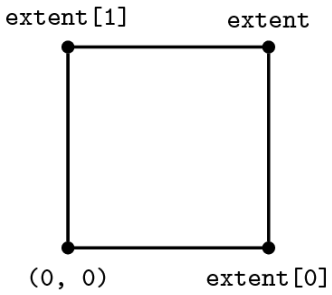


Figura 2.1: Representação de um `Grid` bidimensional com 5 pontos em x e 6 pontos em y. Fonte: [9].

Atualmente, o Devito só fornece a possibilidade de definir domínios retangulares. Entretanto, está sendo definido a possibilidade de se realizar em outros formatos. Edward Caunt está desenvolvendo uma função que permite impor condições de contorno em uma malha não regular, veja mais em `https://github.com/devitocodes/schism`.

Exemplo de Código

No exemplo a seguir, criamos um `Grid` com 5 pontos na direção x e 6 pontos na direção y, cobrindo uma região física quadrada de 1.0×1.0 unidades:

```
grid = Grid(shape=(5, 6), extent=(1., 1.))
grid
```

Saída:

```
Grid[extent=(1.0, 1.0), shape=(5, 6), dimensions=(x, y)]
```

Isso significa que:

- O espaçamento entre pontos na direção x é $\Delta x = 1.0/(5-1) = 0.25$.
- O espaçamento entre pontos na direção y é $\Delta y = 1.0/(6-1) = 0.2$.
- O domínio é representado por uma malha com 30 pontos no total.

Essa malha serve como base para a criação de `Function`'s e `TimeFunction`'s, campos discretos e que evoluem no tempo, respectivamente, que serão usados para representar variáveis como pressão, velocidade, temperatura, entre outras, na resolução de EDPs.

2.1.2 Funções e Derivadas

Para expressar equações na forma simbólica e discretizá-las usando diferenças finitas, o Devito fornece um conjunto de tipos `Function`. Um objeto `Function`:

- Comporta-se como um símbolo `sympy.Function`;
- Gerencia dados associados ao símbolo.

Criando uma Função

```
f = Function(name='f', grid=grid)
f.data
```

Saída:

```
Data([[0., 0., 0., 0., 0., 0.],
      [0., 0., 0., 0., 0., 0.],
      [0., 0., 0., 0., 0., 0.],
      [0., 0., 0., 0., 0., 0.],
      [0., 0., 0., 0., 0., 0.]], dtype=float32)
```

Para problemas dependentes do tempo, o Devito fornece `TimeFunction`:

```
g = TimeFunction(name='g', grid=grid)
g.shape
```

Saída:

```
(2, 5, 6)
```

Por padrão, a ordem temporal de `TimeFunction` é 1, o formato de g é $(2, 5, 6)$, significando que o Devito alocou dois *buffers* para representar $g(t, x, y)$ e $g(t + dt, x, y)$.

Derivadas de Funções Simbólicas

Todas as funções criadas no Devito até o momento atuam como `simpy.Function` objetos, o que significa que é possível formar expressões de derivadas simbólicas a partir delas. O Devito fornece um conjunto de expressões abreviadas (implementadas como propriedades Python) que permitem gerar diferenças finitas na forma simbólica. Por exemplo, a propriedade `f.dx` denota $\frac{\partial}{\partial x} f(x, y)$ - apenas que o Devito já a discretizou com uma expressão de diferenças finitas. Também há um conjunto de expressões abreviadas para:

Tabela 2.1: Abreviações de derivadas de 1^a Ordem no Devito.

Diferenças	Abreviação	Discretização
Progressiva	`f.dxr`	$\frac{f(x+h_x,y)}{h_x} - \frac{f(x,y)}{h_x}$
Regressiva	`f.dxl`	$\frac{f(x,y)}{h_x} - \frac{f(x-h_x,y)}{h_x}$
Centrada	`f.dxc`	$\frac{f(x+h_x,y)}{2h_x} - \frac{f(x-h_x,y)}{2h_x}$

Existe um conjunto similar de expressões para cada dimensão espacial definida em uma malha, por exemplo, `f.dy` e `f.dyl`. Da mesma forma, é possível obter derivadas no tempo de objetos `TimeFunction`. Por exemplo, para calcular a primeira derivada no tempo de g, pode-se simplesmente escrever:

```
g.dt
```

$\frac{\partial}{\partial t} g(t, x, y)$

Também é possível visualizar como o estêncil será gerado baseado na discretização escolhida:

```
g.dt.evaluate
```

$-\frac{g(t,x,y)}{dt} + \frac{g(t+dt,x,y)}{dt}$

Existem também abreviações para expressar o ponto de estêncil seguinte e anterior $g(t + dt, x, y)$ e $g(t - dt, x, y)$:

```
g.forward
```

$g(t + dt, x, y)$

```
g.backward
```

$g(t - dt, x, y)$

Derivadas de 2ª Ordem ou Superiores

No exemplo anterior, estava presente apenas uma combinação de derivadas de primeira ordem na equação governante. Entretanto, derivadas de segunda ordem (ou superiores) frequentemente estão presentes em problemas científicos de interesse, notavelmente em qualquer EDP que modela a difusão. Para gerar derivadas de segunda ordem, é necessário fornecer ao objeto `Function` uma informação adicional: a discretização desejada.

Primeiramente, define-se uma derivada simples de segunda ordem em x, para a qual é necessário especificar um `space_order` de (pelo menos) 2. A forma abreviada para esta derivada segunda é `u.dx2`.

```
u = TimeFunction(name='u', grid=grid, space_order=2)
u.dx2
```

$\frac{\partial^2}{\partial x^2} u(t, x, y)$

```
u.dx2.evaluate
```

$-\frac{2.0u(t,x,y)}{h_x^2} + \frac{u(t,x-h_x,y)}{h_x^2} + \frac{u(t,x+h_x,y)}{h_x^2}$

Para implementar as equações de difusão ou de onda, é necessário calcular o Laplaciano $\nabla^2 u$, que corresponde à soma das segundas derivadas em todas as dimensões espaciais. Para isso, o Devito também fornece uma expressão abreviada, o que significa que não é necessário codificar manualmente a dimensão do problema (2D ou 3D) no código. Para alterar a dimensão do problema, pode-se criar outro objeto `Grid` e utilizá-lo para redefinir as `Function`.

```
grid_3d = Grid(shape=(5, 6, 7), extent=(1., 1., 1.))

u = TimeFunction(name='u', grid=grid_3d, space_order=2)
u
```

$u(t, x, y, z)$

Assim,

```
u = TimeFunction(name='u', grid=grid_3d)
u.laplace
```

é o mesmo que

```
u.dx2 + u.dy2 + u.dz2
```

$\frac{\partial^2}{\partial x^2} u(t, x, y, z) + \frac{\partial^2}{\partial y^2} u(t, x, y, z) + \frac{\partial^2}{\partial z^2} u(t, x, y, z)$

É importante ressaltar que a discretização realizada pelo Devito é baseada no argumento `space_order`. O *framework* realizará uma discretização espacial de mesma ordem que `space_order`, assim como uma discretização temporal de mesma ordem que `time_order` em `TimeFunction`.

Quanto maior o valor de `space_order`, maior é a largura do estêncil utilizado pelo Devito para aproximar as derivadas espaciais. Consequentemente, um número maior de pontos vizinhos participa do cálculo da derivada, aumentando a precisão da aproximação, mas também o custo computacional.

Para uma melhor visualização acerca da discretização espacial empregada pelo Devito, apresenta-se os seguintes estênceis. A Figura 2.2 mostra a discretização para uma função unidimensional, para diferentes `space_order`.

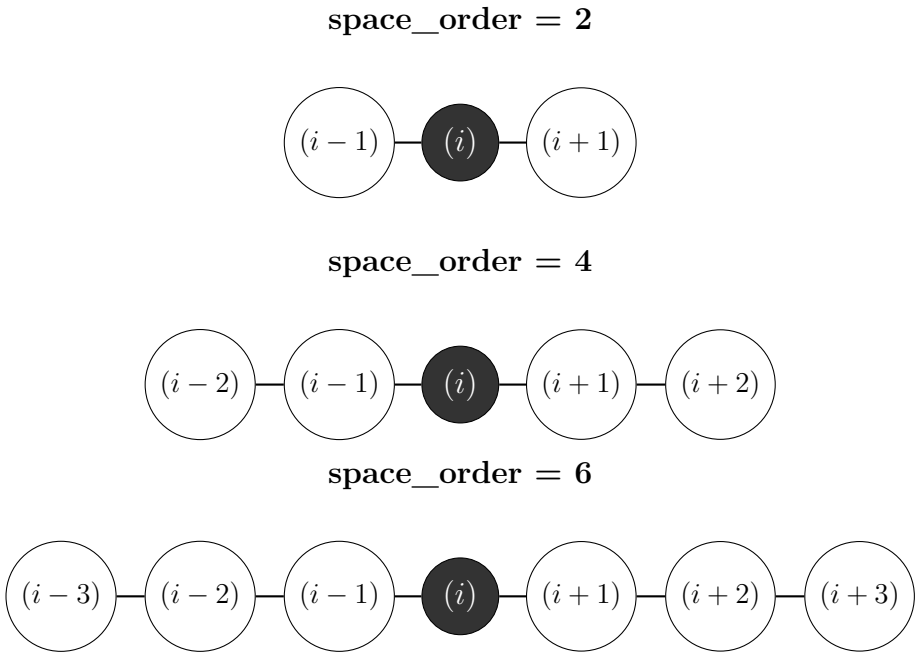


Figura 2.2: Moléculas computacionais unidimensionais para diferentes `space_order`.

A Figura 2.3 ilustra a molécula do Laplaciano bidimensional com `space_order=2`, que envolve o ponto central e seus quatro vizinhos imediatos (norte, sul, leste, oeste).

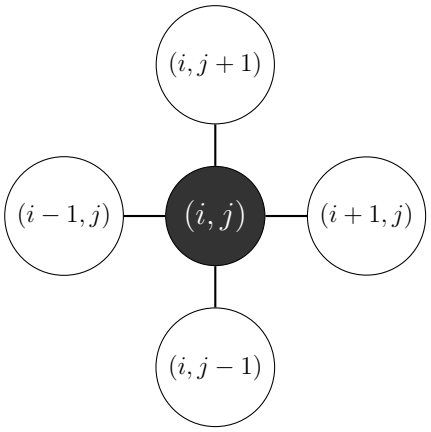


Figura 2.3: Molécula do Laplaciano 2D (`space_order=2`).

Capítulo

3

Problemas Clássicos

O objetivo principal deste capítulo é realizar uma avaliação prática e comparativa do desempenho e da usabilidade do Devito, utilizando como referência diferentes bases de comparação conforme a natureza de cada problema. Para isso, selecionaram-se problemas modelo clássicos da física matemática:

- **Equação elíptica**, representada pela equação de Poisson, cujos resultados numéricos serão comparados com a solução analítica;
- **Equação hiperbólica**, representada por problemas de convecção, cuja implementação em Devito será comparada com uma implementação equivalente usando NumPy;
- **Equação parabólica**, representada por problemas de difusão, que será validada frente à solução analítica conhecida.

Além da comparação de desempenho e precisão, este capítulo tem um caráter didático, buscando demonstrar de forma clara a metodologia de utilização do Devito. Para cada problema, serão apresentados e explicados trechos de código essenciais, ilustrando desde a definição simbólica das variáveis e equações até a execução do operador gerado, permitindo assim uma compreensão prática do seu fluxo de trabalho e sintaxe.

Cada problema será formulado, discretizado pelo método de diferenças finitas e implementado no Devito. A análise focará na precisão da solução, quando comparada à analítica, e, quando aplicável, nos tempos de execução e eficiência computacional em relação à implementação em NumPy. A clareza, concisão e facilidade de desenvolvimento oferecidas pelo Devito também serão avaliadas.

Através dessa investigação, busca-se elucidar as vantagens, desvantagens e cenários de aplicação ideais para o Devito, contribuindo para uma compreensão mais clara de seu potencial como ferramenta para prototipagem rápida e simulação de alto desempenho em problemas científicos.

Ademais, faz-se necessária uma distinção clara entre as funcionalidades nativas do Devito e as implementações originalmente desenvolvidas neste trabalho. Nessa seção, todos os problemas são originais, assim como a condição inicial do problema hiperbólico unidimensional (3.5). Na subseção do PETSc, entretanto, o problema fora fornecido e antes resolvido por Zoe Leiwobitz [15, 17].

3.1 Equação Elíptica

Equações diferenciais parciais elípticas representam uma classe fundamental de problemas na física matemática e engenharia, caracterizando fenômenos em estado estacionário cujos efeitos se propagam instantaneamente em todas as direções. A Equação de Poisson constitui o exemplo paradigmático deste tipo de equação.

Nessa seção, aborda-se a resolução numérica da Equação de Poisson utilizando o Devito, com ênfase na validação da precisão numérica mediante comparação com soluções analíticas. A estratégia de soluções manufaturadas é empregada, permitindo a quantificação precisa dos erros de discretização. Serão considerados problemas bidimensionais e tridimensionais em geometrias regulares, com condições de contorno de Dirichlet.

O método de diferenças finitas é utilizado para a discretização espacial, enquanto a técnica de pseudo-transiente é empregada para contornar a limitação do Devito em resolver sistemas lineares implicitamente, transformando o problema elíptico estacionário em um problema parabólico transiente fictício até a convergência para o estado estacionário.

3.1.1 Problema Bidimensional (2D)

Para o problema bidimensional, considere o domínio $\Omega = [0,1] \times [0,1]$ e condições de Dirichlet homogêneas sobre a fronteira, descrito como:

$$\begin{cases} \dfrac{\partial^2 p}{\partial x^2} + \dfrac{\partial^2 p}{\partial y^2} = f(x,y), \\ p(0,y) = p(x,0) = p(1,y) = p(1,x) = 0, \quad \forall x, y \in \Omega, \end{cases} \tag{3.1}$$

sendo p uma função escalar e $f(x,y)$ o termo fonte, também uma função escalar de duas variáveis. Antes de proceder à implementação numérica no Devito, deriva-se uma solução manufaturada $p(x,y)$ para permitir a validação da solução numérica mediante comparação com a solução analítica.

Solução Manufaturada

Define-se a solução analítica como:

$$p(x,y) = x(x-1)^2 y(y-1)^2. \tag{3.2}$$

Verifica-se que as condições de contorno são satisfeitas, pois $p(0,y) = p(x,0) = p(1,y) = p(1,x) = 0$. Para determinar o termo fonte, calculam-se as derivadas parciais de segunda de $p(x,y)$:

$$\begin{aligned} \frac{\partial p}{\partial x} &= \left(3x^2 - 4x\right)\left(y^3 - 2y^2 + y\right), \\ \frac{\partial^2 p}{\partial x^2} &= (6x-4)\left(y^3 - 2y^2 + y\right), \\ \frac{\partial p}{\partial y} &= \left(3y^2 - 4y\right)\left(x^3 - 2x^2 + x\right), \\ \frac{\partial^2 p}{\partial y^2} &= (6y-4)\left(x^3 - 2x^2 + x\right). \end{aligned}$$

Substituindo na Equação (3.1), obtém-se:

$$\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} = (6x-4)\left(y^3 - 2y^2 + y\right) + (6y-4)\left(x^3 - 2x^2 + x\right) = f(x,y).$$

Implementação no Devito

A implementação no Devito requer a definição inicial dos parâmetros de discretização e a configuração do domínio computacional. Os passos incluem a criação de uma malha regular, a definição simbólica das funções envolvidas e a especificação da equação discretizada por diferenças finitas. Primeiro, os parâmetros iniciais:

```
nx    = 41       # pontos em x
ny    = nx       # pontos em y
xmin  = 0.       # x inicial
xmax  = 1.       #   e final
ymin  = 0.       # y inicial
ymax  = 1.       #   e final


compx = xmax - xmin
compy = ymax - ymin
dx    = (compx) / (nx - 1)       # espaçamento em x
dy    = (compy) / (ny - 1)       # espaçamento em y

x_    = np.linspace(xmin, xmax, nx)
y_    = np.linspace(ymin, ymax, ny)
X, Y  = np.meshgrid(x_,y_,indexing='ij')

sigma = 0.1
dt    = (sigma * dx * dy)        # espaçamento temporal
tstop = 1                        # tempo final
nt    = round(tstop / dt)        # pontos no tempo

grid  = Grid(shape=(nx, ny), extent=(compx, compy))  # malha
x, y  = grid.dimensions
t     = grid.stepping_dim
```

Perceba que a definição de uma quantidade de pontos e de um espaçamento no tempo, bem como de um tempo final, é necessária devido a uma limitação inerente ao Devito: sua incapacidade de resolver sistemas lineares oriundos de formulações implícitas diretamente. A equação de Poisson, quando discretizada por diferenças finitas, assume a forma clássica:

$$\frac{P_{i-1,j} - 2P_{i,j} + P_{i+1,j}}{\Delta x^2} + \frac{P_{i,j-1} - 2P_{i,j} + P_{i,j+1}}{\Delta y^2} = f(x_i, y_j),$$

em que $P_{i,j} \approx p(x_i, y_j)$ representa a solução numérica aproximada e $\Delta x = \Delta y$ denota o espaçamento da malha espacial, que é colocalizada. Tradicionalmente, este problema discretizado resulta em um sistema linear de equações, resolvido por métodos diretos ou iterativos. Contudo, como o Devito não possui um solver linear embutido para tais sistemas, uma abordagem alternativa deve ser empregada.

Para contornar essa limitação, utiliza-se uma técnica conhecida como *pseudo-timesteping* (ou pseudo-transiente). Essa estratégia consiste em transformar o problema elíptico estacionário em um problema parabólico fictício, introduzindo um termo transiente. Dessa forma, a equação é reescrita como:

$$P_{i,j}^{n+1} = \frac{\left(P_{i-1,j}^{n} + P_{i+1,j}^{n}\right)\Delta y^2 + \left(P_{i,j-1}^{n} + P_{i,j+1}^{n}\right)\Delta x^2 - f_{i,j}^{n}\Delta x^2 \Delta y^2}{2\left(\Delta x^2 + \Delta y^2\right)}.$$

Agora, definem-se as funções para p e para o termo fonte f. A variável p é declarada como uma `TimeFunction`, o que permite evolução temporal, necessária para a técnica de pseudo-transiente. Logo, o domínio torna-se $[0, t_f] \times \Omega$, em que t_f é o tempo final.

O termo fonte f, por sua vez, é definido como uma `Function`, pois permanece constante ao longo do tempo artificial e não requer atualização a cada iteração. Essa abordagem otimiza o cálculo, evitando recálculos desnecessários do termo fonte.

Após a definição das equações simbólicas e das condições de contorno, cria-se um objeto `Operator`. O `Operator` é o componente responsável por transformar as expressões simbólicas do Devito em um código executável. Para isso, ele recebe uma lista de equações ('eq'), realiza automaticamente a análise das dependências entre elas, aplica otimizações de compilação e gera um único código em linguagem C, compilado em tempo de execução (Just-In-Time).

```
configuration['log-level'] = 'ERROR'  # Silencia o registro de desempho de
    execução

eq         = Eq(p.laplace, f)   # p.laplace é o mesmo que p.dx2 + p.dy2
stencil    = solve(eq, p)
eq_stencil = Eq(p.forward, stencil)

# Condições de contorno
bc  = [Eq(p[t + 1, x, 0], 0.)]        # p(x,0) = 0
bc += [Eq(p[t + 1, x, ny-1], 0.)]     # p(x,1) = 0
bc += [Eq(p[t + 1, 0, y], 0.)]        # p(0,y) = 0
bc += [Eq(p[t + 1, nx-1, y], 0.)]     # p(1,y) = 0

op = Operator([eq_stencil] + bc)
op(time=nt)
# (usar '%time op(time=nt)' calcula o tempo de execução)
plot_field(p.data[0], xmin=xmin, ymin=ymin, xmax=xmax, ymax=ymax, view
    =(30,225))
# plot_field é uma função do Devito para plotagem dos resultados
```

O construtor `Operator` recebe como argumento uma lista contendo as equações simbólicas e as condições de contorno. A ordem dessa lista é preservada durante a geração do código, respeitando as dependências existentes entre as expressões. Após sua construção, a chamada `op(time=nt)` executa o código compilado, realizando `nt` iterações temporais da discretização.

Perceba que 'eq', 'stencil' e 'eq_stencil' representam:

- Equação discretizada ('eq'):

$$\left(-\frac{2P(t,x,y)}{\Delta x^2}+\frac{P(t,x-\Delta x,y)}{\Delta x^2}+\frac{P(t,x+\Delta x,y)}{\Delta x^2}\right)+$$
$$+\left(-\frac{2P(t,x,y)}{\Delta y^2}+\frac{P(t,x,y-\Delta y)}{\Delta y^2}+\frac{P(t,x,y+\Delta y)}{\Delta y^2}\right)=f(x,y)$$

- Isolamento do termo forward ('stencil'):

$$-\frac{0.5\left(\Delta x^2\Delta y^2 f(x,y)-\Delta x^2P(t,x,y-\Delta y)-\Delta x^2P(t,x,y+\Delta y)\right)}{\Delta x^2+\Delta y^2}-$$
$$-\frac{0.5\left(\Delta y^2P(t,x-\Delta x,y)-\Delta y^2P(t,x+\Delta x,y)\right)}{\Delta x^2+\Delta y^2}$$

- Equação final ('eq_stencil'):

$$P(t+\Delta t,x,y)=-\frac{0.5\left(\Delta x^2\Delta y^2 f(x,y)-\Delta x^2P(t,x,y-\Delta y)-\Delta x^2P(t,x,y+\Delta y)\right)}{\Delta x^2+\Delta y^2}-$$
$$-\frac{0.5\left(\Delta y^2P(t,x-\Delta x,y)-\Delta y^2P(t,x+\Delta x,y)\right)}{\Delta x^2+\Delta y^2}$$

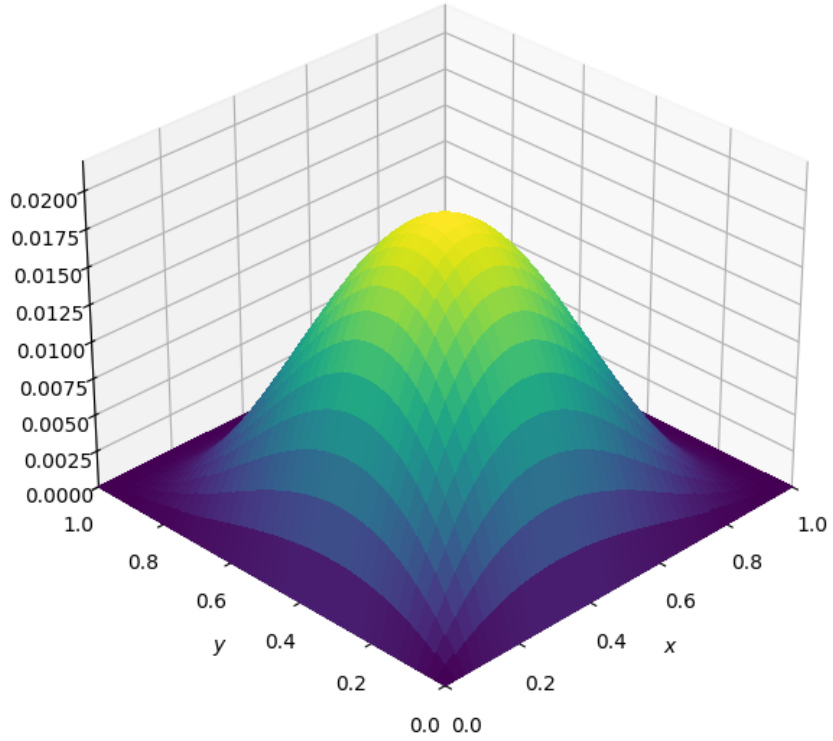


Figura 3.1: Resultado numérico para $nx = ny = 41$.

Resultados e Análises

Para os resultados numéricos, considera-se que há a mesma quantidade de pontos na malha nas direções x e y, isto é, $nx = ny$. A seguir é apresentado alguns resultados numéricos mostrando o tempo de execução e o erro relativo para diferentes malhas, além da comparação visual entre a solução numérica e a analítica através do seguinte *plot*:

```
def plot_3d_comparison(x, y, numerical_solution, analytical_solution):
    """
    Plota comparação 3D entre solução numérica (superfície) e analítica (
    pontos)

    Parâmetros:
    x : array 1D - coordenadas x da malha
    y : array 1D - coordenadas y da malha
    numerical_solution : array 2D - solução numérica
    analytical_solution : array 2D - solução analítica
    """
    # Cria malha para plotagem 3D
    X, Y = np.meshgrid(x, y, indexing='ij')

    fig = plt.figure(figsize=(14, 8))
    ax = fig.add_subplot(111, projection='3d')

    # Plota superfície da solução numérica
    surf = ax.plot_surface(X, Y, analytical_solution.T,
                           cmap='viridis', alpha=0.7,
                           label='Solução Analítica')

    # Plota pontos da solução analítica
```

```
    scatter = ax.scatter(X[::2,::2], Y[::2,::2], numerical_solution.T
    [::2,::2],
                         color='red', s=50, depthshade=True,
                         label='Solução Numérica')

    # Configurações do gráfico
    ax.set_xlabel(r'x', fontsize=12)
    ax.set_ylabel(r'y', fontsize=12)
    ax.set_zlabel(r'p(x,y)', fontsize=12)

    # Criando legendas
    surf._edgecolors2d = surf._edgecolor3d
    surf._facecolors2d = surf._facecolor3d
    ax.legend()

    # Ajusta a visualização
    ax.view_init(elev=20, azim=60)
    plt.tight_layout()
    plt.show()
```

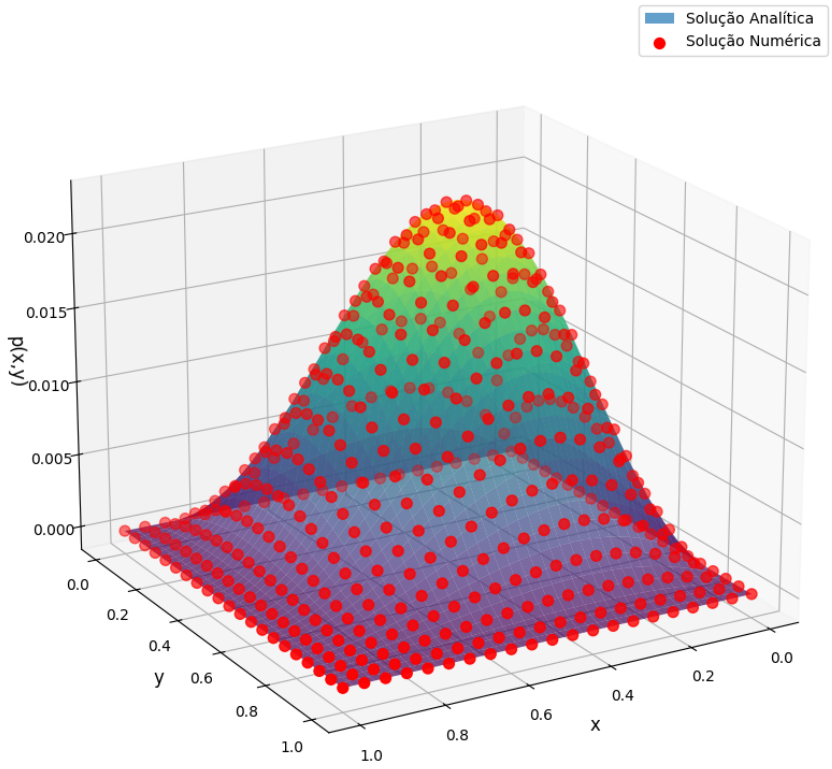


Figura 3.2: Comparação entre as soluções numérica e analítica para $nx = ny = 41$.

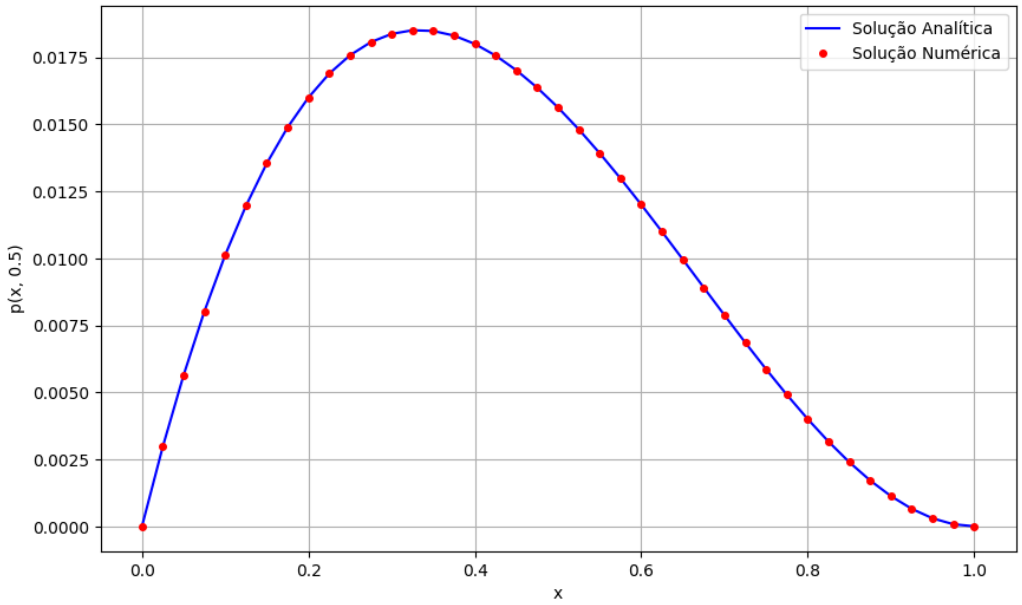


Figura 3.3: Comparação entre as soluções numérica e analítica para $nx = ny = 41$ em um corte central ($y = 0.5$).

Qualitativamente, observa-se que, mesmo em uma malha com baixa resolução, a solução numérica apresenta uma concordância visual significativa com a solução analítica, conforme ilustrado nas Figuras 3.2 e 3.3.

Tabela 3.1: Resultados da simulação da equação de Poisson 2D para diferentes tamanhos de malha.

nx	Tempo	Erro Relativo
21	35.4 ms	$4.55 \cdot 10^{-6}$
41	40.7 ms	$2.31 \cdot 10^{-5}$
81	152 ms	$2.60 \cdot 10^{-5}$
161	1.57 s	$1.28 \cdot 10^{-4}$

Nota-se que o tempo de execução mantém-se relativamente baixo mesmo para malhas mais refinadas. Contudo, o erro relativo ϵ_r, calculado conforme a expressão

$$\epsilon_r = \frac{\sqrt{\sum_{i,j} \left(p(x_i, y_j) - P_{i,j}\right)^2}}{\sqrt{\sum_{i,j} \left(p(x_i, y_j)\right)^2}}, \tag{3.3}$$

apresenta um aumento com o refinamento da malha. Este comportamento é atípico, uma vez que se esperaria uma redução do erro com o aumento da resolução espacial.

3.1.2 Problema Tridimensional (3D)

Para o problema tridimensional, considera-se uma extensão natural do caso bidimensional, com domínio $\Omega = [0, 1] \times [0, 1] \times [0, 1]$ e condições de contorno de Dirichlet homogêneas aplicadas em todas as faces do domínio. O problema é formulado como:

$$\begin{cases} \dfrac{\partial^2 p}{\partial x^2} + \dfrac{\partial^2 p}{\partial y^2} + \dfrac{\partial^2 p}{\partial z^2} = f(x,y,z), \\ p(0,y,z) = p(1,y,z) = 0, \quad \forall (y,z) \in [0,1]^2, \\ p(x,0,z) = p(x,1,z) = 0, \quad \forall (x,z) \in [0,1]^2, \\ p(x,y,0) = p(x,y,1) = 0, \quad \forall (x,y) \in [0,1]^2. \end{cases} \tag{3.4}$$

Segue-se a mesma metodologia de soluções manufaturadas aplicada no problema bidimensional. Define-se a solução analítica como:

$$p(x,y,z) = (x^2 - x)(y^2 - y)(z^2 - z). \tag{3.5}$$

Verifica-se que esta função satisfaz as condições de contorno homogêneas em todo o contorno do domínio cúbico. O termo fonte $f(x,y,z)$ correspondente obtido é:

$$f(x,y,z) = 2\left[(y^2 - y)(z^2 - z) + (x^2 - x)(z^2 - z) + (x^2 - x)(y^2 - y)\right].$$

Implementação no Devito

A implementação no Devito segue a estrutura do problema bidimensional, com a adição da dimensão espacial z. A malha tridimensional é definida com dimensões (nx, ny, nz), e as condições de contorno são especificadas para todas as faces do domínio.

```
grid       = Grid(shape=(nx, ny, nz), extent=(compx, compy, compz))
x, y, z    = grid.dimensions
t          = grid.stepping_dim
p          = TimeFunction(name='p', grid=grid, space_order=2)
p.data[:] = 0.

f          = Function(name='f', shape=(nx, ny, nz), dimensions=(x, y, z),
    grid=grid)
f_data     = 2*((Y**2 - Y)*(Z**2 - Z) + (X**2 - X)*(Z**2 - Z) + (X**2 - X)*(
    Y**2 - Y))
f.data[:] = f_data.reshape((nx, ny, nz))

# Condições de contorno
bc  = [Eq(p[t+1, x, y, 0], 0.)]      # z = 0
bc += [Eq(p[t+1, x, y, nz-1], 0.)]   # z = 1

bc += [Eq(p[t+1, x, 0, z], 0.)]      # y = 0
bc += [Eq(p[t+1, x, ny-1, z], 0.)]   # y = 1

bc += [Eq(p[t+1, 0, y, z], 0.)]      # x = 0
bc += [Eq(p[t+1, nx-1, y, z], 0.)]   # x = 1
```

Resultados e Análises

Para a análise numérica, considerou-se malhas regulares com igual número de pontos nas direções x, y e z. A Figura 3.4 apresenta um corte da solução em z, demonstrando uma excelente concordância qualitativa entre as soluções numérica e analítica, mesmo para uma malha moderada de 41 pontos.

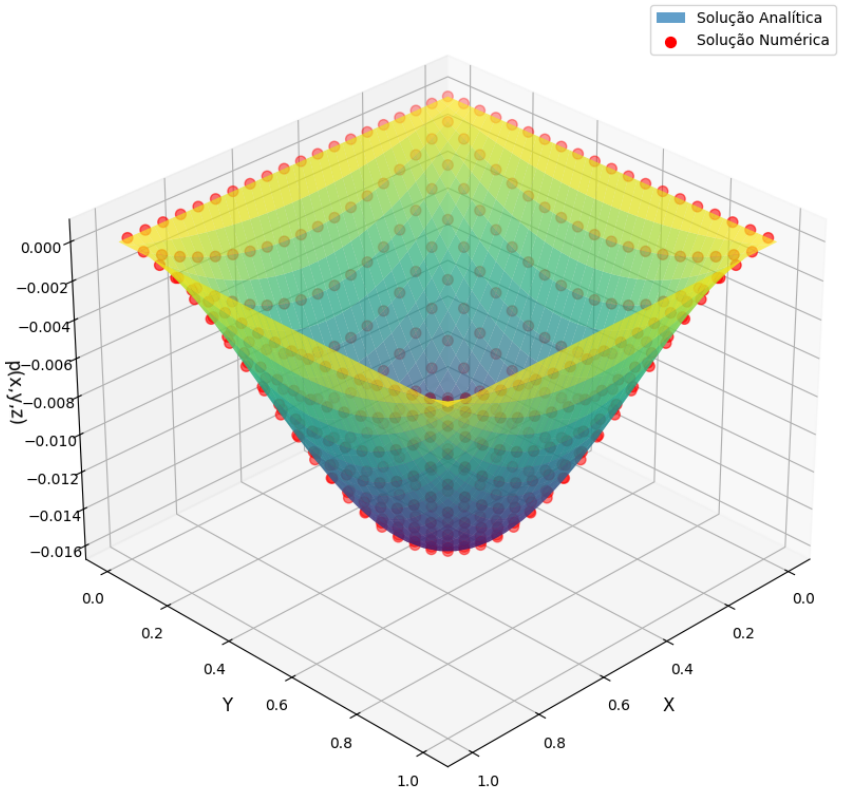


Figura 3.4: Comparação entre as soluções numérica e analítica em um corte em $z = 0.5$, utilizando malha regular de 41^3 pontos.

A Tabela 3.2 apresenta os resultados quantitativos para diferentes refinamentos de malha. Nota-se que o comportamento do erro relativo segue o padrão observado no caso bidimensional, com aumento progressivo conforme o refinamento da malha. Contudo, o aspecto mais significativo é o crescimento exponencial do tempo de computação.

Tabela 3.2: Resultados da simulação da equação de Poisson 3D para diferentes tamanhos de malha.

nx	Tempo	Erro Relativo
11	986 ms	$2.54 \cdot 10^{-6}$
21	1.42 s	$3.29 \cdot 10^{-6}$
41	28.9 s	$5.21 \cdot 10^{-6}$
81	35 min	$4.84 \cdot 10^{-5}$

A análise dos resultados revela dois aspectos importantes. Primeiramente, confirma-se o padrão observado no problema $2D$, no qual o erro relativo aumenta com o refinamento da malha. Em segundo lugar, evidencia-se o impacto computacional significativo da terceira dimensão, com o tempo de execução aumentando drasticamente para malhas mais refinadas.

Vale destacar que o Devito possui capacidades nativas de execução paralela, permitindo a distribuição do cálculo entre múltiplos núcleos de processamento ou mesmo em *clusters* computacionais. Essa funcionalidade está integrada ao *framework* e pode ser ativada através de configurações simples, sem a necessidade de modificações significativas no código. Para problemas que demandem malhas realmente extensas ou simulações de grande escala, essa capacidade de paralelização pode mitigar substancialmente o im-

pacto do aumento do tempo computacional, tornando viável o tratamento de problemas complexos que seriam proibitivos em ambientes sequenciais.

3.2 Equação Hiperbólica

Nesta seção, o enfoque é direcionado à aplicação do Devito na resolução de equações diferenciais parciais (EDPs) de natureza hiperbólica. Tais equações caracterizam-se pela propagação de informações ao longo de características, sendo particularmente relevantes para modelar fenômenos de transporte e propagação de ondas. Um aspecto fundamental das EDPs hiperbólicas é sua capacidade de desenvolver e propagar descontinuidades, mesmo a partir de condições iniciais suaves.

O principal desafio numérico nesse contexto reside no controle de erros de discretização, que se manifestam principalmente como:

- **dissipação numérica:** amortecimento não físico dos gradientes da solução;
- **dispersão numérica:** surgimento de oscilações espúrias em regiões de grandes gradientes.

Serão analisados dois casos distintos: um problema unidimensional e um problema bidimensional. A abordagem numérica adotará o esquema *upwind* de primeira ordem implementado no Devito, com validação por meio de comparações com implementações equivalentes em NumPy e com a solução analítica.

3.2.1 Problema Unidimensional (1D)

O problema de convecção unidimensional representa um caso fundamental para a análise de esquemas numéricos aplicados a equações hiperbólicas. A equação de advecção linear, dada por:

$$u_t + c_x u_x = 0, \tag{3.6}$$

na qual c_x representa a velocidade positiva de propagação na direção x e $u(t, x)$ representa o transporte de uma propriedade em um domínio $\Omega = [0, 3]$ progredindo no tempo $[0, t_f]$ até um tempo final t_f. No entanto, essa aparente simplicidade esconde desafios numéricos significativos, particularmente relacionados aos fenômenos de dissipação e dispersão introduzidos pela discretização. Para uma análise abrangente, será considerada a seguinte condição inicial:

$$u(0, x) = \begin{cases} 2, & 0 \leq x < 0.5 \\ 1, & 0.5 \leq x < 1.0 \\ \frac{1}{2}, & 1.0 \leq x < 1.5 \\ -8x^2 + 32x - 30, & 1.5 \leq x < 2.5 \\ 0, & \text{caso contrário,} \end{cases} \tag{3.7}$$

caracterizada por descontinuidades abruptas (degraus) e por uma transição suave (região parabólica), conforme mostra a Figura 3.5:

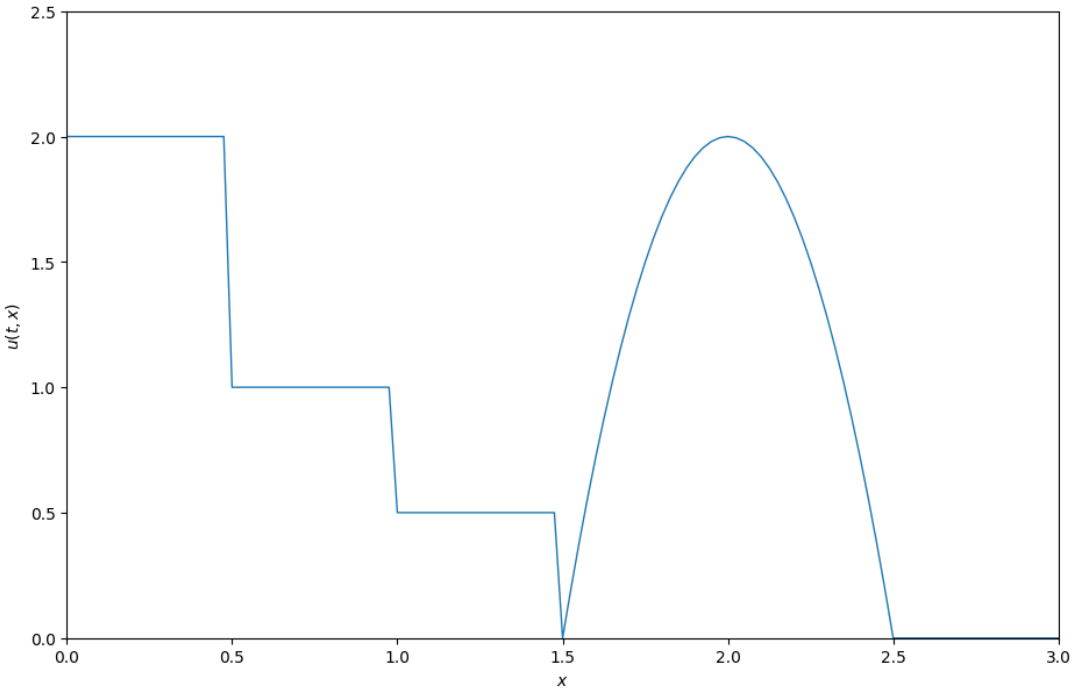


Figura 3.5: Condição inicial com preenchimento de cores.

Objetiva-se avaliar o comportamento dos esquemas numéricos na preservação da forma do problema.

Solução Analítica

Para obter a solução analítica da equação (3.6), aplica-se o método das características. A solução geral desta equação de transporte linear é dada por:

$$u(t, x) = f(x - ct),$$

em que f é uma função determinada pela condição inicial $u(0, x) = f(x)$.

Considerando a condição inicial (3.7) composta por segmentos constantes e uma região suave, a solução para $t > 0$ mantém este perfil deslocado com velocidade c_x:

$$u(t,x) = \begin{cases} 2, & c_x t + 0 \leq x < c_x t + 0.5 \\ 1, & c_x t + 0.5 \leq x < c_x t + 1.0 \\ \frac{1}{2}, & c_x t + 1.0 \leq x < c_x t + 1.5 \\ -8(x - c_x t)^2 + 32(x - c_x t) - 30, & c_x t + 1.5 \leq x < c_x t + 2.5 \\ 0, & \text{caso contrário.} \end{cases}$$

Essa solução representa o transporte puro do perfil inicial ao longo do domínio, sem alteração na forma. A região parabólica mantém sua forma quadrática original durante a propagação, caracterizando uma transição suave entre os valores constantes adjacentes. Para a simulação, consideram-se as condições de contorno do tipo Dirichlet iguais a zero.

Implementação no Devito

Primeiramente, definem-se os parâmetros iniciais:

```
nx     = 31              # quantidade de pontos na direção x
L      = 3.              # ponto final
```

```
tstop = 0.5            # tempo final
cx    = 0.5            # velocidade na direção x
dx    = L / (nx - 1)   # espaçamento temporal

sigma = 0.1
dt    = (sigma * dx)/cx  # espaçamento temporal - condição CFL
nt    = tstop / dt
nti   = round(nt)        # quantidade de pontos no tempo

grid  = Grid(shape=(nx), extent=(L))
x     = grid.dimensions
t     = grid.stepping_dim
```

Agora, a condição inicial (3.7):

```
def uni_ini_cond(field, dx, value=2., bgvalue=0., n=60):
    """Define condição inicial com degraus e transição suave em um array:

    u(0.0 <= x < 0.5) = 2
    u(0.5 <= x < 1.0) = 1
    u(1.0 <= x < 1.5) = 0.5
    u(1.5 <= x < 2.5) = parábola suave
    caso contrário = 0

    Parâmetros:
    field : array_like
        Dados do campo para inicializar.
    dx : float
        Espaçamento na dimensão x.
    value : float, opcional
        Valor da parte superior da função. Padrão = 2.
    bgvalue : float, opcional
        Valor de fundo. Padrão = 0.
    n : int, opcional
        Número de pontos no domínio. Padrão = 60.
    """
    coords = np.linspace(0, L, n+1)

    # Toda a demais região
    field[:] = bgvalue

    # Primeiro degrau: 0.0 <= x < 0.5
    field[int(.0 / dx):int(0.5 / dx)] = value

    # Segundo degrau: 0.5 <= x < 1.0
    field[int(.5 / dx):int(1 / dx)] = value - 1.

    # Terceiro degrau: 1.0 <= x < 1.5
    field[int(1 / dx):int(1.5 / dx)] = value  - 1.5

    # Transição suave: 1.5 <= x < 2.5 (parábola)
    x_vals = coords[int(1.5 / dx):int(2.5 / dx)]
    field[int(1.5 / dx):int(2.5 / dx)] = -8*(x_vals**2) +32*x_vals - 30
```

Seguindo uma estrutura similar à técnica de pseudo-transiente, utiliza-se o método de Euler Explícito para discretização temporal. O código para definição das funções, condições de contorno, aplicação da condição inicial e construção do operador é apresentado a seguir:

```
u = TimeFunction(name='u', grid=grid)    # u(t,x)
uni_init_hat_especial(field=u.data[0], dx=dx, grid = grid)
```

```
3
4 eq = Eq(u.dt + cx*u.dxl)
5 stencil = solve(eq, u.forward)
6 eq_stencil = Eq(u.forward, stencil)
7
8 bc = [Eq(u[t+1,0], 0.)]          # u(t,0) = 0
9 bc += [Eq(u[t+1,-1], 0.)]        # u(t,L) = 0
10
11 op = Operator([eq_stencil] + bc)
12 %time op(time = nti, dt=dt)
```

Note que, como c_x é positivo, utiliza-se `.dxl` para realizar a diferença regressiva. Caso c_x fosse negativo, seria necessário alterar para `.dxr`.

Como o método de discretização é o *Upwind*, define-se o argumento `space_order=1`, pois o método é de primeira ordem. Por padrão, `TimeFunction` e `Function` possuem `space_order=1`.

A diferença fundamental em relação ao método da seção anterior reside na formulação da equação. A sequência de comandos:

```
1 eq = Eq(u.dt + cx*u.dxl)
2 stencil = solve(eq, u.forward)
3 eq_stencil = Eq(u.forward, stencil)
```

representa a discretização por diferenças finitas da equação (3.6), tal que $U(t,x) \approx u(t,x)$ é a solução numérica, através dos seguintes passos:

- Equação discretizada ('eq'):

$$\left(-\frac{U(t,x)}{\Delta t} + \frac{U(t+\Delta t,x)}{\Delta t}\right) + c_x\left(\frac{U(t,x)}{\Delta x} - \frac{U(t,x-\Delta x)}{\Delta x}\right) = 0.$$

- Isolamento do termo forward ('stencil'):

$$\Delta t\left(-c_x\left(\frac{U(t,x)}{\Delta x} - \frac{U(t,x-\Delta x)}{\Delta x}\right) + \frac{U(t,x)}{\Delta t}\right).$$

- Equação final ('eq_stencil'):

$$U(t+\Delta t,x) = \Delta t\left(-c_x\left(\frac{U(t,x)}{\Delta x} - \frac{U(t,x-\Delta x)}{\Delta x}\right) + \frac{U(t,x)}{\Delta t}\right).$$

Essa formulação corresponde ao esquema *upwind* de primeira ordem, na qual a derivada espacial é aproximada pela diferença regressiva, garantindo estabilidade numérica quando satisfeita a condição CFL (Courant-Friedrichs-Lewy) $\left|c_x \frac{\Delta t}{\Delta x}\right| \leq 1$.

Resultados e Análises

Para a resolução numérica, foi considerado $\Delta t = 0.1 \cdot \Delta x$, tempo final de $0.6s$ e $c_x = 0.5\ m/s$.

Tabela 3.3: Resultados da simulação para diferentes tamanhos de malha.

	NumPy		Devito	
nx	Tempo	Erro Relativo	Tempo	Erro Relativo
31	0.334 ms	0.2639	25.5 ms	0.2594
61	0.489 ms	0.1998	28.7 ms	0.1971
121	0.669 ms	0.1586	33.6 ms	0.1574
241	1.51 ms	0.1294	45.5 ms	0.1288
240001	261 s	0.0223	21 s	0.0249

Para malhas de menor dimensão ($nx \leq 241$), a implementação NumPy demonstra tempos de execução substancialmente inferiores, variando de 0.334 ms a 1.51 ms, enquanto o Devito requer entre 25.5 ms e 45.5 ms para as mesmas configurações. Essa diferença inicial pode ser atribuída à geração de código otimizado realizados pelo Devito, cujo custo torna-se mais perceptível em problemas de pequena escala.

Contudo, esse cenário modifica-se radicalmente quando consideramos malhas mais extensas. Para $nx = 240001$, observa-se uma inversão no desempenho: o NumPy requer 261 segundos para completar a simulação, enquanto o Devito executa a mesma tarefa em apenas 21 segundos, representando uma aceleração de aproximadamente 12 vezes. Esse resultado evidencia a eficácia das estratégias de otimização automática oferecidas pelo Devito, as quais tornam-se progressivamente mais vantajosas à medida que a complexidade computacional do problema aumenta.

No que concerne à precisão numérica, ambos os métodos apresentam erros relativos bastante similares para cada tamanho de malha, com diferenças inferiores a 2% em todos os casos analisados, exceto o último. Essa concordância valida que ambas as implementações reproduzem consistentemente o mesmo esquema numérico *upwind* de primeira ordem. Adicionalmente, verifica-se que o erro relativo diminui monotonicamente com o refinamento da malha, comportamento este que está em conformidade com a literatura.

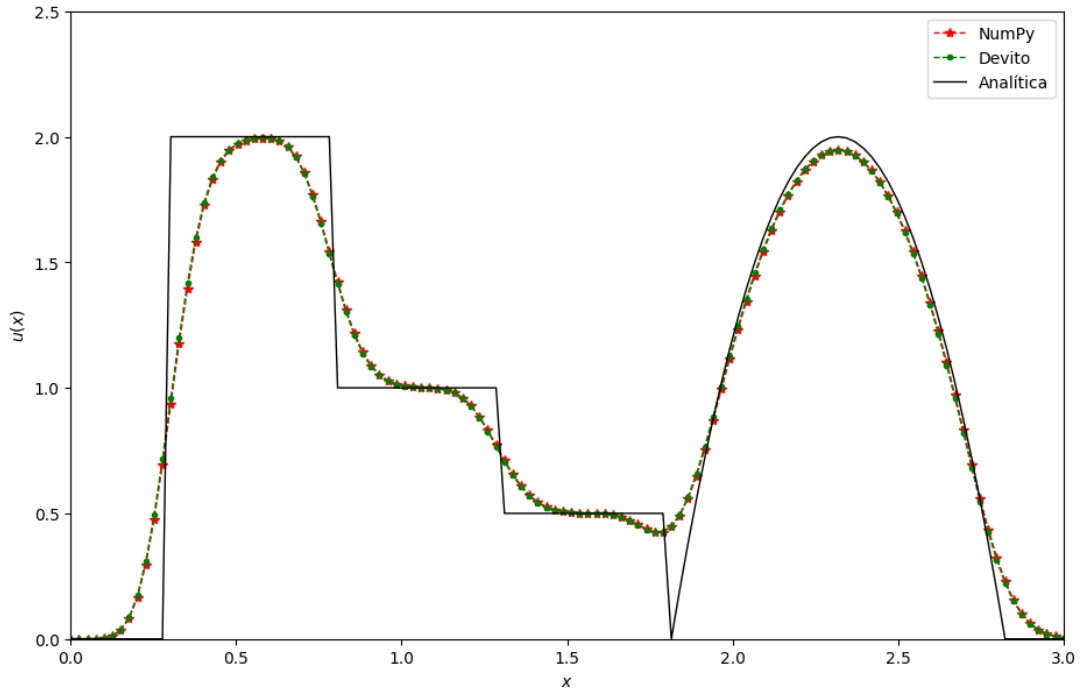


Figura 3.6: Soluções numéricas e analítica para $nx = 121$.

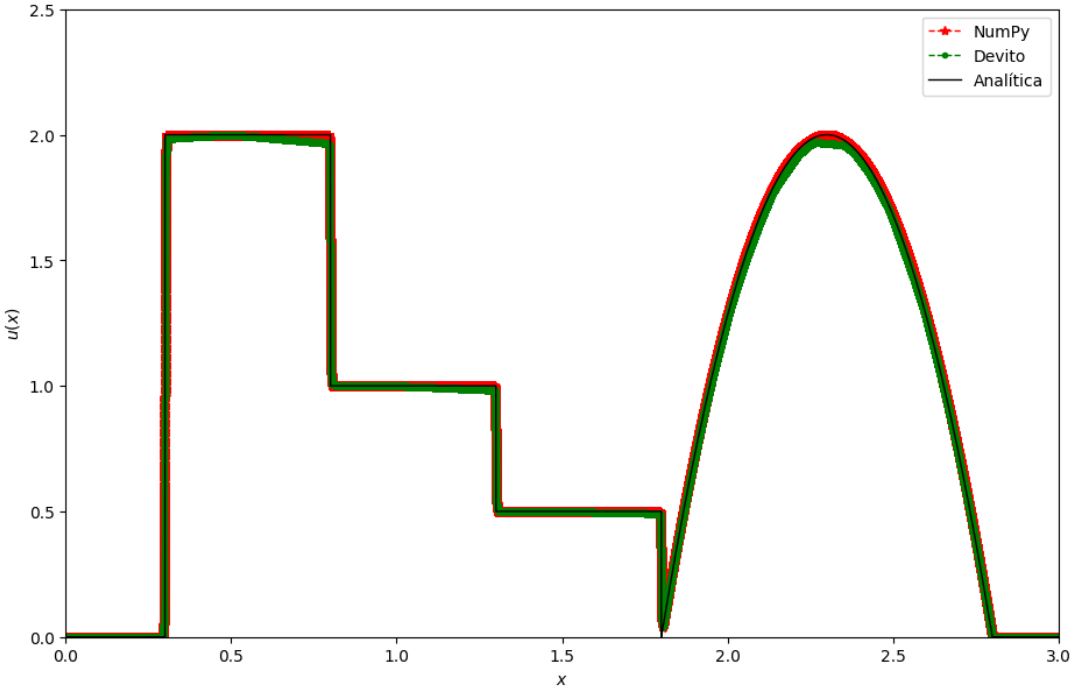


Figura 3.7: Soluções numéricas e analítica para $nx = 240001$.

A análise qualitativa das soluções, no entanto, revela nuances importantes. Para uma malha com $nx = 121$, visualizada na Figura 3.6, observa-se a característica suavização da solução nas regiões de descontinuidade, enquanto a região parabólica é adequadamente capturada por ambas as implementações. Contudo, para a malha altamente refinada com $nx = 240001$, apresentada na Figura 3.7, , a solução do Devito apresenta uma aproximação visualmente inferior em certas regiões, como no vértice da parábola. Esta discrepância qualitativa pode ser vista na métrica do erro relativo, em que NumPy forneceu um erro relativo menor do que o Devito, com um tempo de simulação maior.

3.2.2 Problema Bidimensional (2D)

A extensão do problema de convecção para duas dimensões espaciais considera a equação:

$$u_t + c_x u_x + c_y u_y = 0, \tag{3.8}$$

que descreve o transporte de uma propriedade $u(t, x, y)$ com velocidades c_x e c_y nas direções x e y, respectivamente, em um domínio $\Omega = [0, 2] \times [0, 2]$ com condições de contorno de Dirichlet iguais a zero. Para este estudo, será utilizada uma condição inicial fornecida pela própria biblioteca Devito, consistindo em uma região de descontinuidade:

$$u(0, x, y) = \begin{cases} 2, & 0.5 \leq x, y < 1 \\ 0, & \text{caso contrário.} \end{cases} \tag{3.9}$$

Uma funcionalidade avançada do Devito explorada nesta seção é o mecanismo de armazenamento temporal (`save`) integrado às `TimeFunctions`. Por padrão (`save=None`), o Devito utiliza *buffers* alternados para escrita cíclica, no qual o tamanho do *buffer* temporal é determinado por `time_order + 1`. Esse modo é eficiente em termos de memória, mas não preserva todos os passos temporais intermediários.

Quando há necessidade de acessar a solução em múltiplos instantes de tempo, como para geração de animações ou análise temporal detalhada, deve-se especificar explicitamente um valor inteiro para `save`. Essa configuração desativa os *buffers* alternados e armazena todas as soluções nos instantes especificados, permitindo a reconstrução completa da evolução temporal do fenômeno.

Implementação no Devito

A implementação no Devito inicia-se com a definição dos parâmetros de discretização e simulação:

```
nx = 81
ny = nx
cx = 1.
cy = 1.
L  = 2

dx = L / (nx - 1)
dy = L / (ny - 1)
sigma = .2

dtx = dx/cx  # CFL para x
dty = dy/cy  # CFL para y
dt  = sigma * min(dtx,dty)

tstop = 0.5
nt = round(tstop / dt)
```

Em seguida, define-se a malha computacional e as funções necessárias, aplicando a condição inicial fornecida pela biblioteca Devito:

```
grid = Grid(shape=(nx, ny), extent=(L, L))
x,y  = grid.dimensions
t    = grid.stepping_dim

save = round(nt)
u    = TimeFunction(name='u', grid=grid, save = save)
init_hat(field=u.data[0], dx=dx, dy=dy, bgvalue=0.)  # função do Devito
plot_field(u.data[0]) # função do Devito. Visualização da condição inicial
```

A formulação do problema é completada com a definição da equação de convecção e sua discretização:

```
eq         = Eq(u.dt + cx*u.dxl + cy*u.dyl)
stencil    = solve(eq, u.forward)
eq_stencil = Eq(u.forward, stencil)
```

Finalmente, estabelecem-se as condições de contorno de Dirichlet homogêneas em todas as fronteiras e executa-se a simulação através do operador:

```
bc = [Eq(u[t+1,x,0],0)]
bc += [Eq(u[t+1,x,ny-1],0)]
bc += [Eq(u[t+1,0,y],0)]
bc += [Eq(u[t+1,nx-1,y],0)]

op = Operator([eq_stencil] + bc)
%time op(dt=dt)
plot_field(u.data[save-1])  # visualização da solução final
```

Resultados e Análises

Considere $c_x = c_y = 1$ (m/s) e tempo final de 0.5 segundos. A análise do problema de convecção bidimensional considerou tanto o desempenho computacional quanto o comportamento da solução numérica. A Tabela 3.4 apresenta os tempos de execução para diferentes tamanhos de malha e configurações do parâmetro `save`, que controla o armazenamento de soluções em passos temporais (nt) intermediários.

Tabela 3.4: Tempos de execução para diferentes configurações de malha e parâmetros de armazenamento temporal.

nx	`save`	Tempo
	None	40.4 ms
81	$nt/2$	42.7 ms
	nt	43.3 ms
	None	46.3 ms
161	$nt/2$	53.1 ms
	nt	65.4 ms
	None	171 ms
641	$nt/2$	184 ms
	nt	307 ms

Os resultados demonstram que o custo computacional aumenta significativamente com o refinamento da malha e com a quantidade de passos temporais armazenados. A configuração `save=None`, que utiliza *buffers* cíclicos, apresenta o melhor desempenho, enquanto o armazenamento completo de todos os passos temporais (`save=nt`) implica em um aumento de custo considerável, particularmente evidente em malhas mais refinadas.

A evolução temporal da solução é ilustrada nas Figuras 3.8-3.10, que mostram o transporte de u através do domínio computacional e a dissipaçã da solução causada pelo método numérico.

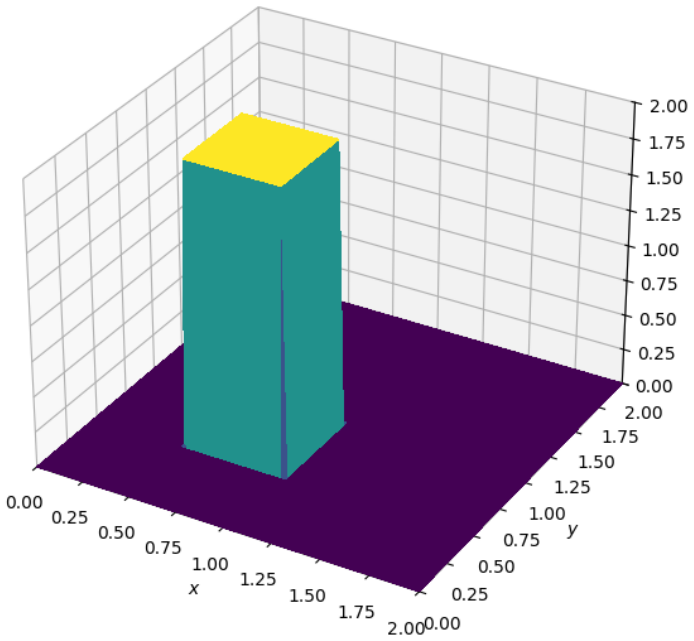


Figura 3.8: Condição inicial do pulso gaussiano em $t = 0$ s.

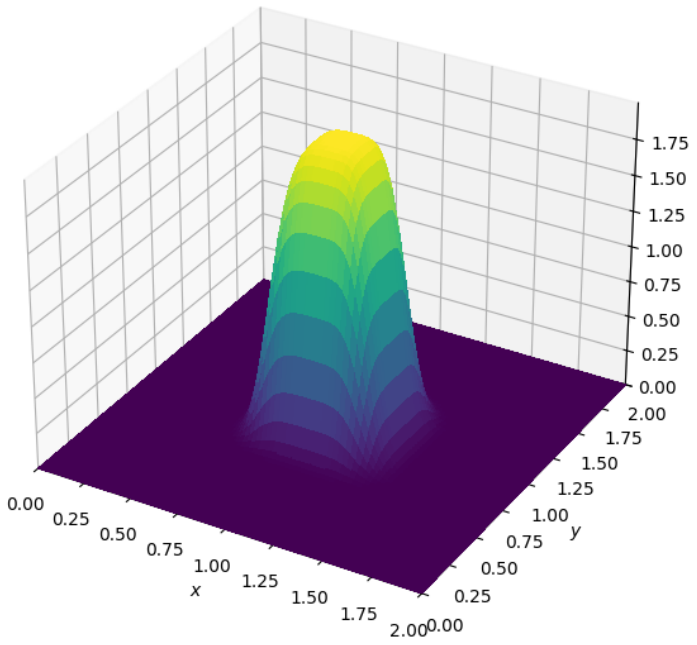


Figura 3.9: Solução no instante intermediário $t = 0.25\ s$.

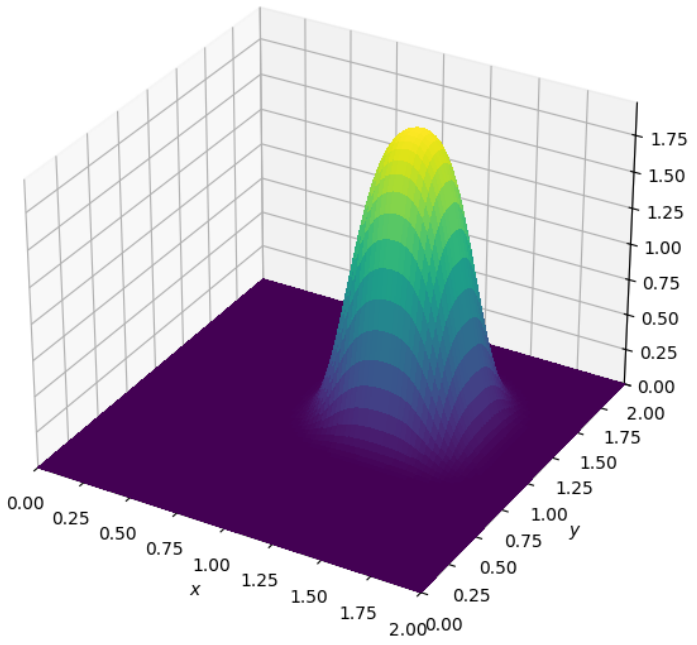


Figura 3.10: Solução no instante final $t = 0.5\ s$.

A capacidade de armazenamento temporal proporcionada pelo parâmetro `save` mostrou-se essencial para a análise qualitativa do fenômeno, permitindo não apenas a verificação da solução final, mas também o acompanhamento completo da dinâmica de transporte, além de facilitar a geração de animações que sintetizam a evolução do sistema, como a que pode ser visualizada no Github [14] ou diretamente clicando aqui.

3.3 Equação Parabólica

A equação de difusão representa um dos modelos mais fundamentais na física matemática, descrevendo fenômenos de transporte de massa, calor ou outras grandezas através de meios materiais. Nesta seção, aborda-se a equação de difusão bidimensional na forma:

$$\frac{\partial T}{\partial t} - \nu\frac{\partial^2 T}{\partial x^2} - \nu\frac{\partial^2 T}{\partial y^2} = f, \tag{3.10}$$

em que $T(t, x, y)$ representa o transporte de calor (^oC), ν o coeficiente de difusidade térmica (m^2/s) e f o termo fonte ($^oC/s$) que modela uma fonte ou sumidouro de calor em um domínio $\Omega_1 = [0, 1] \times [0, 1]$ ou $\Omega_2 = [0, 2\pi] \times [0, 2\pi]$.

A correta discretização do termo fonte é crucial para a precisão numérica de esquemas de diferenças finitas. Diferentes formulações do termo fonte podem representar desde casos simples com geração uniforme até situações complexas com dependência espacial e temporal.

O objetivo principal desta seção é demonstrar a flexibilidade do Devito na implementação de termos fonte com complexidade crescente, seguindo uma progressão pedagógica que inclui:

- Termo fonte constante ($f = c$): caso mais elementar, representando geração uniforme em todo o domínio;
- Termo fonte espacialmente dependente ($f = f(x)$): variação unidimensional ao longo de uma direção espacial;
- Termo fonte bidimensional ($f = f(x, y)$): variação completa no espaço bidimensional;
- Termo fonte espaço-temporal ($f = f(t, x)$): dependência temporal combinada com variação espacial unidimensional;
- Termo fonte completo ($f = f(t, x, y)$): caso mais geral com dependência temporal e espacial completa.

Através desta abordagem progressiva, busca-se não apenas ilustrar as capacidades do Devito para tratar problemas com diferentes níveis de complexidade, mas também fornecer um roteiro para a implementação de termos fonte arbitrários em problemas de equações diferenciais parciais. A análise comparativa das soluções numéricas para cada caso permitirá avaliar a influência do termo fonte no comportamento da solução e verificar a correta implementação no Devito.

Para as simulações numéricas, considera-se $\Delta t = \dfrac{\Delta x \Delta y}{4\nu}$ e tempo final de 1 segundo.

3.3.1 Termo Fonte Constante ($f = c$)

Para a construção do termo fonte constante, emprega-se a técnica de soluções manufaturadas. Define-se a solução analítica como:

$$T(t, x, y) = x$$

com condições de contorno e inicial dadas por:

$$\begin{cases} T(t,0,y) = 0, \\ T(t,1,y) = 1, \\ \dfrac{\partial T}{\partial y}(t,x,0) = \dfrac{\partial T}{\partial y}(t,x,1) = 0, \\ T(0,x,y) = x. \end{cases}$$

Logo, de (3.10), obtém-se:

$$0 - \nu(0+0) = f \quad \Rightarrow \quad f = 0.$$

Implementação no Devito

A implementação segue a estrutura estabelecida nas seções anteriores, com as adaptações necessárias para o termo fonte constante e as condições de contorno de Neumann na direção y.

```
# Condições de Contorno
bc  = [Eq(T[t+1, 0, y], 0.)]                    # Dirichlet: T(t,0,y) = 0
bc += [Eq(T[t+1, nx-1, y], 1)]                  # Dirichlet: T(t,1,y) = 0
bc += [Eq(T[t+1,x, 0], T[t+1,x, 1])]            # Neumann: dT/dy = 0 em y = 0
bc += [Eq(T[t+1,x, ny-1], T[t+1,x, ny-2])]      # Neumann: dT/dy = 0 em y = 1
```

As condições de contorno de Neumann homogêneas:

$$\begin{cases} \left.\dfrac{\partial T}{\partial y}\right|_{y=0} = 0 \\ \left.\dfrac{\partial T}{\partial y}\right|_{y=1} = 0. \end{cases}$$

são discretizadas utilizando diferenças finitas de primeira ordem:

- Para $y = 0$ (fronteira inferior):

$$\frac{T(t,x,0+\Delta y) - T(t,x,0)}{\Delta y} = 0 \Rightarrow T(t,x,0) = T(t,x,\Delta y)$$

- Para $y = 1$ (fronteira superior):

$$\frac{T(t,x,1) - T(t,x,1-\Delta y)}{\Delta y} = 0 \Rightarrow T(t,x,1) = T(t,x,1-\Delta y)$$

que é o que está sendo escrito no Devito, respectivamente, em:

```
bc += [Eq(T[t+1,x, 0], u[t+1,x, 1])]
bc += [Eq(T[t+1,x, ny-1], u[t+1,x, ny-2])]
```

A implementação do termo fonte constante é realizada diretamente na definição da equação:

```
T = TimeFunction(name='T', grid=grid, space_order=2, time_order=1)
f = 0  # Termo fonte constante nulo

eq = Eq(T.dt - nu * (T.dx2 + T.dy2), f)
stencil = solve(eq, T.forward)
eq_stencil = Eq(T.forward, stencil)
```

Resultados e Análises

A Tabela 3.5 apresenta os tempos de execução e erros numéricos obtidos.

Tabela 3.5: Análise de Convergência para o caso $f = 0$.

nx	Δx	Δt	Erro Relativo	$q_{\Delta x}$	$q_{\Delta t}$
11	$1 \cdot 10^{-1}$	$2.5 \cdot 10^{-3}$	$8.6200 \cdot 10^{-8}$	–	–
21	$5 \cdot 10^{-2}$	$6.25 \cdot 10^{-4}$	$5.2422 \cdot 10^{-8}$	0.7175	0.3588
41	$2.5 \cdot 10^{-2}$	$1.5625 \cdot 10^{-4}$	$7.3406 \cdot 10^{-8}$	-0.4857	-0.2429
81	$1.25 \cdot 10^{-2}$	$3.9063 \cdot 10^{-5}$	$9.0365 \cdot 10^{-8}$	-0.2999	-0.1499

A Tabela 3.5 apresenta a análise de convergência para o caso com termo fonte constante. Observa-se um comportamento atípico onde o erro numérico aumenta com o refinamento da malha, evidenciado pelos valores negativos das taxas de convergência q_h e $q_{\Delta t}$, tal que esse é a ordem de convergência para o método no tempo (Euler Explícito) e aquele a ordem de convergência para o método no espaço (Diferenças Finitas de 2ª Ordem). Apesar da convergência irregular, os erros relativos mantêm-se na ordem de 10^{-8}, indicando uma precisão adequada. Fato que pode ser visualizado nas Figuras 3.11 e 3.12.

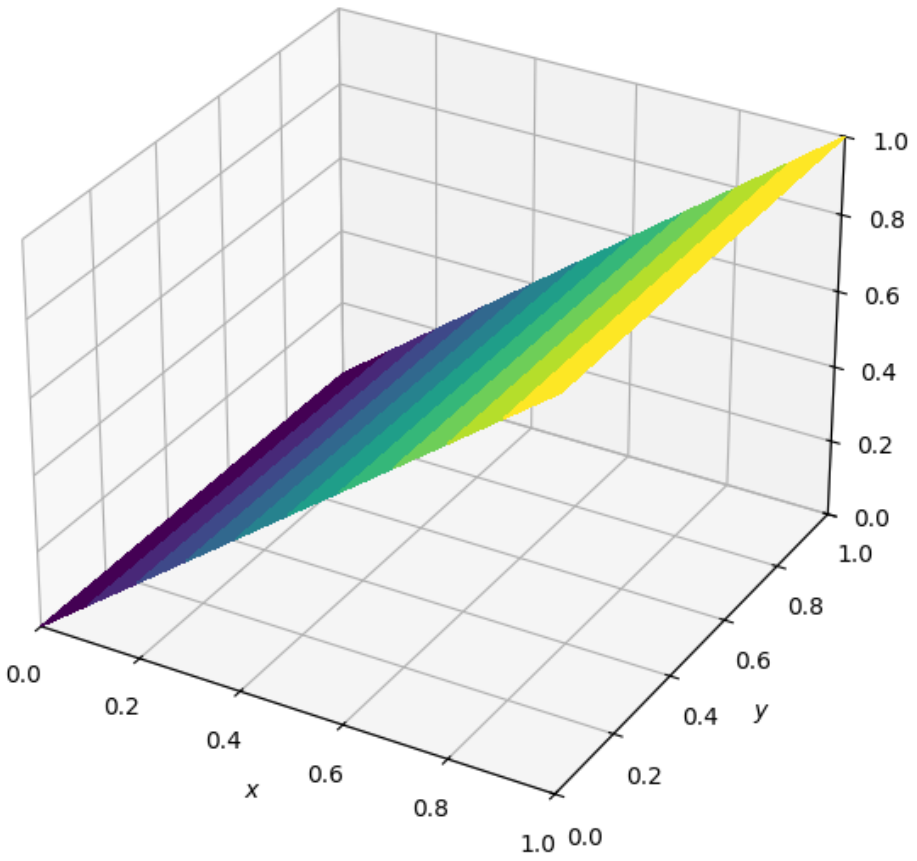


Figura 3.11: Solução numérica para $nx = 11$.

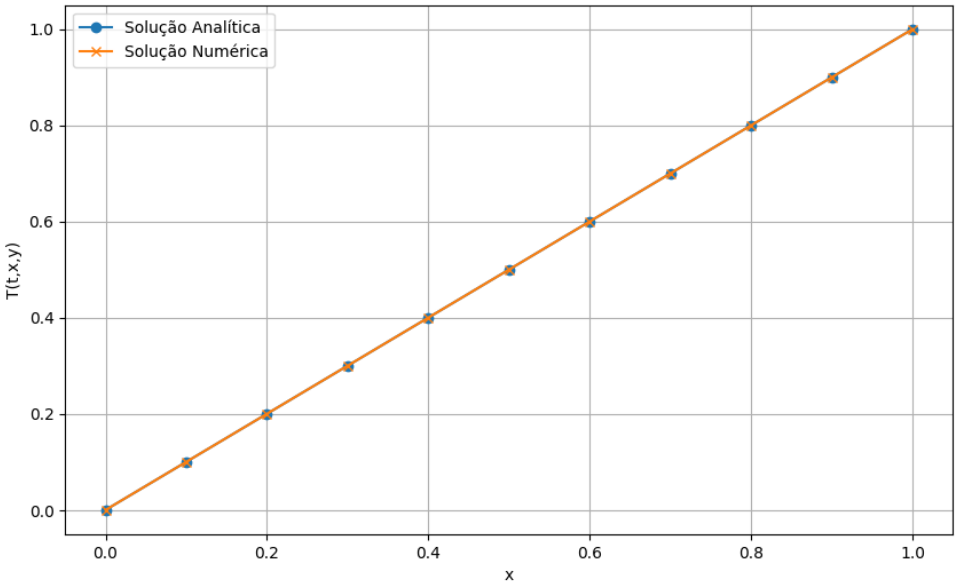


Figura 3.12: Solução numérica e analítica em um corte em $y = 0.5$ para $nx = 11$.

3.3.2 Termo Fonte em Uma Variável Espacial($f = f(x)$)

Agora considere as seguintes condições em Ω_1:

$$\begin{cases} T(t,0,y) = 0, \\ T(t,1,y) = 0, \\ \dfrac{\partial T}{\partial y}(t,x,0) = \dfrac{\partial T}{\partial y}(t,x,1) = 0, \\ T(0,x,y) = x^3 - 2x^2 + x, \end{cases}$$

e a solução analítica:

$$T(t,x,y) = x^3 - 2x^2 + x.$$

Logo, o termo fonte é:

$$f = f(x) = -\nu(6x - 4).$$

Implementação no Devito

A implementação mantém a mesma estrutura do caso anterior, modificando apenas a definição do termo fonte:

```
X, Y = np.meshgrid(np.linspace(0, 1, nx), np.linspace(0, 1, ny), indexing='
    ij')
f_data = -nu * (6*X - 4)
f = Function(name='f', grid=grid)
f.data[:] = f_data
```

Resultados e Análises

A Tabela 3.6 apresenta a análise de convergência para este caso, que exibe o mesmo padrão observado anteriormente: redução inicial do erro seguida de aumento com refinamentos sucessivos da malha.

Tabela 3.6: Análise de convergência para o caso $f = f(x)$.

nx	Δx	Δt	Erro Relativo	$q_{\Delta x}$	$q_{\Delta t}$
11	$1.0 \cdot 10^{-1}$	$2.50 \cdot 10^{-3}$	$5.83 \cdot 10^{-8}$	–	–
21	$5.0 \cdot 10^{-2}$	$6.25 \cdot 10^{-4}$	$5.56 \cdot 10^{-8}$	0.068	0.034
41	$2.5 \cdot 10^{-2}$	$1.56 \cdot 10^{-4}$	$7.18 \cdot 10^{-8}$	−0.368	−0.184
81	$1.25 \cdot 10^{-2}$	$3.91 \cdot 10^{-5}$	$1.07 \cdot 10^{-7}$	−0.577	−0.288

Comparando com os resultados da Tabela 3.5, observa-se que ambos os casos apresentam a mesma tendência de convergência irregular, com taxas negativas em malhas refinadas, os erros mantêm-se na mesma ordem de grandeza

As Figuras 3.13 e 3.14 mostra a excelente concordância entre as soluções numérica e analítica, validando a correta implementação do termo fonte dependente de x.

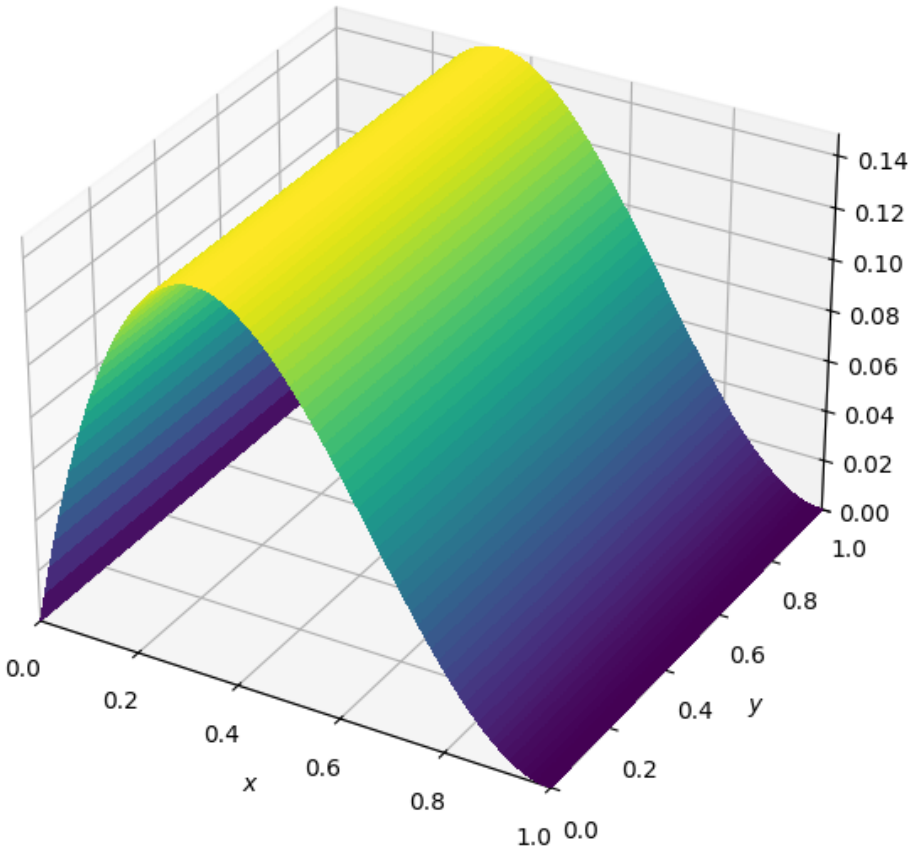


Figura 3.13: Solução numérica para $nx = 81$.

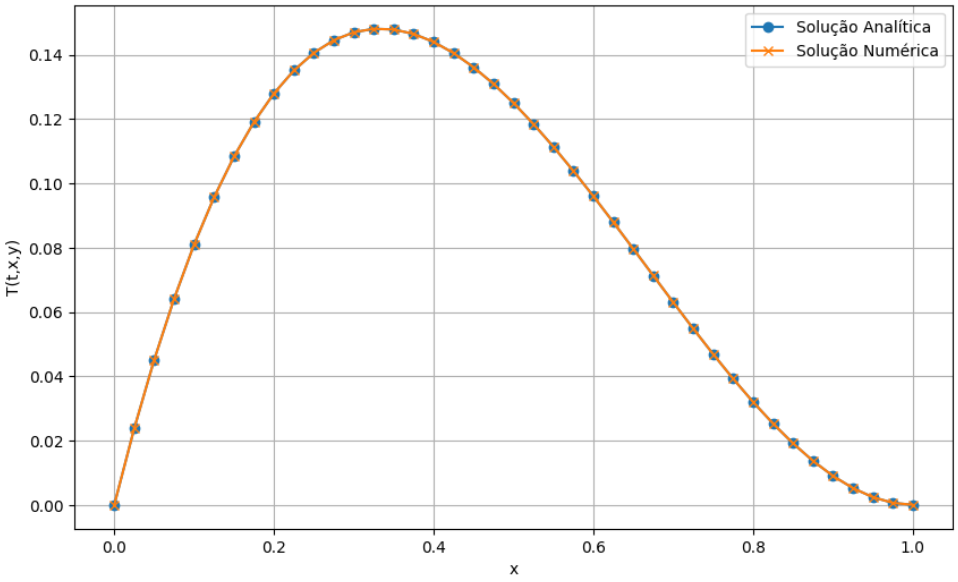


Figura 3.14: Soluções numérica e analítica em corte em $y = 0.5$ para $nx = 41$.

3.3.3 Termo Fonte em Duas Variáveis Espaciais($f = f(x, y)$)

Considera-se agora um termo fonte bidimensional. Através da técnica de soluções manufaturadas, define-se a solução analítica como:

$$T(t, x, y) = x(x-1)^2y^2(y-1)^2,$$

com condições de contorno e inicial em Ω_1:

$$\begin{cases} T(t, 0, y) = 0, \\ T(t, 1, y) = 0, \\ \dfrac{\partial T}{\partial y}(t, x, 0) = \dfrac{\partial T}{\partial y}(t, x, 1) = 0, \\ T(0, x, y) = x(x-1)^2y^2(y-1)^2. \end{cases}$$

Obtém-se, portanto, o termo fonte:

$$f(x, y) = -\nu \left[(6x-4)(y^4 - 2y^3 + y^2) + (x^3 - 2x^2 + x)(12y^3 - 12y + 2)\right].$$

Implementação no Devito

A implementação mantém a estrutura dos casos anteriores, com a definição do termo fonte bidimensional:

```
X, Y = np.meshgrid(np.linspace(0, 1, nx), np.linspace(0, 1, ny), indexing='
    ij')
f_data = -nu * ((6*X - 4)*(Y**4 - 2*Y**3 + Y**2) +
                (X**3 - 2*X**2 + X)*(12*Y**3 - 12*Y + 2))
f = Function(name='f', grid=grid)
f.data[:] = f_data
```

Resultados e Análises

A Tabela 3.7 apresenta uma análise de convergência marcadamente diferente dos casos anteriores, mostrando um comportamento estável e convergente.

Tabela 3.7: Análise de convergência para o caso $f(x, y)$.

nx	Δx	Δt	Erro Relativo	$q_{\Delta x}$	$q_{\Delta t}$
11	$1.0 \cdot 10^{-1}$	$2.50 \cdot 10^{-3}$	0.8537	–	–
21	$5.0 \cdot 10^{-2}$	$6.25 \cdot 10^{-4}$	0.3894	1.132	0.566
41	$2.5 \cdot 10^{-2}$	$1.56 \cdot 10^{-4}$	0.1855	1.070	0.535
81	$1.25 \cdot 10^{-2}$	$3.91 \cdot 10^{-5}$	0.0904	1.037	0.519

Note que há um contraste significativo com os casos anteriores: todas as taxas são positivas, indicando convergência linear consistente; os erros relativos são significativamente maiores (10^{-1} vs 10^{-7}-10^{-8}), refletindo a maior complexidade da solução analítica; diferente dos casos constantes e unidimensionais, este exibe a convergência monotônica esperada teoricamente.

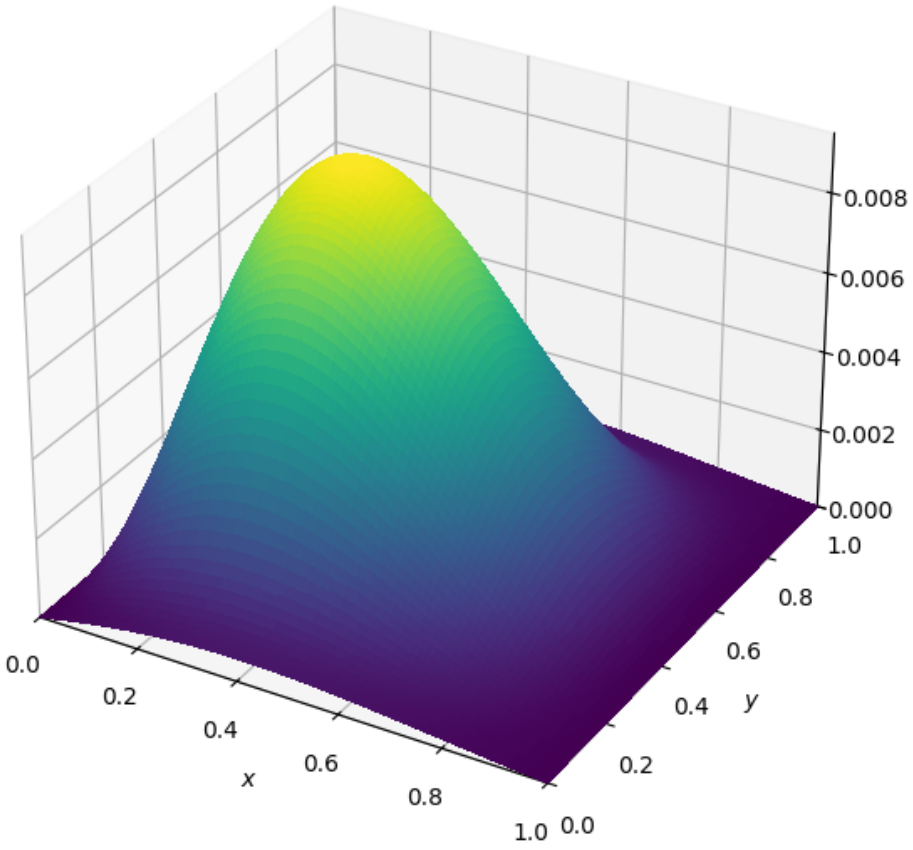


Figura 3.15: Solução numérica para o caso $f(x, y)$ com $nx = 81$.

A sequência de Figuras 3.16 - 3.18 permite a visualização da convergência da solução numérica conforme refinamento da malha.

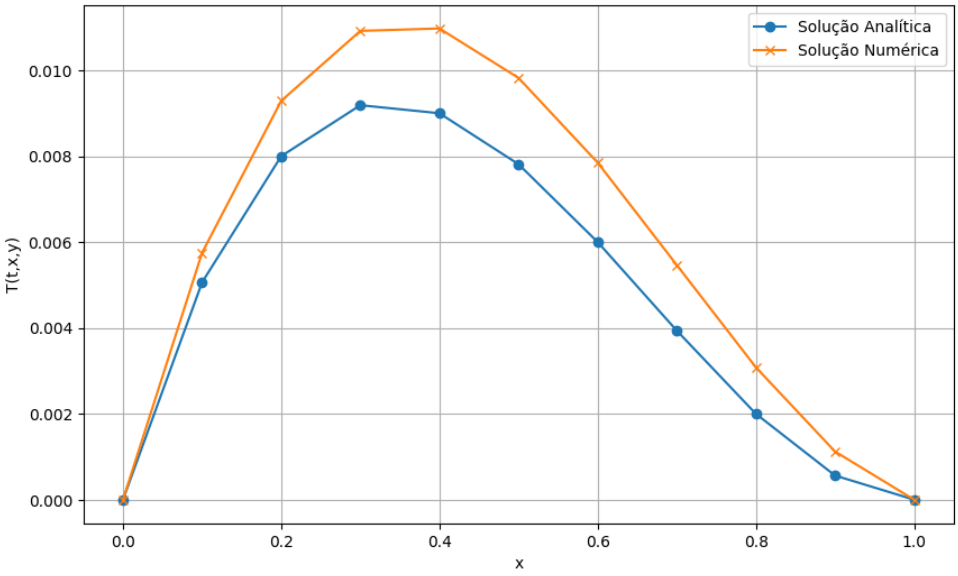


Figura 3.16: Soluções numérica e analítica em corte em $y = 0.5$ com $nx = 11$.

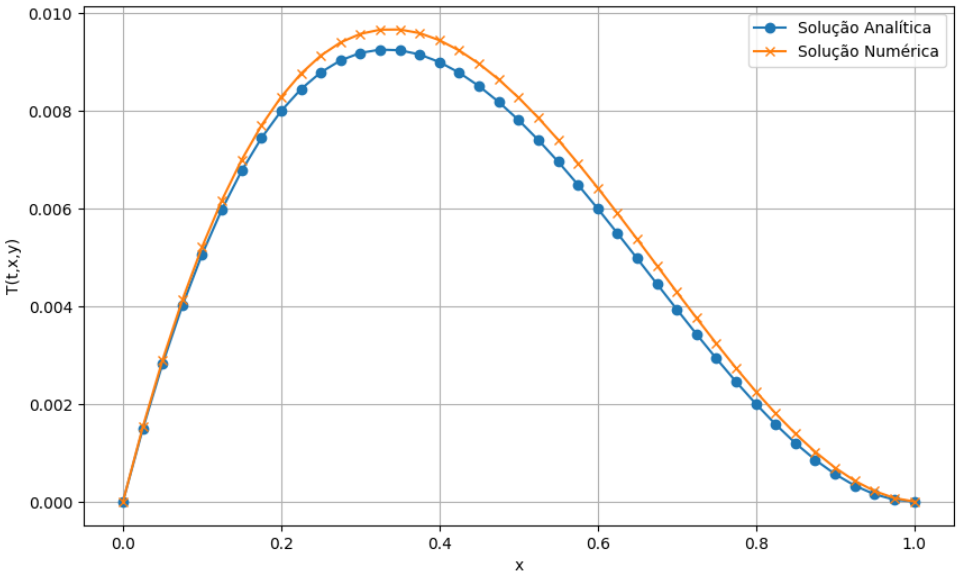


Figura 3.17: Soluções numérica e analítica em corte em $y = 0.5$ com $nx = 41$.

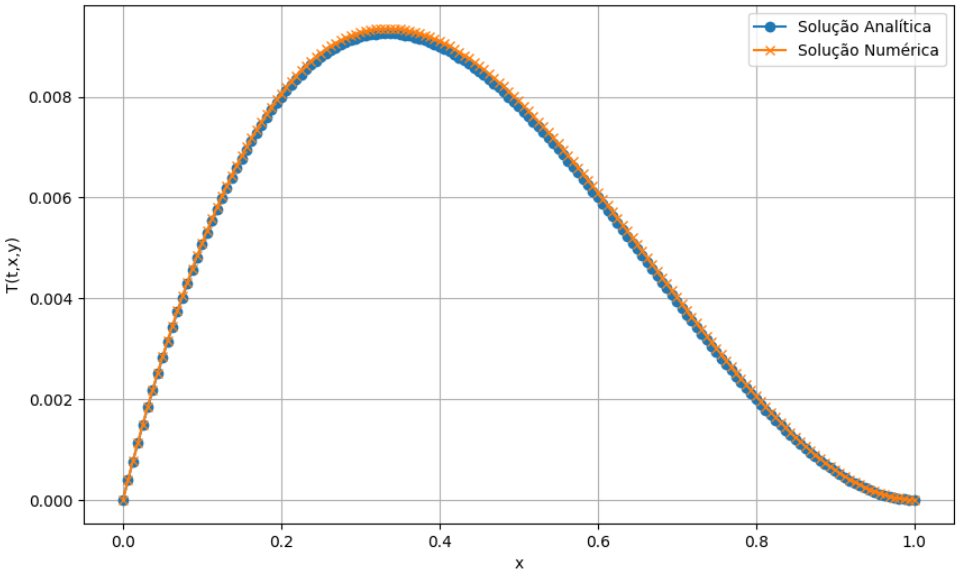


Figura 3.18: Soluções numérica e analítica em corte em $y = 0.5$ com $nx = 161$.

Esse caso demonstra que o Devito lida eficientemente com termos fonte bidimensionais complexos, produzindo resultados numericamente estáveis e convergentes.

3.3.4 Termo Fonte em Uma Variável Espacial e Temporal ($f = f(t, x)$)

Agora, considera-se um termo fonte que varia no espaço e no tempo. Definindo a solução analítica como:

$$T(t, x, y) = -x(x - 1)t,$$

com condições de contorno e inicial em Ω_1:

$$\begin{cases} T(t, 0, y) = 0, \\ T(t, 1, y) = 0, \\ \dfrac{\partial T}{\partial y}(t, x, 0) = \dfrac{\partial T}{\partial y}(t, x, 1) = 0, \\ T(0, x, y) = 0. \end{cases}$$

Logo, obtém-se o termo fonte:

$$f(t, x) = -x^2 + x + 2t\nu.$$

Implementação no Devito

A implementação utiliza uma `Function` com dimensão temporal para definir eficientemente o termo fonte variante no tempo:

```
f = Function(name='f', shape=(nt,nx,ny), dimensions=(time_dim,x,y))
X, Y = np.meshgrid(np.linspace(x0, x1, nx), np.linspace(y0, y1, ny),
    indexing='ij')
time_dime = np.arange(0, tstop, dt).reshape(-1, 1, 1) # Cria-se um vetor 1D
    com todos os instantes temporais e o redimensiona para 3D com dimensões
    (nt, 1, 1)
f_data = - (X**2) + X + 2 * time_dime * nu   # combina as dimensões: (1,nx,
    ny) + (nt,1,1) -> (nt,nx,ny)
f.data[:] = f_data
```

Resultados e Análises

A Tabela 3.8 apresenta uma análise de convergência que combina características dos casos anteriores:

Tabela 3.8: Análise de convergência para o caso $f(t, x)$.

nx	Δx	Δt	Erro Relativo	$q_{\Delta x}$	$q_{\Delta t}$
11	$1.0 \cdot 10^{-1}$	$2.50 \cdot 10^{-3}$	$2.50 \cdot 10^{-3}$	–	–
21	$5.0 \cdot 10^{-2}$	$6.25 \cdot 10^{-4}$	$6.28 \cdot 10^{-4}$	1.993	0.996
41	$2.5 \cdot 10^{-2}$	$1.56 \cdot 10^{-4}$	$1.69 \cdot 10^{-4}$	1.896	0.948
81	$1.25 \cdot 10^{-2}$	$3.91 \cdot 10^{-5}$	$1.00 \cdot 10^{-4}$	0.755	0.377

Observe que os primeiros refinamentos mostram taxas próximas a 2 e 1, ótimo para os esquemas utilizados, mas logo segue o padrão observado nos casos da equação de Poisson.

A Figura 3.19 mostra a evolução temporal da solução, evidenciando o crescimento linear esperado com o tempo. E a Figura 3.20 ilustra a ótima aproximação numérica apesar da malha grossa.

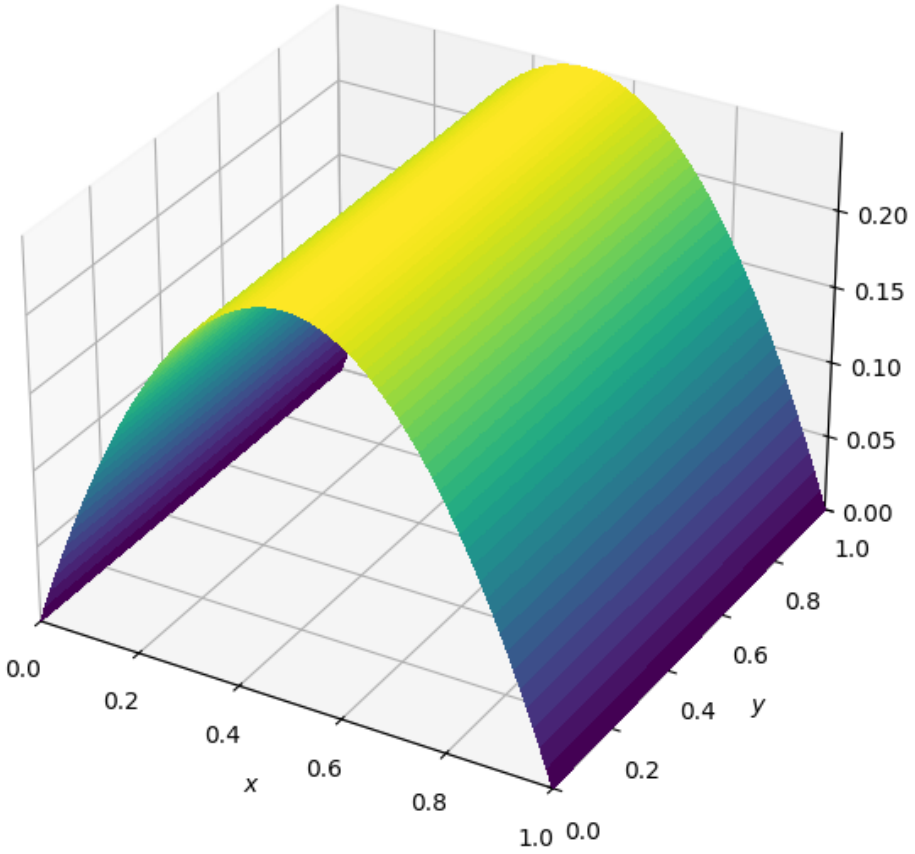


Figura 3.19: Solução numérica para $nx = 81$.

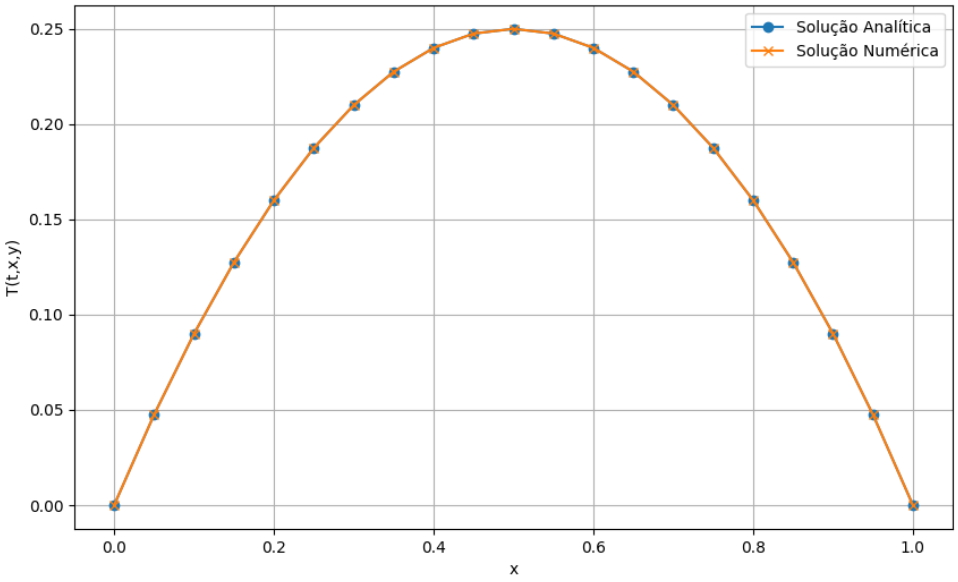


Figura 3.20: Soluções numérica e analítica em um corte em $y = 0.5$ com $nx = 21$.

Esse caso demonstra que o Devito lida eficientemente com termos fonte espaço-temporais, mantendo boa precisão numérica mesmo em problemas com dependência temporal explícita.

3.3.5 Termo Fonte em Duas Variáveis Espaciais e Temporal ($f = f(t, x, y)$)

Por fim, considera-se um termo fonte que varia no espaço bidimensionalmente e no tempo. Definindo a solução analítica como:

$$T(t, x, y) = sen(x)cos(y)t,$$

com condições de contorno e inicial em Ω_2:

$$\begin{cases} T(t, 0, y) = 0, \\ T(t, 2\pi, y) = 0, \\ \dfrac{\partial T}{\partial y}(t, x, 0) = \dfrac{\partial T}{\partial y}(t, x, 2\pi) = 0, \\ T(0, x, y) = 0. \end{cases}$$

Logo, obtém-se o termo fonte:

$$f(t, x, y) = sen(x)cos(y)\left(1 + \nu 2t\right)$$

Implementação no Devito

A implementação mantém a estrutura do caso anterior, com a definição do termo fonte bidimensional e temporal:

```
f = Function(name='f', shape=(nt,nx,ny), dimensions=(time_dim,x,y))
X, Y = np.meshgrid(np.linspace(x0, x1, nx), np.linspace(y0, y1, ny),
    indexing='ij')
time_dime = np.arange(dt, tstop + dt, dt).reshape(nt, 1, 1)
f_data = np.sin(X) * np.cos(Y) * (1 + nu*2*time_dime)
f.data[:] = f_data
```

Resultados e Análises

A Tabela 3.9 apresenta uma análise de convergência consistente e estável:

Tabela 3.9: Análise de convergência para o caso $f(t, x, y)$.

nx	Δx	Δt	Erro Relativo	$q_{\Delta x}$	$q_{\Delta t}$
11	$6.28 \cdot 10^{-1}$	$9.86 \cdot 10^{-2}$	$1.84 \cdot 10^{-1}$	–	–
21	$3.14 \cdot 10^{-1}$	$2.46 \cdot 10^{-2}$	$7.18 \cdot 10^{-2}$	1.489	0.784
41	$1.57 \cdot 10^{-1}$	$6.16 \cdot 10^{-3}$	$2.69 \cdot 10^{-2}$	1.284	0.645
81	$7.85 \cdot 10^{-2}$	$1.54 \cdot 10^{-3}$	$1.24 \cdot 10^{-2}$	1.152	0.578

A consistência na redução do erro é um ponto positivo para a ferramenta Devito. No entanto, os resultados sugerem oportunidades de aprimoramento, particularmente no que concerne à discretização temporal. Efeito que pode ser visto quantitativamente pela Tabela 3.9 e qualitativamente pelas Figuras 3.21 - 3.23.

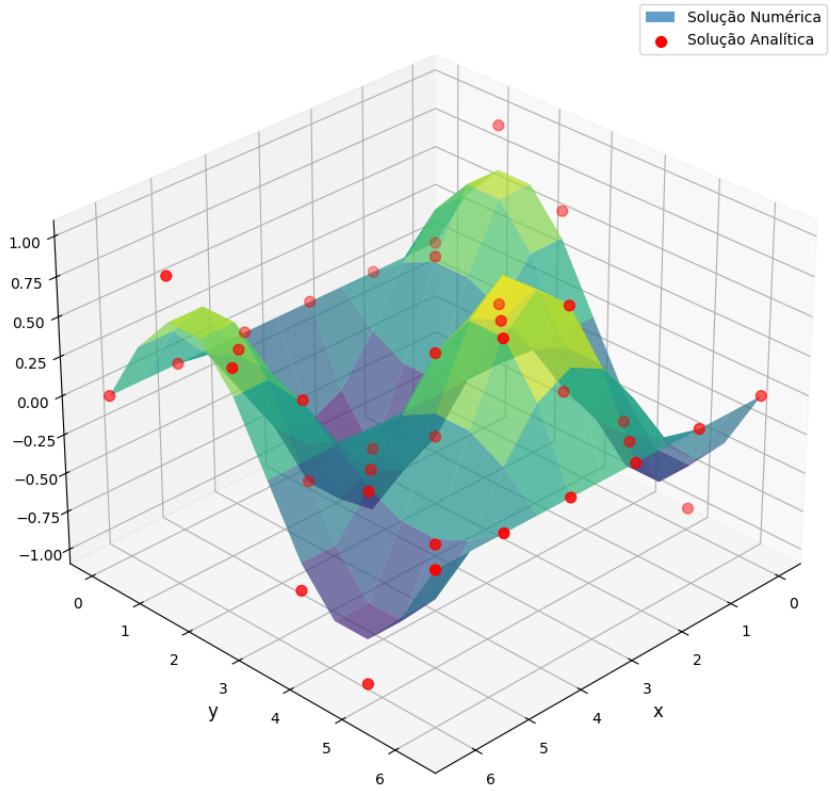


Figura 3.21: Solução numérica para $nx = 11$.

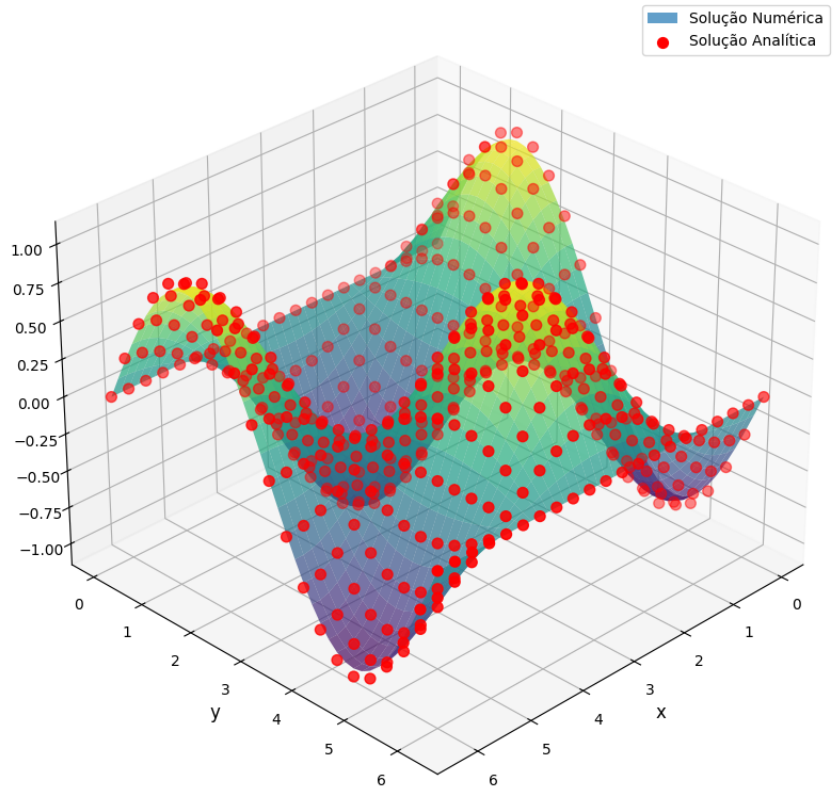


Figura 3.22: Solução numérica para $nx = 41$.

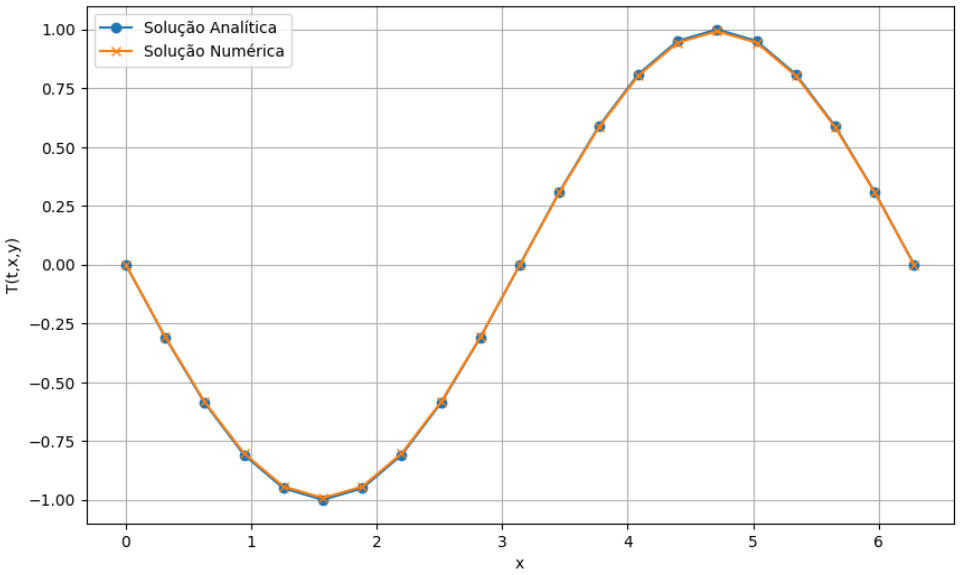


Figura 3.23: Soluções numérica e analítica em um corte em $y = \pi$ com $nx = 21$.

Capítulo

4

Integração Devito-PETSc

Uma limitação fundamental identificada no uso atual do Devito é a dependência predominante de métodos explícitos para a resolução de sistemas lineares, particularmente no contexto da equação de Poisson. Embora a técnica de pseudo-transiente contorne essa restrição, ela introduz um custo computacional adicional e não explora todo o potencial de métodos iterativos avançados.

Para superar essa limitação, este trabalho apresenta uma integração experimental entre o Devito e a biblioteca PETSc (*Portable, Extensible Toolkit for Scientific Computation*), desenvolvida em colaboração com Zoe Leibowitz [15], pós-doutoranda do Imperial College London. Esta integração permite a utilização de métodos implícitos e iterativos combinados com pré-condicionadores avançados, viabilizando a resolução eficiente de sistemas lineares esparsos resultantes da discretização de problemas elípticos.

Cabe ressaltar que o PETSc não está oficialmente implementado nem disponível como parte integrante do Devito. A integração aqui apresentada encontra-se em caráter experimental, sendo desenvolvida ativamente pela equipe do Imperial College London. O suporte e os exemplos fornecidos por Leibowitz foram essenciais para a implementação bem-sucedida das soluções documentadas nesta dissertação.

Este capítulo apresenta uma investigação numérica da equação de Poisson utilizando duas estratégias de solução distintas. A primeira abordagem emprega o Devito utilizando a técnica de pseudo-transiente, como mostrado no capítulo anterior. A segunda abordagem utiliza o PETSc, um conjunto sofisticado de estruturas de dados e rotinas para a solução escalável de EDPs. Diferentemente do método explícito de pseudo-transiente, o PETSc permite a solução direta do sistema linear decorrente da discretização por diferenças finitas, empregando métodos de subespaço de Krylov (como GMRES e gradiente Conjugado) combinados com pré-condicionadores avançados.

Vale notar que, embora o PETSc suporte nativamente pré-condicionadores multigrid, esta funcionalidade específica ainda não está disponível na integração atual com o Devito, constituindo uma direção promissora para trabalhos futuros.

Ambas as metodologias são aplicadas a um problema bidimensional em geometria quadrada com condições de contorno de Dirichlet homogêneas. O método das soluções manufaturadas é empregado para validar a precisão numérica, permitindo a quantificação precisa dos erros de discretização por meio da comparação com a solução analítica.

4.1 Problema de Poisson Bidimensional

Considere o domínio $\Omega = [0,1] \times [0,1]$ e condições de contorno homogêneas de Dirichlet aplicadas em todas as faces do domínio. O problema é formulado como:

$$\begin{cases} \dfrac{\partial^2 p}{\partial x^2} + \dfrac{\partial^2 p}{\partial y^2} = f(x,y), \\ p(0,y) = p(1,y) = 0, \quad \forall y \in [0,1], \\ p(x,0) = p(x,1) = 0, \quad \forall x \in [0,1], \end{cases} \tag{4.1}$$

sendo p uma função escalar e $f(x,y)$ o termo fonte, também uma função escalar de duas variáveis. Para permitir a validação da solução numérica através da comparação com a solução analítica, utiliza-se o método das soluções manufaturadas.

Solução Manufaturada

A solução analítica é definida como:

$$e^{x-y} \cdot x(1-x) \cdot y(1-y). \tag{4.2}$$

Pode-se verificar que as condições de contorno são satisfeitas, uma vez que $p(0,y) = p(1,y) = p(x,0) = p(x,1) = 0$ para todos $x, y \in [0,1]$. Para determinar o termo fonte, calculamos as segundas derivadas parciais de $p(x,y)$:

$$\begin{aligned} \frac{\partial^2 p}{\partial x^2} &= xy(x+3)(y-1)e^{x-y}, \\ \frac{\partial^2 p}{\partial y^2} &= x(x-1)[y(y-1) - 4y + 4]e^{x-y}. \end{aligned}$$

Substituindo na Equação (4.1), obtemos:

$$\begin{aligned} \frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} &= xy(x+3)(y-1)e^{x-y} + x(x-1)[y(y-1) - 4y + 4]e^{x-y} \\ &= f(x,y). \end{aligned}$$

Assim, o termo fonte manufaturado é dado por:

$$f(x,y) = 2x(xy^2 - 3xy + 2x + y^2 + y - 2)e^{x-y}.$$

Discretização

Assim como no capítulo anterior, utiliza-se a técnica de pseudo-transiente para a resolução utilizando somente o Devito. Para a solução numérica, utiliza-se $t_f = 1$ e um passo de tempo de $\Delta t = 0.1 \cdot \Delta x \cdot \Delta y$, com $\Delta x = \Delta y = \dfrac{1}{n}$, em que n é o número de pontos em cada direção espacial e $\Delta x = \Delta y$ são o espaçamento da malha espacial.

Para o *solver* PETSc, nenhum avanço temporal é necessário, pois o método resolve diretamente o sistema linear de estado estacionário decorrente da discretização da equação de Poisson. Esta abordagem permite maior flexibilidade na definição dos critérios de convergência. Neste trabalho, selecionamos duas condições de parada: o número máximo de iterações, definido para igualar o número de passos de pseudo-transiente (nt) utilizado na abordagem Devito ($ksp\_max\_it = nt = \dfrac{1}{\Delta t}$), e uma tolerância de $ksp\_rtol = 10^{-10}$,

que controla a tolerância entre iterações consecutivas. Além disso, empregamos o método do gradiente conjugado como solver iterativo, sem pré-condicionamento, uma vez que o pré-condicionamento não é ainda suportado.

Implementação com PETSc

A integração entre Devito e PETSc requer algumas adaptações na configuração e na definição do problema. A seguir, será descrita as principais modificações necessárias para utilizar o solver linear do PETSc em conjunto com a geração de código do Devito.

A primeira etapa consiste em configurar o ambiente de compilação para utilizar o compilador MPI *(Message Passing Interface)*, necessário para utilizar o PETSc:

```
configuration['compiler'] = 'custom'
os.environ['CC'] = 'mpicc'
```

Em seguida, inicializa-se o PETSc:

```
PetscInitialize()
```

Uma das principais diferenças na abordagem com PETSc é a forma como as condições de contorno são implementadas. Em vez de serem aplicadas diretamente no operador temporal, as condições de contorno são definidas através de subdomínios, permitindo que o solver linear as incorpore adequadamente na montagem do sistema matricial.

No Devito, um `SubDomain` representa uma região específica do domínio computacional. Essa região pode corresponder ao domínio interno, a uma fronteira ou a qualquer subconjunto da malha definido pelo usuário. A classe `SubDomain` permite descrever essas regiões de forma simbólica, especificando quais pontos da malha pertencem ao subdomínio. Durante a geração do código, o Devito utiliza essas informações para restringir a aplicação de operadores ou condições de contorno apenas aos pontos pertencentes ao subdomínio especificado.

Para um domínio bidimensional com condições de Dirichlet homogêneas em todas as fronteiras, definem-se quatro subdomínios:

```
class SubTop(SubDomain):
name = 'subtop'
def define(self, dimensions):
x, y = dimensions
return {x: x, y: ('right', 1)}

class SubBottom(SubDomain):
name = 'subbottom'
def define(self, dimensions):
x, y = dimensions
return {x: x, y: ('left', 1)}

class SubLeft(SubDomain):
name = 'subleft'
def define(self, dimensions):
x, y = dimensions
return {x: ('left', 1), y: ('middle', 1, 1)}

class SubRight(SubDomain):
name = 'subright'
def define(self, dimensions):
x, y = dimensions
return {x: ('right', 1), y: ('middle', 1, 1)}
```

As expressões `('left', 1)` e `('right', 1)` indicam, respectivamente, a primeira e a última camada de pontos da malha naquela dimensão, enquanto `('middle', 1, 1)` representa a região interna, excluindo uma camada de pontos em cada extremidade. Assim, por exemplo, o subdomínio `SubTop` corresponde apenas à fronteira superior, enquanto `SubLeft` corresponde exclusivamente à fronteira esquerda, desconsiderando os cantos, que já pertencem aos subdomínios superior e inferior.

A instanciação dos subdomínios é realizada conforme abaixo:

```
sub1 = SubTop()
sub2 = SubBottom()
sub3 = SubLeft()
sub4 = SubRight()

subdomains = (sub1, sub2, sub3, sub4)
```

Com os subdomínios definidos, as condições de contorno são expressas como `EssentialBC`, associando a variável alvo, o valor na fronteira e o subdomínio correspondente:

```
bcs = [EssentialBC(phi, bc, subdomain=sub1)]
bcs += [EssentialBC(phi, bc, subdomain=sub2)]
bcs += [EssentialBC(phi, bc, subdomain=sub3)]
bcs += [EssentialBC(phi, bc, subdomain=sub4)]
```

O solver PETSc é configurado através da função `petscsolve`, que recebe:

- As expressões que definem o problema (`exprs`);
- A variável alvo (`phi`);
- Parâmetros do solver, incluindo:
  - `ksp_type`: Tipo de método de Krylov (ex: `'cg'` para Gradiente Conjugado);
  - `ksp_rtol`: Tolerância relativa para convergência;
  - `ksp_converged_reason`: Habilita a exibição da razão de convergência.

```
petsc = petscsolve(exprs, target=phi, solver_parameters={'ksp_type': 'cg',
    'ksp_rtol': 1e-10, 'ksp_converged_reason': None})
```

Por fim, o operador é gerado e executado utilizando a linguagem `'petsc'`:

```
with switchconfig(language='petsc'):
op = Operator(petsc)
op.apply()
```

O contexto `switchconfig` altera temporariamente a configuração do Devito para gerar código compatível com o PETSc, produzindo um operador que monta e resolve o sistema linear utilizando os métodos especificados.

4.2 Resultados e Análise

Devito

A Figura 4.1 mostra a solução numérica após atingir o estado estacionário para uma malha de 11^2, demonstrando excelente concordância qualitativa com a solução analítica manufaturada, mesmo para uma malha moderada de 11^2 pontos.

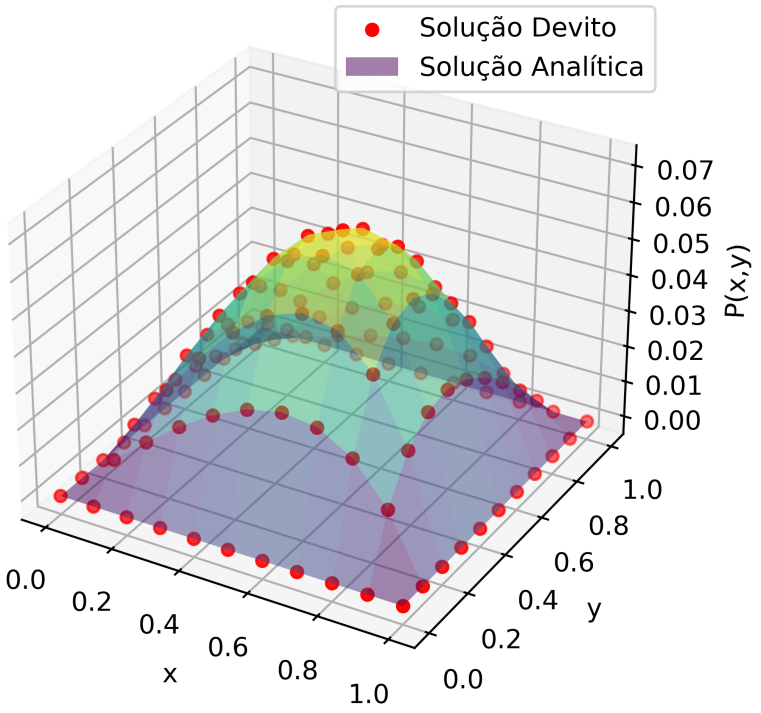


Figura 4.1: Comparação entre as soluções numérica e analítica utilizando Devito em uma malha de 11^2.

A Tabela 4.1 apresenta os resultados quantitativos para diferentes refinamentos de malha, variando de 11^2 a 161^2 pontos de grade. O erro na norma L^2 e a ordem de convergência são reportados para fornecer uma avaliação abrangente dos erros de discretização. A ordem de convergência (O) demonstra que o método atinge precisão de segunda ordem, como esperado para o esquema de diferenças finitas centradas empregado. Para as malhas mais refinadas, a ordem de convergência se aproxima do valor teórico de 2 com notável precisão (1.9999), validando a corretude da implementação.

Em relação ao desempenho computacional, o tempo necessário para a simulação aumenta aproximadamente linearmente com o número de passos de tempo e quadraticamente com o tamanho da malha. A malha de 11^2 requer apenas 0.37 segundos e 1000 iterações, enquanto a malha de 161^2 demanda 115 segundos e 256000 iterações. Este comportamento reflete a natureza explícita do método de pseudo-transiente, onde o número de iterações deve aumentar à medida que a malha é refinada devido à restrição de estabilidade CFL.

Tabela 4.1: Resultados da simulação da equação de Poisson $2D$ utilizando Devito.

n	Tempo (s)	(nt)	Erro L^2	O
11	0.37	1000	$4.99 \cdot 10^{-3}$	—
21	1.45	4000	$1.25 \cdot 10^{-3}$	1.9988
41	6.08	16000	$3.12 \cdot 10^{-4}$	1.9996
81	25.15	64000	$7.81 \cdot 10^{-5}$	1.9999
161	115.21	256000	$1.95 \cdot 10^{-5}$	1.9999

Integração PETSc

A Figura 4.2 mostra a solução numérica obtida com PETSc para uma malha de 21^2. Mesmo nesta resolução relativamente grosseira, a solução exibe excelente concordância com a solução analítica, demonstrando a precisão do método.

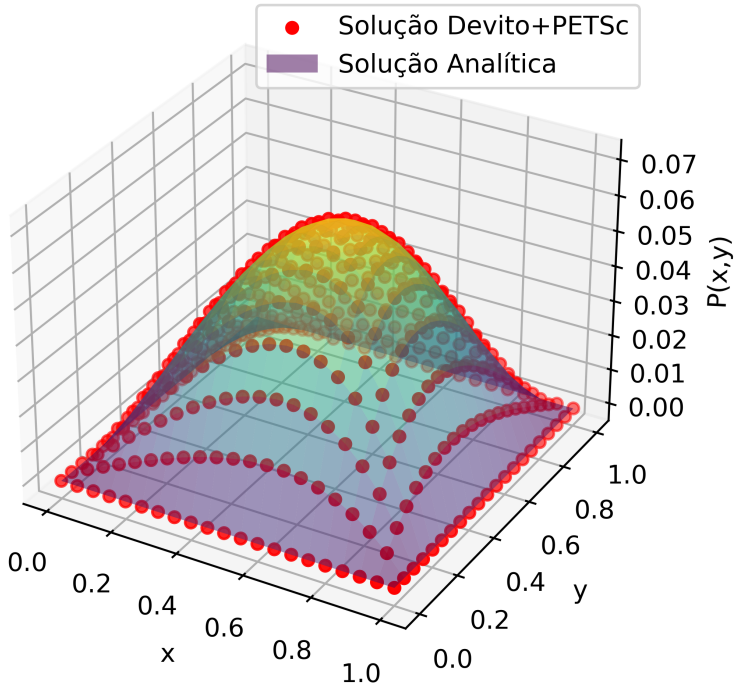


Figura 4.2: Comparação entre as soluções numérica e analítica utilizando PETSc em uma malha de 21^2.

A Tabela 4.2 apresenta os resultados quantitativos para refinamentos de malha variando de 11^2 a 161^2 pontos de grade. Várias observações merecem discussão.

Tabela 4.2: Resultados da simulação da equação de Poisson $2D$ utilizando PETSc.

n	Tempo (s)	Iterações	Erro L^2	O
11	0.01	37	$4.99 \cdot 10^{-3}$	—
21	0.01	82	$1.25 \cdot 10^{-3}$	1.9988
41	0.01	167	$3.12 \cdot 10^{-4}$	1.9996
81	0.02	337	$7.81 \cdot 10^{-5}$	1.9999
161	0.09	683	$1.95 \cdot 10^{-5}$	1.9999

Primeiramente, os níveis de erro alcançados pelo PETSc são muito semelhantes aos da outra abordagem. Os erros na norma L^2 variam de $4.99 \cdot 10^{-3}$ para a malha mais grosseira a $1.95 \cdot 10^{-5}$ para a malha mais refinada, e a ordem de convergência confirma a precisão de segunda ordem também para o solver PETSc. Esta concordância valida que ambas as implementações representam corretamente a mesma discretização por diferenças finitas.

Em segundo lugar, o número de iterações necessário para a convergência aumenta com o refinamento da malha, variando de 37 iterações para a malha de 11^2 a 683 iterações para a malha de 161^2. Este comportamento reflete o aumento do número de condição do Laplaciano discreto à medida que a malha é refinada. No entanto, o número de iterações

cresce apenas moderadamente em comparação com o aumento do tamanho do problema (de 121 para 25921 pontos de grade).

Terceiro, e mais importante, o tempo computacional exibe um comportamento de escalonamento fundamentalmente diferente em comparação com a abordagem explícita do Devito. Para as malhas mais grosseiras (11^2 a 41^2), o tempo de solução é efetivamente constante em 0.01 segundos, muito pequeno para ser medido de forma confiável com rotinas de temporização padrão. Para a malha de 81^2, o PETSc requer apenas 0.02 segundos, em comparação com 25.15 segundos para o Devito, um fator de aceleração superior a 1200. Mesmo para a maior malha considerada, 161^2, o PETSc completa a solução em apenas 0.09 segundos, enquanto o Devito requer 115 segundos. Esta diferença dramática de desempenho ressalta a vantagem dos métodos implícitos com *solvers* iterativos modernos, particularmente à medida que os tamanhos dos problemas aumentam. Deve-se notar que estes resultados foram obtidos sem pré-condicionamento; uma vez que o suporte completo a pré-condicionadores for integrado ao Devito, a abordagem baseada em PETSc deverá se tornar uma opção ainda mais atraente para simulações em larga escala.

4.3 Conclusão

Os resultados apresentados nas Tabelas 4.1 e 4.2 revelam diferenças marcantes entre as duas estratégias de solução. Embora ambos os métodos atinjam a mesma precisão e convergência de segunda ordem, confirmando a corretude de ambas as implementações, seu desempenho computacional difere por ordens de grandeza.

As implicações práticas destes resultados são claras. A abordagem de pseudo-transiente no Devito, embora conceitualmente simples e fácil de implementar, torna-se proibitivamente cara para todas, exceto as malhas bidimensionais mais grosseiras. Sua utilidade é amplamente confinada a problemas onde a prototipagem rápida e a conveniência da geração de código superam as preocupações com eficiência computacional, ou onde o esquema de avanço temporal explícito se alinha naturalmente com a física do problema (por exemplo, simulações dependentes do tempo).

A abordagem integrada Devito-PETSc, em contraste, combina o melhor dos dois mundos: a conveniência simbólica e a geração automatizada de código do Devito para construir operadores discretos e gerenciar grades computacionais, combinada com os sofisticados *solvers* lineares e o desempenho escalável do PETSc. Esta abordagem híbrida permite a solução de problemas elípticos com alta precisão e recursos computacionais mínimos, abrindo possibilidades para simulações mais realistas.

Capítulo

5

Soluções das Equações de Navier-Stokes Utilizando Devito

As Equações de Navier-Stokes representam o pilar fundamental da mecânica dos fluidos newtonianos. Estas equações descrevem a conservação de massa, quantidade de movimento e energia para fluidos viscosos incompressíveis, constituindo um sistema acoplado de equações diferenciais parciais não-lineares.

Nesse capítulo, aborda-se a implementação numérica das equações de Navier-Stokes utilizando o Devito, com ênfase em uma configuração clássica da dinâmica dos fluidos computacional: o escoamento em cavidade *lid-driven*. Problema esse consolidado na literatura, permitindo a avaliação da capacidade do Devito em capturar fenômenos fluidodinâmicos complexos.

O objetivo central é demonstrar a aplicabilidade do Devito na simulação de escoamentos viscosos incompressíveis, avaliando sua precisão na reprodução de estruturas vorticais, perfis de velocidade e corrente. Através de comparações com resultados numéricos e experimentais da literatura, busca-se estabelecer os limites e potencialidades dessa ferramenta no contexto da dinâmica dos fluidos computacional.

5.1 Formulação Corrente-Vorticidade

As equações de Navier-Stokes admitem diversas formulações matemáticas equivalentes. Neste trabalho, adota-se a formulação corrente-vorticidade, cujo principal objetivo é eliminar a necessidade do cálculo explícito do campo de pressão, simplificando numericamente o problema enquanto mantém a descrição física completa do escoamento.

A formulação em termos da função de corrente ψ e da vorticidade ω apresenta como principal vantagem o fato de a equação da continuidade ser automaticamente satisfeita pelos campos de velocidade. Adicionalmente, as equações resultantes possuem uma estrutura matemática mais tratável numericamente em comparação com a formulação primitiva das equações de Navier-Stokes.

Consideram-se inicialmente as equações do momento na forma não-conservativa e adimensional para um fluido incompressível:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -\frac{\partial p}{\partial x} + \frac{1}{Re}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \tag{5.1}$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} = -\frac{\partial p}{\partial y} + \frac{1}{Re}\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) \tag{5.2}$$

que representam a conservação de momento na direção x e y, respectivamente. Re (5.8) é o número de Reynolds, que indica o padrão de fluxo de um fluido, p a pressão e u e v são as componentes do vetor velocidade $\mathbf{u}(t,x,y) = (u(t,x,y), v(t,x,y))$ em coordenadas Cartesianas bidimensionais.

A dificuldade fundamental nesta formulação reside no tratamento do termo de pressão, que aparece acoplado às equações do momento sem possuir uma equação de evolução própria. Para contornar esse obstáculo, desenvolvem-se formulações alternativas que eliminam a pressão do sistema. Entre estas, a formulação corrente-vorticidade.

Para escrever as equações acima na formulação $\psi - \omega$, calcula-se

$$\frac{\partial}{\partial x}(\text{equação (5.2)}) - \frac{\partial}{\partial y}(\text{equação (5.1)}),$$

o que resulta em

$$\begin{aligned} \frac{\partial}{\partial t}\left(\frac{\partial u}{\partial y} - \frac{\partial v}{\partial x}\right) + u\frac{\partial}{\partial x}\left(\frac{\partial u}{\partial y} - \frac{\partial v}{\partial x}\right) + v\frac{\partial}{\partial y}\left(\frac{\partial u}{\partial y} - \frac{\partial v}{\partial x}\right) = \\ = \frac{1}{Re}\left[\frac{\partial^2}{\partial x^2}\left(\frac{\partial u}{\partial y} - \frac{\partial v}{\partial x}\right) + \frac{\partial^2}{\partial y^2}\left(\frac{\partial u}{\partial y} - \frac{\partial v}{\partial x}\right)\right]. \end{aligned} \tag{5.3}$$

O termo

$$\frac{\partial u}{\partial y} - \frac{\partial v}{\partial x} \tag{5.4}$$

é conhecido como vorticidade ω. A vorticidade é uma medida de rotação de um elemento de fluido em torno de um ponto. Em termos da vorticidade ω, é possível escrever a equação (5.3) como

$$\frac{\partial \omega}{\partial t} + u\frac{\partial \omega}{\partial x} + v\frac{\partial \omega}{\partial y} = \frac{1}{Re}\left(\frac{\partial^2 \omega}{\partial x^2} + \frac{\partial^2 \omega}{\partial y^2}\right), \tag{5.5}$$

que é uma equação de advecção-difusão para a vorticidade do fluido.

A equação da continuidade pode ser identicamente satisfeita pelo campo de velocidades ao se definir uma função corrente ψ tal que

$$u = \frac{\partial \psi}{\partial y} \quad \text{e} \quad v = -\frac{\partial \psi}{\partial x}. \tag{5.6}$$

Substituindo-se as definições para u e v na definição de vorticidade (5.4), tem-se

$$\frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = -\omega \tag{5.7}$$

que é uma equação de Poisson para função corrente. A partir do instante inicial t_0, e com as condições auxiliares apropriadas, as equações (5.5) e (5.7) podem ser utilizadas para a determinação do campo de velocidades do fluido.

5.2 Problema da Cavidade *Lid-Driven*

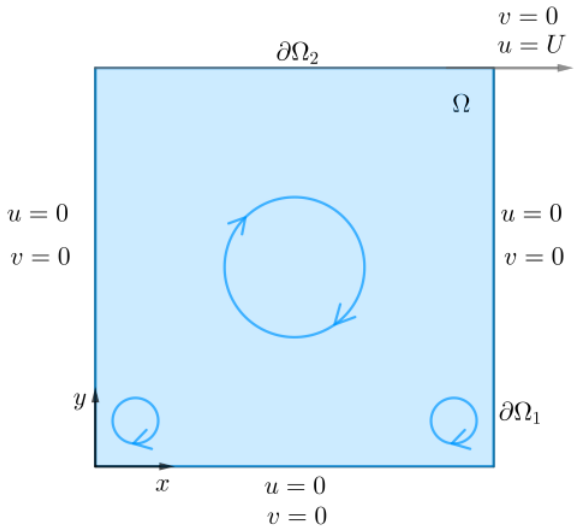


Figura 5.1: Esboço da geometria para um escoamento forçado pelo movimento da tampa de uma caixa (cavidade).

Considere a cavidade mostrada na Figura 5.1. O interior da cavidade é totalmente preenchido com fluido formando o interior do domínio Ω. Inicialmente, tanto a tampa da cavidade como o fluido estão em repouso. As paredes são sólidas e impermeáveis, formando o desenho de uma caixa, que representa a fronteira do domínio $\partial\Omega_1$. No instante t_0 a tampa da cavidade ($\partial\Omega_2$) é instantaneamente acelerada para a velocidade $U > 0$. Devido a tensões viscosas, o movimento da tampa "puxa" o fluido que está adjacente a ela, originando o escoamento.

Em função de U, da altura H da cavidade e da viscosidade ν cinemática do fluido, o número de Reynolds que caracteriza esse escoamento é definido como

$$Re = \frac{UH}{\nu}. \tag{5.8}$$

Condições de Contorno e Inicial

As condições iniciais e de contorno para $\mathbf{u}(x, y, t) = (u(x, y, t), v(x, y, t))$ são

$$\begin{cases} \mathbf{u}(t, x, y) = 0, & \text{em } \Omega; \\ \mathbf{u}(t, x, y) = 0, & \text{em } \partial\Omega_1 \times [0, t_f]; \\ u(t, x, y) = U, & \text{em } \partial\Omega_2 \times [0, t_f]; \\ v(t, x, y) = 0, & \text{em } \partial\Omega_2 \times [0, t_f]. \end{cases}$$

As condições de contorno para ψ foram desenvolvidas com base nas condições de u e v nas paredes, que serão nominadas *Top, Left, Right, Bottom*, decorrentes das suas posições conforme a Figura 5.1.

1. Parede *Bottom*

A condição nessa fronteira é dada por $\psi(x, y, t) = 0$ e, ao considerar a impermeabilidade da caixa, $\omega = -\dfrac{\partial^2\psi}{\partial y^2}$. Aplicando diferenças centrais para a derivada de segunda ordem,

$$\omega_{i,j} = -\frac{\psi_{i,j+1} - 2\psi_{i,j} + \psi_{i,j-1}}{h^2}. \tag{5.9}$$

Note que o termo $\psi_{i,j-1}$, cujo ponto (x_i, y_{j-1}) não pertence ao domínio, precisa ser substituído para que o cálculo seja possível ser efetuado. Para tanto, substitui-se esse termo fantasma por uma expressão de diferenças finitas de terceira ordem obtida por expansão em Série de Taylor:

$$u = \left.\frac{\partial\psi}{\partial y}\right|_j = k_b = \frac{2\psi_{i,j-1} + 3\psi_{i,j} - 6\psi_{i,j+1} + \psi_{i,j+2}}{6h} \Rightarrow \psi_{i,j-1} = \frac{6hk_b - 3\psi_{i,j} + 6\psi_{i,j+1} - \psi_{i,j+2}}{2},$$

no qual k_b é uma constante que representa a velocidade da parede. Substituindo a expressão em 5.9, obtém-se

$$\omega_{i,j} = -\frac{\psi_{i,j+1} - 2\psi_{i,j} + \dfrac{6hk_b - 3\psi_{i,j} + 6\psi_{i,j+1} - \psi_{i,j+2}}{2}}{h^2} \Rightarrow$$

$$\Rightarrow \omega_{i,j} = -\frac{8\psi_{i,j+1} - 7\psi_{i,j} - \psi_{i,j+2} + 6hk_b}{2h^2}.$$

2. Parede *Left*

A condição nessa fronteira é dada por $\psi(x, y, t) = 0$ e, ao considerar a impermeabilidade da caixa, $\omega = -\dfrac{\partial^2\psi}{\partial x^2}$. Aplicando diferenças centrais para a derivada de segunda ordem,

$$\omega_{i,j} = -\frac{\psi_{i+1,j} - 2\psi_{i,j} + \psi_{i-1,j}}{h^2}. \tag{5.10}$$

Note que o termo $\psi_{i-1,j}$, cujo ponto (x_{i-1}, y_j) não pertence ao domínio, precisa ser substituído para que o cálculo seja possível ser efetuado. Para tanto, substitui-se esse termo fantasma por uma expressão de diferenças finitas de terceira ordem obtida por expansão em Série de Taylor:

$$k_l = \frac{2\psi_{i-1,j} + 3\psi_{i,j} - 6\psi_{i+1,j} + \psi_{i+2,j}}{6h} \Rightarrow \psi_{i-1,j} = \frac{6hk_l - 3\psi_{i,j} + 6\psi_{i+1,j} - \psi_{i+2,j}}{2},$$

no qual k_l é uma constante que representa a velocidade da parede. Substituindo a expressão em 5.10, obtém-se

$$\omega_{i,j} = -\frac{\psi_{i+1,j} - 2\psi_{i,j} + \dfrac{6hk_l - 3\psi_{i,j} + 6\psi_{i+1,j} - \psi_{i+2,j}}{2}}{h^2} \Rightarrow$$

$$\Rightarrow \omega_{i,j} = -\frac{8\psi_{i+1,j} - 7\psi_{i,j} - \psi_{i+2,j} + 6hk_l}{2h^2}.$$

3. Parede *Right*

A condição nessa fronteira é dada por $\psi(x, y, t) = 0$ e, ao considerar a impermeabilidade da caixa, $\omega = -\dfrac{\partial^2\psi}{\partial x^2}$. Aplicando diferenças centrais para a derivada de segunda ordem,

$$\omega_{i,j} = -\frac{\psi_{i+1,j} - 2\psi_{i,j} + \psi_{i-1,j}}{h^2}. \tag{5.11}$$

Note que o termo $\psi_{i-1,j}$, cujo ponto (x_{i+1}, y_j) não pertence ao domínio, precisa ser substituído para que o cálculo seja possível ser efetuado. Para tanto, substitui-se esse termo

fantasma por uma expressão de diferenças finitas de terceira ordem obtida por expansão em Série de Taylor:

$$k_r = \frac{2\psi_{i+1,j} + 3\psi_{i,j} - 6\psi_{i-1,j} + \psi_{i-2,j}}{6h} \Rightarrow \psi_{i+1,j} = \frac{6hk_r - 3\psi_{i,j} + 6\psi_{i-1,j} - \psi_{i-2,j}}{2},$$

no qual k_r é uma constante que representa a velocidade da parede. Substituindo a expressão em 5.11, obtém-se

$$\omega_{i,j} = -\frac{\frac{6hk_r - 3\psi_{i,j} + 6\psi_{i-1,j} - \psi_{i-2,j}}{2} - 2\psi_{i,j} + \psi_{i-1,j}}{h^2} \Rightarrow$$

$$\Rightarrow \omega_{i,j} = -\frac{8\psi_{i-1,j} - 7\psi_{i,j} - \psi_{i-2,j} + 6hk_r}{2h^2}.$$

4. Parede *Top*

A condição nessa fronteira é dada por $\psi(x, y, t) = 0$ e, ao considerar a impermeabilidade da caixa, $\omega = -\frac{\partial^2 \psi}{\partial y^2}$. Aplicando diferenças centrais para a derivada de segunda ordem,

$$\omega_{i,j} = -\frac{\psi_{i,j+1} - 2\psi_{i,j} + \psi_{i,j-1}}{h^2}. \tag{5.12}$$

Note que o termo $\psi_{i,j+1}$, cujo ponto (x_i, y_{j+1}) não pertence ao domínio, precisa ser substituído para que o cálculo seja possível ser efetuado. Para tanto, substitui-se esse termo fantasma por uma expressão de diferenças finitas de terceira ordem obtida por expansão em Série de Taylor:

$$k_t = \frac{2\psi_{i,j+1} + 3\psi_{i,j} - 6\psi_{i,j-1} + \psi_{i,j-2}}{6h} \Rightarrow \psi_{i,j+1} = \frac{6hk_b - 3\psi_{i,j} + 6\psi_{i,j-1} - \psi_{i,j-2}}{2},$$

no qual k_t é uma constante que representa a velocidade da parede. Substituindo a expressão em 5.12, obtém-se

$$\omega_{i,j} = -\frac{\frac{6hk_b - 3\psi_{i,j} + 6\psi_{i,j-1} - \psi_{i,j-2}}{2} - 2\psi_{i,j} + \psi_{i,j-1}}{h^2} \Rightarrow$$

$$\Rightarrow \omega_{i,j} = -\frac{8\psi_{i,j-1} - 7\psi_{i,j} - \psi_{i,j-2} + 6hk_t}{2h^2}.$$

Implementação no Devito

Diferentemente dos problemas anteriores que envolviam uma única equação diferencial parcial, o escoamento na cavidade *lid-driven* requer a resolução acoplada de um sistema de quatro equações: vorticidade (5.5), função corrente (5.7) e as duas componentes de velocidade (5.6). Esta complexidade adicional demanda uma estratégia de implementação mais elaborada, baseada na criação de uma função personalizada que encapsula todo o sistema para ser executada pelo `Operator`.

A função `vorticity_operator` implementa essa abordagem:

```
1 def vorticity_operator(omega, psi, u, v, Re, grid, nx):
2     grid = grid
3     t = grid.stepping_dim
4     x, y = grid.dimensions
```

```
    U = 1
    nx = nx
    ny = nx

    k_b = 0    # velocidade da parede bottom
    k_t = U    # velocidade da parede top
    k_l = 0    # velocidade da parede left
    k_r = 0    # velocidade da parede right

    # Equação da vorticidade (4.5)
    pde_omega = Eq(omega.dt + u * omega.dxc + v * omega.dyc - (1/Re) * (
    omega.dx2 + omega.dy2))
    stencil_omega = solve(pde_omega, omega.forward)
    update_omega = Eq(omega.forward, stencil_omega, subdomain=grid.interior
    )

    # Equação da corrente (4.7)
    pde_psi = Eq(psi.dx2 + psi.dy2 + omega)
    stencil_psi = solve(pde_psi, psi)
    update_psi = Eq(psi.forward, stencil_psi, subdomain=grid.interior)

    # Equação das velocidades  (4.6)
    eq_u = Eq(u, psi.dyc)
    eq_v = Eq(v, -psi.dxc)
    stencil_u = solve(eq_u, u)
    stencil_v = solve(eq_v, v)
    update_u = Eq(u.forward, stencil_u, subdomain = grid.interior)
    update_v = Eq(v.forward, stencil_v, subdomain = grid.interior)

    # Condições de Contorno
    # Vorticidade
    bc_omega = [Eq(omega[t+1, 0, y], -(8*psi[t+1, 1, y] - 7*psi[t+1,0,y] -
    psi[t+1,2,y] + 6*x.spacing* k_l)/(2* x.spacing**2))]   # left
    bc_omega += [Eq(omega[t+1, nx-1, y], -(8*psi[t+1, nx-2, y] - 7*psi[t+1,
    nx-1,y] - psi[t+1,nx-3,y] + 6*x.spacing* k_r)/(2* x.spacing**2))]  #
    right
    bc_omega += [Eq(omega[t+1, x, 0], -(8*psi[t+1, x, 1] - 7*psi[t+1,x,0] -
     psi[t+1,x,2] + 6*y.spacing* k_b)/(2* y.spacing**2) )]   # bottom
    bc_omega += [Eq(omega[t+1, x, ny-1], -(8*psi[t+1, x, ny-2] - 7*psi[t+1,
    x,ny-1] - psi[t+1,x,ny-3] + 6*y.spacing* k_t)/(2* y.spacing**2))] # top

    # Corrente
    bc_psi = [Eq(psi[t+1, 0, y], 0)]      # left
    bc_psi += [Eq(psi[t+1, nx-1, y], 0)]  # right
    bc_psi += [Eq(psi[t+1, x, 0], 0)]     # bottom
    bc_psi += [Eq(psi[t+1,x,ny-1], 0)]    # top

    # Velocidade nas paredes
    bc_u = [Eq(u[t+1, 0, y], 0)]        # left
    bc_u += [Eq(u[t+1, nx-1, y], 0)]    # right
    bc_u += [Eq(u[t+1, x, 0], 0)]       # bottom
    bc_u += [Eq(u[t+1, x, ny-1], U)]  # top

    bc_v = [Eq(v[t+1, 0, y], 0)]      # left
    bc_v += [Eq(v[t+1, nx-1, y], 0)]  # right
    bc_v += [Eq(v[t+1, x, 0], 0)]     # bottom
    bc_v += [Eq(v[t+1, x, ny-1], 0)]  # top

    bc = bc_omega + bc_psi + bc_u + bc_v

```

```
    # Operador principal
    op = Operator([update_omega, update_psi, update_u, update_v] + bc)

    return op
```

Um aspecto crucial na implementação são as condições de contorno para a vorticidade, que utilizam as fórmulas de diferenças finitas de alta ordem derivadas anteriormente. Os termos `x.spacing` e `y.spacing` representam os espaçamentos da malha nas direções x e y, respectivamente, garantindo uma implementação genérica independente da resolução.

Quanto à discretização da equação de vorticidade, foram utilizadas as aproximações centradas `omega.dxc` e `omega.dyc` para os termos convectivos. Embora diferenças centradas possam introduzir erros dispersivos e oscilações numéricas não físicas em problemas dominados pela convecção, tais efeitos não foram observados de forma significativa nos resultados obtidos neste trabalho.

Idealmente, a discretização deveria ser capaz de avaliar localmente o sinal de u e v e selecionar dinamicamente a direção do estêncil para a discretização de w. No Devito, tal comportamento poderia ser implementado utilizando `ConditionalDimension`, que permite a criação de expressões condicionais dependentes do valor das variáveis no domínio. Contudo, durante o desenvolvimento deste trabalho, identificou-se um erro ainda não solucionado na versão atual do Devito: quando uma `ConditionalDimension` é fornecida através do parâmetro `implicit_dims` em conjunto com `subdomain=grid.interior`, o operador falha na compilação.

A correção deste *bug* no Devito constitui, portanto, uma direção importante para trabalhos futuros, pois permitiria a implementação correta do esquema *upwind* (ou de esquemas de maior ordem, como CUBISTA [5]) e, consequentemente, uma melhoria significativa na precisão das simulações para regimes de escoamento com maiores números de Reynolds.

Em seguida, cria-se uma outra função para a execução da simulação:

```
def run_simulation(nx, Re, alpha, Tmax):
  Lx = 1
  Ly = 1
  ny = nx
  dx = Lx / (nx - 1)
  dy = Ly / (ny - 1)

  dt_visc =  (Re/2) * 1/(1/dx**2 + 1/dy**2)
  dt_cfl = min(dx,dy)
  dt =  alpha * min(dt_visc, dt_cfl)

  nt = int(Tmax/dt)

  x_np = np.linspace(0, Lx, nx)
  y_np = np.linspace(0, Ly, ny)
  X,Y = np.meshgrid(x_np,y_np)
  grid = Grid(shape=(nx, ny), extent=(Lx, Ly))

  w = TimeFunction(name='omega', grid=grid, space_order=2)
  psi = TimeFunction(name='psi', grid=grid, space_order=2)
  u = TimeFunction(name='u', grid=grid, space_order=2)
  v = TimeFunction(name='v', grid=grid, space_order=2)

  # Condições Iniciais
  w.data[0] = np.ones((nx, ny))
  psi.data[0] = np.zeros((nx, ny))
  u.data[0] = np.zeros((nx, ny))
  v.data[0] = np.zeros((nx, ny))
```

```
29
30  # Operador
31  configuration['log-level'] = 'ERROR'
32  op = vorticity_operator(w, psi, u, v, Re, grid, nx)
33  op.apply(omega=w, psi=psi, u=u, v=v, dt=dt, time_M=nt-2)
34
35  n = 8
36  fig = plt.figure(figsize=(7, 5), dpi=100)
37  # Preenchimento com cor
38  plt.contourf(X, Y, psi.data[0].T, levels=900, alpha=0.7, cmap=plt.cm.
      rainbow)
39  plt.colorbar()
40  # Linhas de Contorno
41  plt.contour(X, Y, psi.data[0].T, levels= 10, cmap=cm.viridis)
42  # Vetores da Velocidade
43  plt.quiver(X[::n, ::n], Y[::n, ::n], u.data[0,::n, ::n].T, v.data[0,::n,
      ::n].T)
44  plt.xlabel('x')
45  plt.ylabel('y')
46  plt.show()
47
48  fig, ax = plt.subplots(figsize=(4, 4), dpi=100)
49
50  # Contornos apenas com linhas, sem preenchimento
51  contours = ax.contour(X, Y, psi.data[0].T, levels=20, colors='k',
      linewidths=0.8)
52  ax.clabel(contours, inline=True, fontsize=8) # Adiciona rótulos às linhas
      de contorno
53  ax.set_xlabel('x')
54  ax.set_ylabel('y')
55  #ax.set_title('Linhas de Contorno')
56
57  plt.tight_layout()
58  plt.show()
59
60  return w, psi, u, v
```

As equações de vorticidade, função corrente e velocidades são reunidas em um único objeto `Operator`. A partir da lista de expressões simbólicas fornecida, o compilador do Devito infere automaticamente as dependências entre as variáveis, reorganiza as operações quando necessário e aplica sucessivas etapas de otimização. Como resultado, é gerado um único kernel computacional em linguagem C, compilado em tempo de execução (*Just-In-Time*), mantendo a ordem correta de atualização e o acoplamento entre as variáveis.

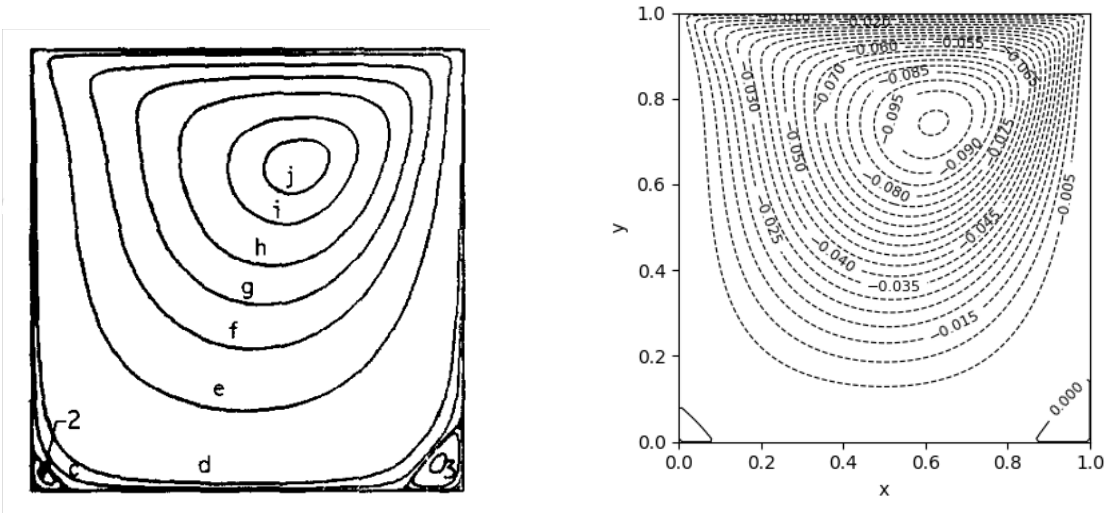
A execução da simulação ocorre por meio do método `op.apply`, que dispara esse kernel a cada passo temporal. Nesse processo, o Devito não forma matrizes explícitas, pois adota computação baseada em estênceis livres de matriz (*matrix-free*), avaliando diretamente as expressões discretizadas no domínio.

Resultados e Análises

Para a simulação, adota-se $U = 1$ como velocidade característica em um domínio $\Omega = [0, 1] \times [0, 1]$. Embora o estudo de referência [12] utilize um critério de parada baseado na diferença entre soluções em instantes consecutivos, tal funcionalidade não está implementada no Devito. Diante desta limitação, optou-se por um tempo final de simulação $t_f = 20$, determinado empiricamente como suficiente para que a solução numérica atinja um regime quasi-estacionário. Testes adicionais demonstraram que aumentos poste-

riores de t_f produzem variações desprezíveis na solução, validando a adequação do tempo de simulação escolhido.

Para uma análise qualitativa, observe as seguintes figuras.



(a) Solução de referência. Fonte: [12].

(b) Solução do Devito.

Figura 5.2: Contorno de ψ para $Re = 100$ e $nx = 129$.

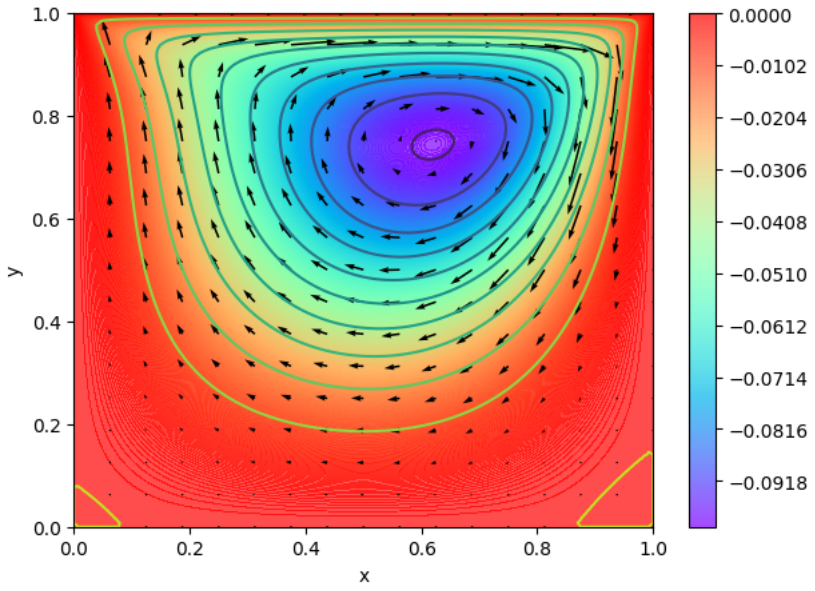
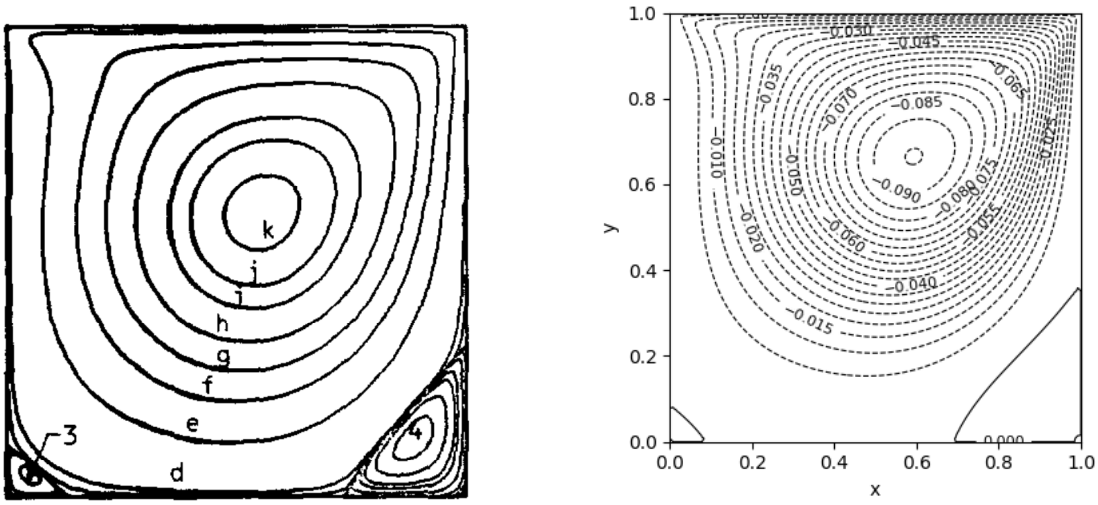


Figura 5.3: Plotagem de ψ com os vetores da velocidade u e v para $Re = 100$.

Nota-se que a solução numérica obtida com o Devito apresenta concordância qualitativa com os resultados de Ghia, particularmente na captura dos vórtices primários e na estrutura geral do escoamento. Entretanto, observa-se que o Devito apresenta limitações na reprodução adequada dos vórtices secundários de menor intensidade. Este comportamento também é verificado para Reynolds 400 com malha de 129 pontos, no qual o método não resolve completamente a complexidade dessas estruturas vorticais.

É importante destacar que, embora a análise qualitativa sugira valores próximos de zero nessas regiões de vórtices secundários, os resultados de referência indicam que a solução nessas áreas não é estritamente nula, mas sim caracterizada por valores de vorticidade significativamente menores em magnitude quando comparados aos vórtices primários.



(a) Solução de referência. Fonte: [12].

(b) Solução do Devito.

Figura 5.4: Contorno de ψ para $Re = 400$ e $nx = 129$.

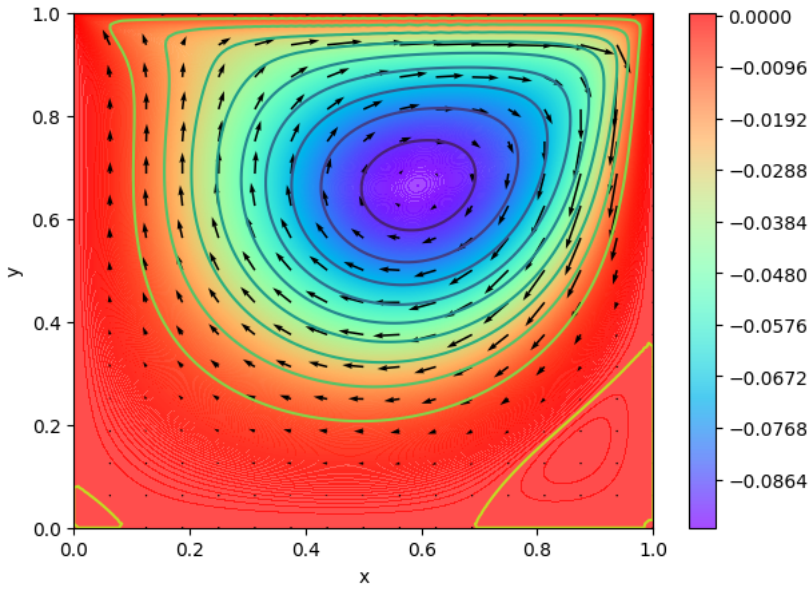


Figura 5.5: Plotagem de ψ com os vetores da velocidade u e v para $Re = 400$.

Para uma avaliação quantitativa do desempenho do Devito na simulação do escoamento na cavidade *lid-driven*, os resultados numéricos são comparados com os dados de referência de Ghia através das Tabelas 5.1 e 5.2. A análise dos perfis de velocidade u no corte vertical central e v no corte horizontal central revela padrões importantes sobre a precisão do método.

Tabela 5.1: Comparação dos resultados para u em um corte central ($x = 0.5$) para uma malha com $nx = 129$.

y	$Re = 100$			$Re = 400$		
	Ghia	Devito	Erro	Ghia	Devito	Erro
1	1	1	0	1	1	0
0.9688	0.78871	0.78340	0.00531	0.68439	0.58757	0.09682
0.8516	0.23151	0.22155	0.00996	0.29093	0.24093	0.05000
0.6172	−0.13641	−0.14281	0.00640	0.02135	−0.05717	0.07852
0.5	−0.20581	−0.20324	0.00257	−0.11477	−0.20438	0.08961
0.2813	−0.15662	−0.14630	0.01032	−0.32726	−0.18978	0.13748
0.0703	−0.04192	−0.04279	0.00087	−0.10338	−0.02740	0.07598
0	0	0	0	0	0	0

Para $Re = 100$, observa-se uma excelente concordância entre os resultados do Devito e os dados de referência. Os erros na componente u (Tabela 5.1) mantêm-se na ordem de 10^{-3} a 10^{-2}, com destaque para a região central ($y = 0.5$) onde o erro é de apenas 0.00257. Similarmente, para a componente v (Tabela 5.2), os erros são consistentemente baixos, variando entre 0.00061 e 0.00591, demonstrando a capacidade do Devito em capturar adequadamente a física do escoamento em regime laminar de baixo Reynolds.

Entretanto, para $Re = 400$, observa-se um aumento significativo nos erros, particularmente nas regiões de maiores gradientes de velocidade. Na componente u, os erros atingem valores de até 0.13748 na região de $y = 0.2813$, enquanto na componente v os erros chegam a 0.14053 em $x = 0.8594$. Esse comportamento é consistente com as limitações observadas qualitativamente na reprodução dos vórtices secundários, sugerindo que o esquema numérico empregado pode apresentar limitações na resolução de estruturas de menor escala em regimes de maior Reynolds.

Tabela 5.2: Comparação dos resultados para v em um corte central ($y = 0.5$) para uma malha com $nx = 129$.

x	$Re = 100$			$Re = 400$		
	Ghia	Devito	Erro	Ghia	Devito	Erro
1	0	0	0	0	0	0
0.9688	−0.05906	−0.05967	0.00061	−0.12146	−0.06302	0.05844
0.8594	−0.22445	−0.22195	0.00250	−0.44993	−0.30940	0.14053
0.5	0.05454	0.05534	0.00080	0.05186	0.08607	0.03421
0.2813	0.17527	0.16953	0.00574	0.30174	0.18414	0.11760
0.1563	0.16077	0.15486	0.00591	0.28124	0.15317	0.12807
0.0703	0.10091	0.09665	0.00426	0.19713	0.09665	0.10048
0	0	0	0	0	0	0

Nota-se que os maiores erros concentram-se nas regiões de transição e próximo aos vórtices secundários, onde os gradientes de velocidade são mais acentuados. As regiões de escoamento principal e próximas às paredes apresentam melhor concordância, com erros inferiores a 0.01 mesmo para $Re = 400$.

Esses resultados quantitativos corroboram as observações qualitativas anteriores, indicando que o Devito produz soluções precisas para escoamentos de baixo Reynolds, mas pode requerer esquemas numéricos mais avançados ou malhas mais refinadas para capturar adequadamente a complexidade de escoamentos em regimes de Reynolds mais elevados. No entanto, esta constatação merece uma investigação mais aprofundada.

Para avaliar o comportamento do método em regimes de alta turbulência, considera-se agora o escoamento com $Re = 10000$ e uma malha significativamente mais refinada com $nx = 257$ pontos. As simulações foram conduzidas com tempo final $t_f = 4000$ e passo temporal $\Delta t = 3,90625 \cdot 10^{-4}$, garantindo estabilidade numérica e convergência para o estado estacionário.

A Figura 5.6 apresenta a solução numérica obtida com o Devito, mostrando o campo de função corrente ψ juntamente com o campo vetorial de velocidades (u, v). Visualmente, a estrutura geral do escoamento primário é bem capturada, incluindo o vórtice principal que domina a cavidade.

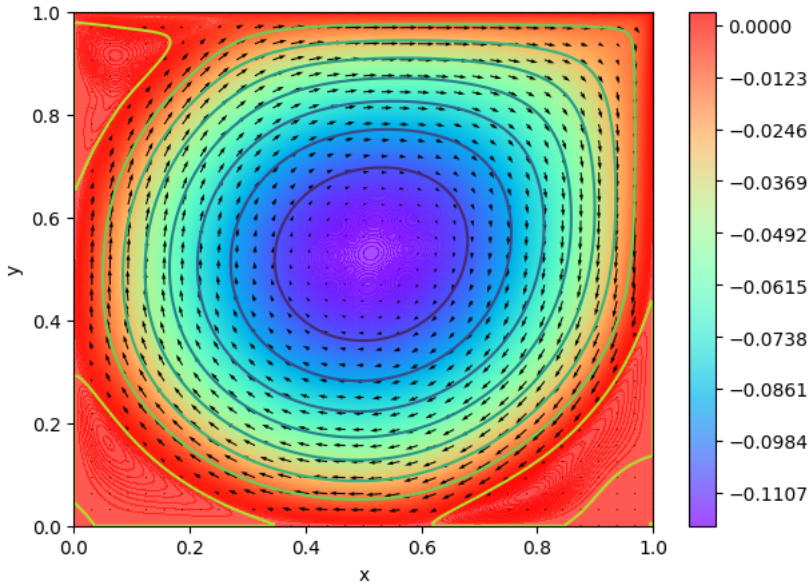
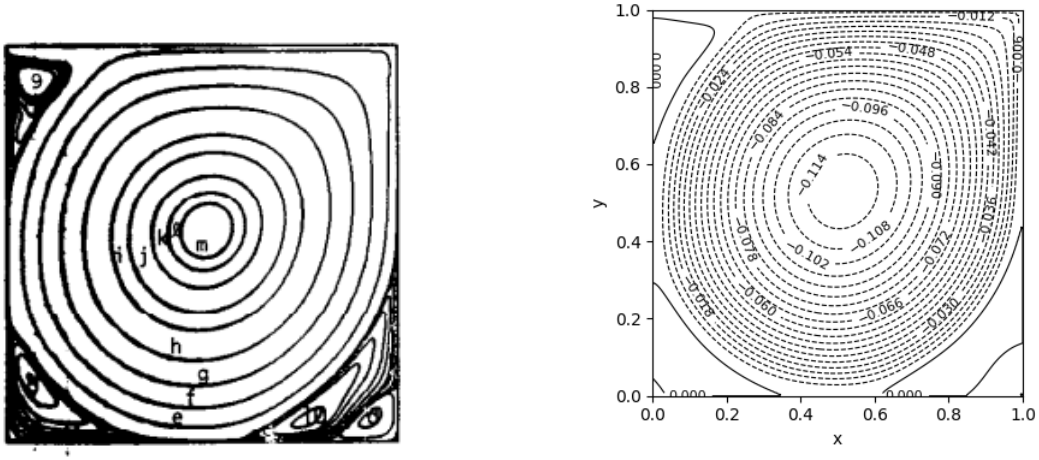


Figura 5.6: Função corrente ψ com vetores de velocidade (u, v) para $Re = 10000$ e malha 257×257.

A Figura 5.7 compara as linhas de contorno de ψ obtidas por Ghia e pelo Devito. Observa-se que, embora os contornos principais sejam razoavelmente reproduzidos, os vórtices secundários nos cantos inferiores da cavidade apresentam diferenças perceptíveis entre as duas soluções.



(a) Solução de referência. Fonte: [12].

(b) Solução obtida com Devito.

Figura 5.7: Comparação das linhas de contorno da função corrente ψ para $Re = 10000$ e malha 257×257.

Uma questão que surge naturalmente é se as discrepâncias observadas nos vórtices secundários decorrem de limitações do método numérico ou de artefatos de visualização. Para investigar esta questão, a Figura 5.8 apresenta um zoom na região do canto inferior esquerdo da cavidade, correspondente ao subdomínio $[0, 0.4] \times [0, 0.4]$, limitando o valor de ψ, tal que $|\psi| < 0.002$ para garantir que os valores de ψ menores que tal sejam plotados devidamente.

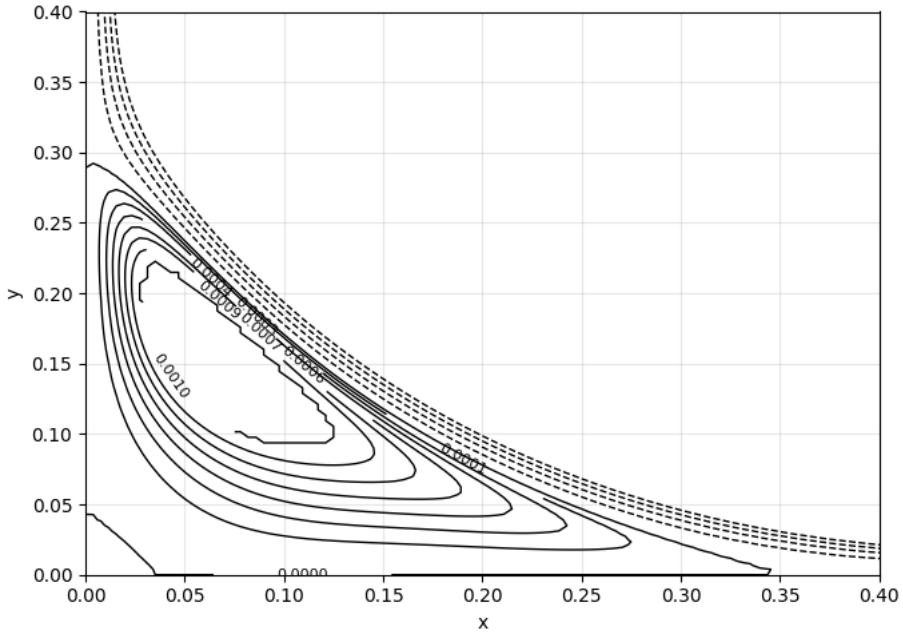


Figura 5.8: Zoom na região do canto inferior esquerdo da cavidade ($[0, 0.4] \times [0, 0.4]$).

A Figura 5.9 mostra a solução do Ghia com um maior foque nos vórtices menores, a imagem fora editada com um pontilhado em $y = 0.1$ para uma comparação mais fácil a seguir.

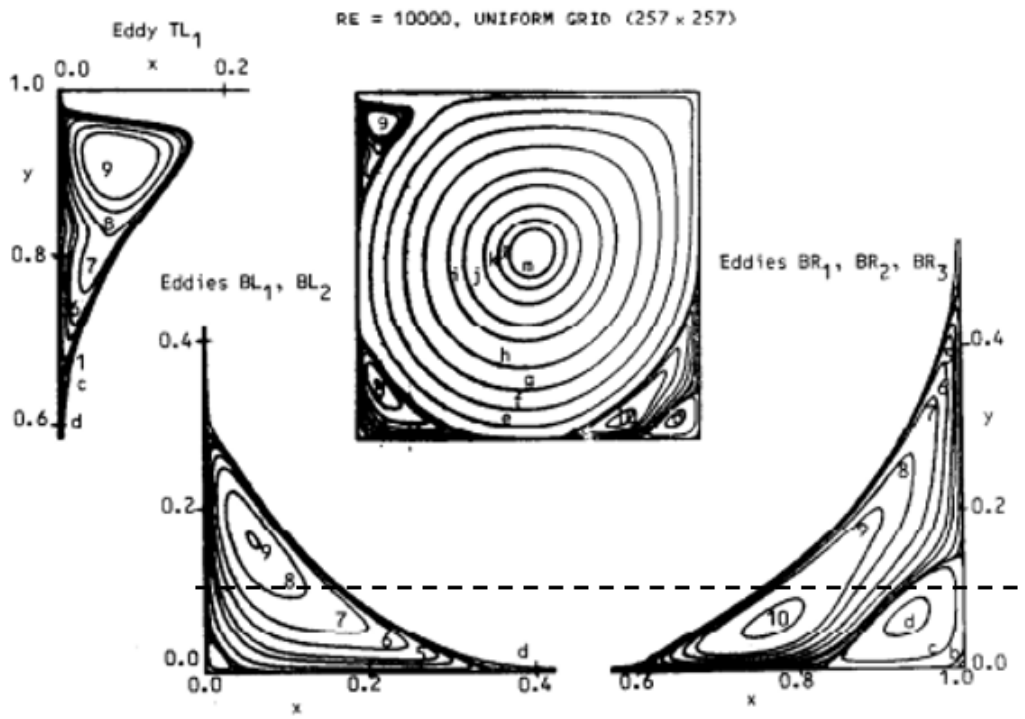


Figura 5.9: Solução de referência editada com um corte pontilhado em $y = 0.1$. Fonte: [12].

Nesta região, os valores da função corrente ψ são extremamente pequenos, da ordem de 10^{-3} a 10^{-4}, conforme documentado na literatura de referência. A Figura 5.10 apresenta os valores tabulados por para os contornos da função corrente ψ e da vorticidade ω.

Stream function				Vorticity	
Contour letter	Value of ψ	Contour number	Value of ψ	Contour number	Value of ω
a	-1.0×10^{-10}	0	1.0×10^{-8}	0	0.0
b	-1.0×10^{-7}	1	1.0×10^{-7}	±1	±0.5
c	-1.0×10^{-5}	2	1.0×10^{-6}	±2	±1.0
d	-1.0×10^{-4}	3	1.0×10^{-5}	±3	±2.0
e	−0.0100	4	5.0×10^{-5}	±4	±3.0
f	−0.0300	5	1.0×10^{-4}	5	4.0
g	−0.0500	6	2.5×10^{-4}	6	5.0
h	−0.0700	7	5.0×10^{-4}		
i	−0.0900	8	1.0×10^{-3}		
j	−0.1000	9	1.5×10^{-3}		
k	−0.1100	10	3.0×10^{-3}		
l	−0.1150				
m	−0.1175				

Figura 5.10: Valores de referência das Figura 5.7(a) e 5.9 para função corrente ψ e vorticidade ω. Fonte: [12].

Para uma análise quantitativa adicional, examina-se o perfil da função corrente ψ em um corte horizontal próximo à parede inferior da cavidade ($y \approx 0.1$), restrito ao subdomínio $x \in [0, 0.4]$ para melhor visualização dos vórtices secundários. Os valores

obtidos, apresentados na Tabela 5.3, evidenciam a variação de ψ ao longo da direção x nesta região.

Tabela 5.3: Perfil da solução numérica de ψ no corte $y \approx 0,1$ para $Re = 10000$ e $nx = 257$ (subdomínio $x \in [0, 0.4]$).

x	ψ
0.0078	$1.5265 \cdot 10^{-5}$
0.0234	$1.5295 \cdot 10^{-4}$
0.0312	$2.7336 \cdot 10^{-4}$
0.0469	$5.7028 \cdot 10^{-4}$
0.0664	$9.1129 \cdot 10^{-4}$
0.1016	$1.1273 \cdot 10^{-3}$
0.1562	$4.8867 \cdot 10^{-4}$
0.1680	$4.8229 \cdot 10^{-5}$

Diante desses valores extremamente pequenos (da ordem de 10^{-5} a 10^{-3}) para os vórtices secundários, conclui-se que as dificuldades de visualização observadas são, em grande parte, decorrentes da escala dos contornos e não necessariamente de limitações do método numérico. O Devito é capaz de capturar essas estruturas, porém a resolução gráfica padrão não consegue representá-las adequadamente devido à sua magnitude reduzida.

Diante desses valores extremamente pequenos (da ordem de 10^{-10} a 10^{-4}) para os vórtices secundários, conclui-se que as dificuldades de visualização observadas nas Figuras 5.6 e 5.7 são, em grande parte, decorrentes da escala dos contornos e não necessariamente de limitações do método numérico. O Devito é capaz de capturar essas estruturas, porém a resolução gráfica padrão não consegue representá-las adequadamente devido à sua magnitude reduzida.

Capítulo

6

Conclusão

Este trabalho investigou a aplicabilidade da plataforma Devito na resolução numérica de escoamentos viscosos incompressíveis utilizando a formulação corrente-vorticidade ($\psi - \omega$) das equações de NavierStokes. Os resultados obtidos ao longo do trabalho demonstraram que a plataforma é capaz de representar adequadamente problemas clássicos de Dinâmica dos Fluidos Computacional por diferenças finitas, reproduzindo soluções manufaturadas e padrões de escoamento amplamente documentados na literatura.

A aplicação da metodologia ao problema da cavidade *lid-driven* para diferentes números de Reynolds evidenciou a capacidade do Devito em capturar as principais estruturas do escoamento, apresentando resultados consistentes com referências clássicas da área. Esses resultados confirmam o potencial da plataforma como ferramenta para prototipagem e desenvolvimento de métodos numéricos baseados em diferenças finitas.

Além da análise numérica, o trabalho também permitiu investigar aspectos computacionais relevantes associados ao uso do Devito em problemas envolvendo equações elípticas e sistemas acoplados. Observou-se que a estratégia de pseudo-transiente utilizada na solução da equação de Poisson, embora funcional e relativamente simples de implementar, apresenta limitações de desempenho em malhas refinadas devido ao elevado número de iterações necessárias para convergência.

Nesse contexto, a integração experimental entre o Devito e a biblioteca PETSc mostrou-se particularmente promissora. Os testes realizados indicaram ganhos expressivos de desempenho na solução da equação de Poisson, evidenciando o potencial da combinação entre geração automática de código e *solvers* lineares especializados de alta performance. Embora a integração ainda apresente desafios técnicos e limitações de implementação, os resultados obtidos sugerem que essa abordagem constitui uma direção relevante para futuras aplicações envolvendo problemas de maior porte computacional.

O trabalho também permitiu identificar limitações atuais da plataforma relacionadas à implementação de discretizações convectivas com seleção dinâmica de direção, necessárias em esquemas *upwind* e em métodos convectivos mais sofisticados. Tais limitações evidenciam que a utilização do Devito em problemas de Dinâmica dos Fluidos Computacional ainda demanda avanços em determinados recursos internos da ferramenta. Ao mesmo tempo, esses aspectos abrem perspectivas para o desenvolvimento de novas estratégias de discretização e implementação adaptadas à arquitetura simbólica da plataforma.

De maneira geral, os resultados obtidos indicam que o Devito constitui uma ferramenta promissora para aplicações em Dinâmica dos Fluidos Computacional baseadas em diferenças finitas, especialmente quando associado a bibliotecas externas especializadas, como o PETSc. Espera-se que os avanços recentes da plataforma e o amadurecimento de suas funcionalidades permitam ampliar sua utilização em problemas envolvendo escoamentos

mais complexos, métodos numéricos de maior ordem e arquiteturas computacionais de alto desempenho.

Estudos Futuros

Como perspectivas futuras, destacam-se diferentes possibilidades de expansão e aprofundamento da pesquisa desenvolvida neste trabalho. Entre elas, pode-se citar a incorporação de pré-condicionadores na resolução dos sistemas lineares, visando aumentar ainda mais a eficiência computacional da integração Devito-PETSc em simulações de grande porte. Além disso, a extensão da metodologia para problemas tridimensionais constitui uma direção natural para avaliar a robustez e a escalabilidade da abordagem em cenários mais complexos de Dinâmica dos Fluidos Computacional.

Outra perspectiva relevante consiste na investigação de geometrias mais complexas e diferentes tipos de condições de contorno, incluindo condições de Neumann e Robin, ampliando o conjunto de aplicações tratáveis pela plataforma. Também se destaca a realização de estudos de escalabilidade paralela utilizando *MPI*, permitindo avaliar quantitativamente o desempenho da metodologia em arquiteturas de computação de alto desempenho.

Referências

[1] M. Badri; F. Sabetghadam. A transformation for imposing the rigid bodies on the solution of the vorticity-stream function formulation of incompressible Navier-Stokes equations. *International Journal of Computational Methods*, 17(9):1950063, 2020.

[2] L.A. Barba; G.F. Forsyth. CFD Python: the 12 steps to Navier-Stokes equations. *Journal of Open Source Education*, 1(9):21, 2018.

[3] B. Belucci. *Modelamento sísmico por diferenças finitas: aplicação do pacote Devito para resolução da equação da onda.* Monografia, UNICAMP, Campinas, SP, Brasil, 2022.

[4] H.A. Castillo-Sánchez; J. Bertoco; M.S.B. Araújo; A. Castelo. Numerical simulation of thixotropic-viscoelastic models for structured fluids in hierarchical grids. *Computers and Fluids*, 266:106045, 2023.

[5] L. Corrêa; G.A.B. Lima; V.G. Ferreira. Metodologia para desenvolvimento de esquemas upwind de alta resolução. 2011.

[6] J.A. Cuminato; M.Jr. Meneguette. *Discretização de equações diferenciais parciais: técnicas de diferenças finitas.* Coleção matemática aplicada. SBM, Rio de Janeiro, 2013.

[7] M.C.C. Cunha. *Métodos Numéricos.* Editora Unicamp, Campinas, SP, 2003.

[8] M.M.G. Dakuzaku; G.S. Paulo; C. Viezel. Soluções numéricas para um problema de convecção-difusão. *Anais do Simpósio de Matemática da FCT/UNESP*, 2022.

[9] Devito Community. Site oficial do Devito project. `https://www.devitoproject.org`, 2023. Acessado em 19/04/2026.

[10] N.M.B. Franco. *Cálculo Numérico.* Pearson Prentice Hall, São Paulo, 2006.

[11] J.E. Fromm. The time dependent flow of an incompressible viscous fluid. *Math. Comput. Phys.*, 3:345–382, 1964.

[12] U. Ghia; K.N. Ghia; C.T. Shin. High-Re Solutions for Incompressible Flow Using the Navier-Stokes Equations and a Multigrid Method. *Journal of Computational Physics*, 48:387–411, 1982.

[13] M. Girfoglio; A. Quaini; G. Rozza. A POD-Galerkin reduced order model for the Navier-Stokes equations in stream function-vorticity formulation. *Computers and Fluids*, 244:105–536, 2022.

[14] T.B. Ikeda. Códigos da dissertação. `https://github.com/ikeda-thiago/Mestrado`, 2026.

[15] Z. Leibowitz. Devito-PETSc-Benchmarks. `https://github.com/ZoeLeibowitz/Devito-PETSc-Benchmarks`, 2025.

[16] Z. Leibowitz; R. Nelson; F. Luporini; M. Louboutin; M. Knepley; L. Mitchell; G. Gorman. Automatic PETSc Code Generation For Finite Differences. SIAM PP24, 2024.

[17] Z. Leibowitz; R. Nelson; F. Luporini; M. Louboutin; M. Knepley; L. Mitchell; J. Betteridge; E. Caunt; G. Bisbas; M. Piggott; G. Gorman. Automatic Generation of Matrix-Free Routines for PDE solvers with Devito via PETSc. PETSc User Meeting, 2025.

[18] R.J. Leveque. *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-state and Time-dependent Problems*. Society for Industrial and Applied Mathematics, Filadélfia, EUA, 2007.

[19] M. Louboutin; M. Lange; F. Luporini; N. Kukreja; P.A. Writte; F.J. Herrmann; P. Velesko; G.J. Gorman. Devito (v3.1.0): an embedded domain-specific language for finite differences and geophysical exploration. *Geoscientific Model Development*, 12(3):1165–1187, 2019.

[20] F. Luporini; M. Loubotin; M. Langel; N. Kukreja; P.A. Writte; J. Hückelheim; C. Yount; P.H.J. Kelly; F.J. Herrmann; G.J. Gorman. Architecture and Performance of Devito, a System for Automated Stencil Computation. *ACM Transactions on Mathematical Software*, 46(1), abril 2020.

[21] C.R. Maliska. *Transferência de calor e mecânica dos fluidos computacional*. LTC, Rio de Janeiro, 2017.

[22] P. Mayeli; G.J. Sheard. Natural convection and entropy generation in square and skew cavities due to large temperature differences: A Gay-Lussac-Type vorticity stream-function approach. *International Journal for Numerical Methods in Fluids*, 93(7):2396–2420, 2021.

[23] J. P. Narain. Lid driven cavity flow: Review and future trends. *American Journal of Fluid Dynamics*, 1:1–15, 2022.

[24] P.J. Oliveira. *Problema de convecção/difusão unidimensional*. Covilhã, 2002.

[25] G.S. Paulo; C. Viezel; L.L. Ferrás. A robust finite difference method for confined and free surface flows with slip at the wall. *Journal of Non-Newtonian Fluid Mechanics*, 321:105127, 2023.

[26] J. Pontes; N. Mangiavacchi. *Fenômenos de transferência com aplicações às ciências físicas e à engenharia*. Coleção Matemática Aplicada. SBM, Rio de Janeiro, 2021.

[27] S. Post. *Mecânica dos fluidos aplicada e computacional*. LTC, 2013.

[28] P.N. Shankar; M.D. Deshpande. Fluid mechanics in the driven cavity. *Annual Review of Fluid Mechanics*, 32:93–136, 2000.