Instituto Politécnico do Estado do Rio de Janeiro

Curso de Engenharia da Computação

Guilherme Cagide Fialho

Análise Comparativa de Soluções Numéricas e Analíticas para as Equações de Advecção e Advecção-Difusão

Nova Friburgo

2024

Instituto Politécnico do Estado do Rio de Janeiro

Graduação em Engenharia da Computação

Guilherme Cagide Fialho

Análise Comparativa de Soluções Numéricas e Analíticas para as Equações de Advecção e Advecção-Difusão

Relatório da Disciplina Métodos Númericos
para Equações Diferenciais 2

Professor: Hélio Pedro Amaral Souto

Nova Friburgo

2024

RESUMO

CAGIDE FIALHO, G. Relatório do projeto de Métodos Numéricos para Equações Diferenciais II. 2024. 16 f. Trabalho da Disciplina Métodos Numéricos para Equações Diferenciais II (Graduação em Engenharia da Computação) – Graduação em Engenharia da Computação, Universidade do Estado do Rio de Janeiro, Nova Friburgo, 2024.

Este trabalho analisa soluções numéricas e analíticas das equações de advecção e advecção-difusão em um domínio, considerando a influência dos parâmetros de velocidade e coeficiente de difusão no transporte e dispersão de substâncias. As soluções analíticas foram obtidas com métodos de características e transformações de variáveis, enquanto as soluções numéricas foram desenvolvidas usando métodos como Upwind, Lax-Wendroff, Beam-Warming e Fromm. Os resultados das simulações numéricas destacam as diferenças entre os modelos de advecção pura e advecção-difusão, demonstrando como a difusão contribui para uma maior homogeneização da concentração ao longo do tempo. A análise comparativa das soluções revela a importância do coeficiente de difusão no comportamento de dispersão do soluto, servindo como uma ferramenta para a modelagem de fenômenos naturais e aplicações práticas em engenharia e ciências ambientais.

Palavras-chave: Métodos Numéricos, Equação de Advecção, Equação de Advecção-Difusão, Características, Simulação Numérica, Dispersão de Soluto.

ABSTRACT

CAGIDE FIALHO, G. Project Report on Numerical Methods for Differential Equations II. 2024. 16 p. Course Completion Work for Numerical Methods for Differential Equations II (Bachelor's in Computer Engineering) – Bachelor's in Computer Engineering, State University of Rio de Janeiro, Nova Friburgo, 2024.

This work analyzes numerical and analytical solutions to advection and advection-diffusion equations in an domain, considering the influence of velocity and diffusion coefficients on the transport and dispersion of substances. Analytical solutions were obtained using characteristic methods and variable transformations, while numerical solutions were developed using methods such as Upwind, Lax-Wendroff, Beam-Warming, and Fromm. The results of numerical simulations highlight the differences between pure advection and advection-diffusion models, demonstrating how diffusion contributes to a more homogeneous concentration over time. The comparative analysis of the solutions reveals the importance of the diffusion coefficient in influencing solute dispersion, serving as a tool for modeling natural phenomena and providing practical applications in engineering and environmental sciences.

Keywords: Numerical Methods, Advection Equation, Advection-Diffusion Equation, Characteristics, Numerical Simulation, Solute Dispersion.

Lista de Figuras

Lista de Tabelas

Lista de Códigos

Sumário

1 Objetivo

O objetivo deste trabalho é investigar e comparar soluções analíticas e numéricas para as equações de advecção e advecção-difusão em um domínio, com foco nos efeitos dos parâmetros de velocidade e difusão sobre o transporte e dispersão de substâncias. Para isso, são aplicados métodos numéricos (Upwind, Lax-Wendroff, Beam-Warming e Fromm) e técnicas analíticas (método das características e transformações de variáveis) a fim de avaliar a eficácia e limitações de cada abordagem. A análise busca oferecer insights sobre a influência do coeficiente de difusão na homogeneização da concentração ao longo do tempo e gerar conhecimento aplicável a contextos práticos, como a modelagem de fenômenos naturais e problemas de engenharia relacionados à dispersão de solutos.

2 Introdução

A equação de advecção é uma ferramenta matemática crucial para modelar o transporte de substâncias em um fluido, sendo amplamente empregada em áreas da engenharia e das ciências aplicadas. Neste trabalho, a equação de advecção unidimensional é resolvida numericamente utilizando o método dos Volumes Finitos com quatro abordagens diferentes: Upwind de primeira ordem, Lax-Wendroff, Beam-Warming e Fromm. Essas abordagens são implementadas para modelar a evolução temporal da concentração de um traçador, com o objetivo de avaliar e comparar as soluções fornecidas por cada método.

Para garantir a estabilidade das simulações, é fixado o número de Courant em $C = 0,8$, respeitando a condição CFL. A condição inicial do traçador é definida por uma combinação de uma função gaussiana e uma função por partes que representa uma concentração constante em um intervalo específico do domínio. Cada método é aplicado para obter a solução numérica da equação nos instantes de tempo $t = 1$, $t = 3$ e $t = 5$, sob condições de contorno periódicas.

Os resultados são visualizados por meio de gráficos que comparam as concentrações obtidas em cada instante com a condição inicial, o que permite observar o comportamento das soluções e a precisão dos métodos. Além disso, as saídas numéricas são salvas em formato de tabelas, facilitando uma análise quantitativa da evolução da concentração ao longo do tempo.

O código feito foi organizado de forma a gerar automaticamente os gráficos e tabelas, armazenando-os em diretórios específicos. Este trabalho, portanto, fornece uma implementação prática e comparativa de métodos numéricos para a solução de problemas de advecção, com foco na análise de precisão e comportamento de cada método ao longo do tempo.

3 Desenvolvimento Teórico

A equação de advecção unidimensional descreve o transporte de uma quantidade conservada, como a concentração de um traçador, ao longo de um eixo espacial. Para resolver essa equação numericamente, é utilizado o método dos Volumes Finitos, que permite a discretização do espaço e do tempo, garantindo uma formulação adequada para a conservação da quantidade transportada. A equação de advecção, em sua forma conservativa, é dada por:

$$\frac{\partial \Phi}{\partial t} + \frac{\partial}{\partial x}(u\Phi) = 0, \tag{1}$$

onde Φ representa a variável dependente (concentração do traçador) e u é a velocidade de advecção. Com u constante, a equação simplifica-se para:

$$\frac{\partial \Phi}{\partial t} + u\frac{\partial \Phi}{\partial x} = 0. \tag{2}$$

Neste trabalho, a solução numérica é obtida utilizando quatro métodos clássicos:

- **Método Upwind de Primeira Ordem**: Este método é conhecido por sua simplicidade e robustez, especialmente em casos onde a solução apresenta descontinuidades. A sua forma discretizada é dada por:

$$Q_i^{n+1} = Q_i^n - C(Q_i^n - Q_{i-1}^n), \tag{3}$$

onde $C = \frac{u\Delta t}{\Delta x}$ é o número de Courant. Esse método calcula o fluxo considerando apenas os valores à montante, garantindo estabilidade para $0 \leq C \leq 1$.

- **Método de Lax-Wendroff**: Este método de segunda ordem melhora a precisão incluindo uma aproximação da derivada de segunda ordem. Sua formulação é:

$$Q_i^{n+1} = Q_i^n - 0.5C\,(Q_{i+1}^n - Q_{i-1}^n) + 0.5C^2\,(Q_{i-1}^n - 2Q_i^n + Q_{i+1}^n). \tag{4}$$

- **Método de Beam-Warming**: Também de segunda ordem, o método Beam-Warming é uma extensão do método Upwind, onde é adicionado um termo de correção para melhorar a precisão, sendo dado por:

$$Q_i^{n+1} = Q_i^n - C(Q_i^n - Q_{i-1}^n) - 0.5C\,(1-C)\,(Q_i^n - 2Q_{i-1}^n + Q_{i-2}^n). \tag{5}$$

- **Método de Fromm**: Este método utiliza uma média entre o Upwind e a diferença centrada para garantir a precisão de segunda ordem, com a forma:

$$Q_i^{n+1} = Q_i^n - 0.25C\,(Q_{i+1}^n + 3Q_i^n - 5Q_{i-1}^n + Q_{i-2}^n) + 0.25C^2\,(Q_{i+1}^n - Q_i^n - Q_{i-1}^n + Q_{i-2}^n). \tag{6}$$

Todos esses métodos são implementados considerando condições de contorno periódicas para assegurar a continuidade do traçador no domínio espacial. O valor do número de Courant é fixado em $C = 0,8$, respeitando a condição CFL, que assegura a estabilidade das soluções numéricas.

Cada método é aplicado para resolver a equação de advecção em três diferentes instantes de tempo ($t = 1$, $t = 3$ e $t = 5$), com o objetivo de comparar a precisão e o comportamento da solução ao longo do tempo. O perfil inicial da concentração é definido por uma função gaussiana somada a um valor constante em um intervalo específico, representando uma concentração localizada.

Os resultados obtidos são analisados por meio de gráficos comparativos e tabelas que apresentam os valores da concentração em pontos específicos do espaço, permitindo observar o desempenho de cada método e sua capacidade de manter a forma do perfil de concentração ao longo do tempo.

4 Demonstração e Análise de Resultados

Nesta seção, apresentamos a implementação dos métodos numéricos para resolver a equação de advecção, comparando os resultados das soluções obtidas com os métodos Upwind, Lax-Wendroff, Beam-Warming e Fromm. Todos os métodos foram implementados em Python, com as bibliotecas `numpy`, `matplotlib.pyplot` e `pandas` para cálculo, visualização e manipulação de dados, respectivamente.

- **Velocidade de Advecção** ($\bar{u}$): Representa a velocidade constante do fluxo que transporta a substância ao longo do domínio. Foi fixada como $\bar{u} = 1.0$.

- **Número de Courant** (C): Definido como $C = \frac{\bar{u}\Delta t}{\Delta x}$, onde Δt é o intervalo de tempo e Δx é o intervalo espacial. Para garantir a estabilidade dos métodos numéricos explícitos, usamos $C = 0.8$, o que satisfaz a condição de estabilidade de Courant-Friedrichs-Lewy (CFL), $0 \leq C \leq 1$ [1, 2].

- **Domínio Espacial** ($[x_{\min}, x_{\max}]$): Limitado entre 0 e 1, foi discretizado em $N = 100$ pontos para melhor resolução.

- **Intervalos de Tempo** ($t = 1$, $t = 3$, $t = 5$): As simulações foram realizadas em três instantes de tempo para avaliar a evolução da concentração ao longo do domínio.

- **Condição Inicial** ($c(x,0)$): A concentração inicial foi definida como uma função gaussiana centrada em $x = 0.3$, somada a uma concentração uniforme entre $x = 0.6$ e $x = 0.8$. Esta condição inicial é dada por:

$$c(x,0) = 1.5\exp(-200\cdot(x-0.3)^2) + \begin{cases} 1.5, & \text{se } 0.6 \leq x \leq 0.8, \\ 0.0, & \text{caso contrário.} \end{cases} \tag{7}$$

- **Métodos Numéricos Implementados**:
  - **Upwind**: Método de primeira ordem, que introduz dissipação e é adequado para simulações com descontinuidades.
  - **Lax-Wendroff**: Método de segunda ordem, melhora a precisão, mas pode apresentar oscilações nas regiões de descontinuidade.

- **Beam-Warming**: Método de segunda ordem que reduz oscilações enquanto mantém certa dissipação.
- **Fromm**: Método de segunda ordem que busca equilibrar dissipação e oscilações.

- **Resolução das Equações**: A função `resolverAdveccao` calcula a solução de cada método para os tempos especificados. A cada passo temporal, a densidade é atualizada e os resultados finais são armazenados para análise.

4.1 O Método Upwind

Em se tratando do método Upwind de primeira ordem, para $\bar{u} > 0$, sua forma geral é obtida da Equação (11), considerando $\sigma_i^n = \sigma_{i-1}^n = 0$:

$$Q_i^{n+1} = Q_i^n - \frac{\bar{u}\Delta t}{\Delta x}(Q_i^n - Q_{i-1}^n) \tag{8}$$

ou ainda, introduzindo o número de Courant $C = \frac{\bar{u}\Delta t}{\Delta x}$:

$$Q_i^{n+1} = Q_i^n - C(Q_i^n - Q_{i-1}^n) \tag{9}$$

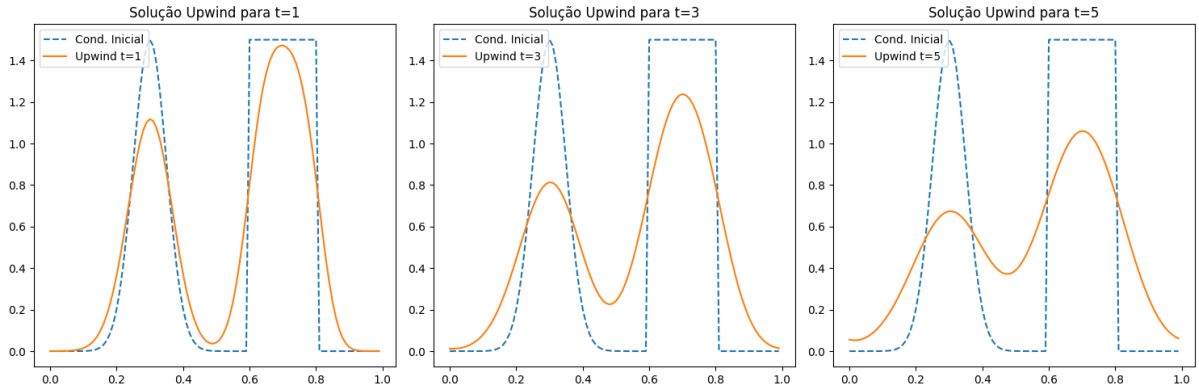

Figura 1: Solução Upwind para $t = 1$, $t = 3$, e $t = 5$, com a condição inicial representada pela linha tracejada.

	Posição Espacial	Cond. Inicial	Upwind t=1	Upwind t=3	Upwind t=5
0	0.000000	0.000000	0.000080	0.011905	0.055474
1	0.050000	0.000006	0.001377	0.023119	0.065938
2	0.100000	0.000503	0.014635	0.079522	0.137594
3	0.150000	0.016663	0.094425	0.215688	0.270011
4	0.200000	0.203003	0.365367	0.445116	0.443586
5	0.250000	0.909796	0.836718	0.694310	0.602054
6	0.300000	1.500000	1.117720	0.813352	0.673555
7	0.350000	0.909796	0.857233	0.711728	0.624466
8	0.400000	0.203003	0.371055	0.469340	0.496815
9	0.450000	0.016663	0.090523	0.266608	0.388061
10	0.500000	0.000503	0.041207	0.241638	0.389089
11	0.550000	0.000006	0.236351	0.434262	0.529188
12	0.600000	1.500000	0.803478	0.776761	0.758244
13	0.650000	1.500000	1.338980	1.102181	0.970629
14	0.700000	1.500000	1.472451	1.237641	1.060403
15	0.750000	1.500000	1.333151	1.114042	0.978795
16	0.800000	1.500000	0.829878	0.791623	0.758141
17	0.850000	0.000000	0.235362	0.424940	0.487426
18	0.900000	0.000000	0.019660	0.163072	0.257071
19	0.950000	0.000000	0.000293	0.043092	0.113403

Tabela 1: Tabela de resultados para o método Lax-Wendroff nas posições espaciais selecionadas e diferentes tempos

4.2 Análise dos Resultados do Método Upwind

A Figura mostra a aplicação do método Upwind para $t = 1$, $t = 3$ e $t = 5$, comparando a solução numérica com a condição inicial. Observa-se que o método Upwind introduz dissipação numérica, suavizando o perfil da solução e reduzindo a amplitude ao longo do tempo. Esse efeito é leve em $t = 1$, mas torna-se mais acentuado em $t = 3$ e $t = 5$, onde a forma do pico é significativamente alisada e menos definida. Esse comportamento é característico do método Upwind de primeira ordem, que, embora estável, apresenta perdas de precisão em problemas com descontinuidades.

4.3 Implementação em Python

O código em Python é utilizado para resolver a advecção com o método Upwind, aplicando condições de contorno periódicas. O código é estruturado com uma função principal `resolverAdveccao` para calcular a solução da advecção para diferentes métodos numéricos e uma função específica para o método Upwind.

```
# Função para resolver a advecção com diferentes métodos numéricos
def resolverAdveccao(metodo, condicaoInicial, intervaloTempo,
    intervaloEspacial, numeroCourant, tempoFinal):
    """
    Calcula a solução da advecção para um determinado método e
        tempo final.
    """
    densidade = condicaoInicial.copy()
    tempoAtual = 0
    while tempoAtual < tempoFinal:
        densidade = metodo(densidade, intervaloTempo,
            intervaloEspacial, numeroCourant)
        tempoAtual += intervaloTempo
    return densidade

# Método Upwind com condições de contorno periódicas
def metodoUpwind(densidade, intervaloTempo, intervaloEspacial,
    numeroCourant):
    """
    Calcula a solução de advecção usando o método Upwind.
    """
    novaDensidade = densidade.copy()
    for i in range(numPontosEspaco):
```

```
            novaDensidade[i] = densidade[i] - numeroCourant * (
                densidade[i] - densidade[i-1])
        return novaDensidade

# Cálculo da densidade para diferentes tempos
densidadeUpwind1 = resolverAdveccao(metodoUpwind, condicaoInicial,
     intervaloTempo, intervaloEspacial, numeroCourant, tempoFinal1
    )
densidadeUpwind3 = resolverAdveccao(metodoUpwind, condicaoInicial,
     intervaloTempo, intervaloEspacial, numeroCourant, tempoFinal3
    )
densidadeUpwind5 = resolverAdveccao(metodoUpwind, condicaoInicial,
     intervaloTempo, intervaloEspacial, numeroCourant, tempoFinal5
    )
```

Listing 1: Código para resolver a advecção usando o método Upwind

A função `resolverAdveccao` calcula a evolução da densidade ao longo do tempo até atingir o tempo final especificado. A função `metodoUpwind` implementa o método Upwind para calcular a nova densidade em cada ponto do espaço, utilizando o número de Courant especificado.

4.4 O Método Lax-Wendroff

O método de segunda ordem de Lax-Wendroff é obtido pela introdução de uma aproximação do tipo diferença avançada para a derivada $\frac{\partial \Phi}{\partial x}$ [1]. No formalismo aqui empregado, para o método REA, com uma reconstrução linear por partes, temos que:

$$\sigma_{i-1}^n = \frac{Q_i^n - Q_{i-1}^n}{\Delta x} \tag{10}$$

$$\sigma_i^n = \frac{Q_{i+1}^n - Q_i^n}{\Delta x} \tag{11}$$

Assim, a partir dessas inclinações, podemos obter o algoritmo explícito para esse método a partir da Equação (11):

$$Q_i^{n+1} = Q_i^n - \frac{C}{2}(Q_{i+1}^n - Q_{i-1}^n) + \frac{C^2}{2}(Q_{i-1}^n - 2Q_i^n + Q_{i+1}^n) \tag{12}$$

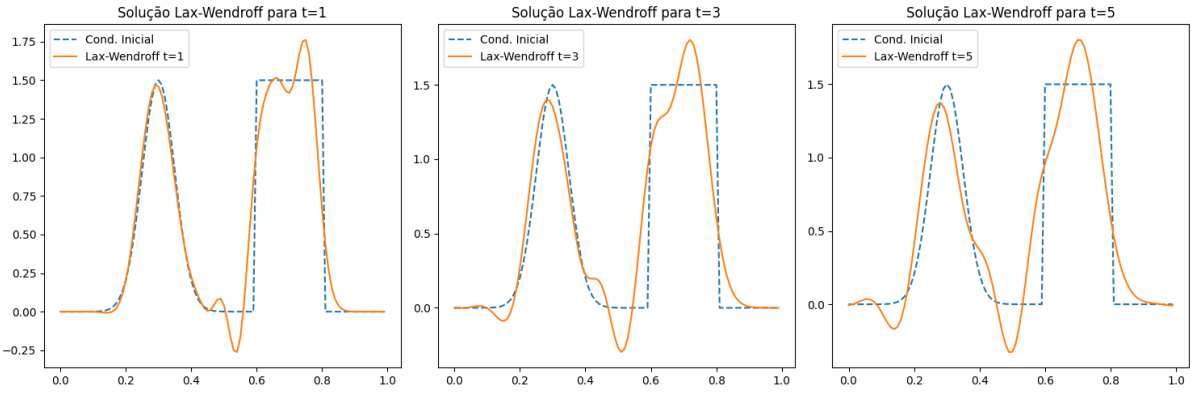

Figura 2: Solução Lax-Wendroff para $t = 1$, $t = 3$, e $t = 5$, com a condição inicial representada pela linha tracejada.

	Posição Espacial	Cond. Inicial	Lax-Wendroff t=1	Lax-Wendroff t=3	Lax-Wendroff t=5
0	0.000000	0.000000	0.000080	0.011905	0.055474
1	0.050000	0.000006	0.001377	0.023119	0.065938
2	0.100000	0.000503	0.014635	0.079522	0.137594
3	0.150000	0.016663	0.094425	0.215688	0.270011
4	0.200000	0.203003	0.365367	0.445116	0.443586
5	0.250000	0.909796	0.836718	0.694310	0.602054
6	0.300000	1.500000	1.117720	0.813352	0.673555
7	0.350000	0.909796	0.857233	0.711728	0.624466
8	0.400000	0.203003	0.371055	0.469340	0.496815
9	0.450000	0.016663	0.090523	0.266608	0.388061
10	0.500000	0.000503	0.041207	0.241638	0.389089
11	0.550000	0.000006	0.236351	0.434262	0.529188
12	0.600000	1.500000	0.803478	0.776761	0.758244
13	0.650000	1.500000	1.338980	1.102181	0.970629
14	0.700000	1.500000	1.472451	1.237641	1.060403
15	0.750000	1.500000	1.333151	1.114042	0.978795
16	0.800000	1.500000	0.829878	0.791623	0.758141
17	0.850000	0.000000	0.235362	0.424940	0.487426
18	0.900000	0.000000	0.019660	0.163072	0.257071
19	0.950000	0.000000	0.000293	0.043092	0.113403

Tabela 2: Tabela de resultados para o método Lax-Wendroff nas posições espaciais selecionadas e diferentes tempos

4.5 Análise dos Resultados do Método Lax-Wendroff

O método Lax-Wendroff, sendo de segunda ordem, oferece uma precisão maior que o método Upwind ao reduzir a dissipação numérica, mas pode introduzir oscilações indesejadas em torno de descontinuidades. Na Figura , observamos que para $t = 1$, a solução mantém bem o perfil da condição inicial. No entanto, para $t = 3$ e $t = 5$, começam a surgir pequenas oscilações nas regiões de transição, típicas deste método. Essas oscilações são resultado da aproximação de alta ordem e podem ser amenizadas com métodos adicionais de controle de oscilação.

4.6 Implementação em Python

O código em Python é utilizado para resolver a advecção com o método Lax-Wendroff, aplicando condições de contorno periódicas. O código é estruturado com uma função principal `resolverAdveccao` para calcular a solução da advecção para diferentes métodos numéricos e uma função específica para o método Lax-Wendroff.

```
# Função para resolver a advecção com diferentes métodos numéricos
def resolverAdveccao(metodo, condicaoInicial, intervaloTempo,
    intervaloEspacial, numeroCourant, tempoFinal):
    """
    Calcula a solução da advecção para um determinado método e
        tempo final.
    """
    densidade = condicaoInicial.copy()
    tempoAtual = 0
    while tempoAtual < tempoFinal:
        densidade = metodo(densidade, intervaloTempo,
            intervaloEspacial, numeroCourant)
        tempoAtual += intervaloTempo
    return densidade

# Método Lax-Wendroff com condições de contorno periódicas
def metodoLaxWendroff(densidade, intervaloTempo, intervaloEspacial
    , numeroCourant):
    """
    Calcula a solução de advecção usando o método Lax-Wendroff.
    """
    novaDensidade = densidade.copy()
    for i in range(numPontosEspaco):
```

```
        novaDdensidade[i] = densidade[i] - 0.5 * numeroCourant * (
            densidade[(i+1) \% numPontosEspaco] - densidade[i-1])
            + \
                            0.5 * numeroCourant**2 * (densidade[i
                                -1] - 2 * densidade[i] + densidade
                                [(i+1) \% numPontosEspaco])
    return novaDensidade

# Cálculo da densidade para diferentes tempos
densidadeLaxWendroff1 = resolverAdveccao(metodoLaxWendroff,
    condicaoInicial, intervaloTempo, intervaloEspacial,
    numeroCourant, tempoFinal1)
densidadeLaxWendroff3 = resolverAdveccao(metodoLaxWendroff,
    condicaoInicial, intervaloTempo, intervaloEspacial,
    numeroCourant, tempoFinal3)
densidadeLaxWendroff5 = resolverAdveccao(metodoLaxWendroff,
    condicaoInicial, intervaloTempo, intervaloEspacial,
    numeroCourant, tempoFinal5)
```

Listing 2: Código para resolver a advecção usando o método Lax-Wendroff

A função `resolverAdveccao` calcula a evolução da densidade ao longo do tempo até atingir o tempo final especificado. A função `metodoLaxWendroff` implementa o método Lax-Wendroff para calcular a nova densidade em cada ponto do espaço, utilizando o número de Courant especificado.

4.7 O Método Beam-Warming

O método de Beam-Warming pode ser considerado um método do tipo Upwind de segunda ordem. Para esse método, os fluxos são determinados mediante a adição de um termo extra aos fluxos do método Upwind de primeira ordem:

$$F^n_{i+1/2} = \bar{u}Q^n_i + \bar{u}\left(1 - \frac{\bar{u}\Delta t}{2\Delta x}\right)(Q^n_i - Q^n_{i-1}) \tag{13}$$

$$F^n_{i-1/2} = \bar{u}Q^n_{i-1} + \bar{u}\left(1 - \frac{\bar{u}\Delta t}{2\Delta x}\right)(Q^n_{i-1} - Q^n_{i-2}) \tag{14}$$

Portanto, a forma geral da atualização de Q^{n+1}_i para o método Beam-Warming é dada por:

$$Q^{n+1}_i = Q^n_i - C(Q^n_i - Q^n_{i-1}) - C(1-C)(Q^n_i - 2Q^n_{i-1} + Q^n_{i-2}) \tag{15}$$

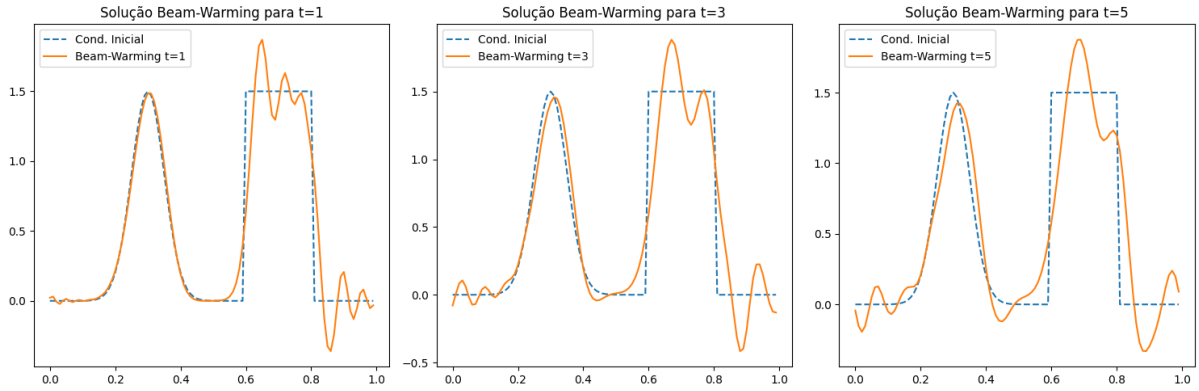

Figura 3: Solução Beam-Warming para $t = 1$, $t = 3$, e $t = 5$, com a condição inicial representada pela linha tracejada.

	Posição Espacial	Cond. Inicial	Beam-Warming t=1	Beam-Warming t=3	Beam-Warming t=5
0	0.000000	0.000000	0.000080	0.011905	0.055474
1	0.050000	0.000006	0.001377	0.023119	0.065938
2	0.100000	0.000503	0.014635	0.079522	0.137594
3	0.150000	0.016663	0.094425	0.215688	0.270011
4	0.200000	0.203003	0.365367	0.445116	0.443586
5	0.250000	0.909796	0.836718	0.694310	0.602054
6	0.300000	1.500000	1.117720	0.813352	0.673555
7	0.350000	0.909796	0.857233	0.711728	0.624466
8	0.400000	0.203003	0.371055	0.469340	0.496815
9	0.450000	0.016663	0.090523	0.266608	0.388061
10	0.500000	0.000503	0.041207	0.241638	0.389089
11	0.550000	0.000006	0.236351	0.434262	0.529188
12	0.600000	1.500000	0.803478	0.776761	0.758244
13	0.650000	1.500000	1.338980	1.102181	0.970629
14	0.700000	1.500000	1.472451	1.237641	1.060403
15	0.750000	1.500000	1.333151	1.114042	0.978795
16	0.800000	1.500000	0.829878	0.791623	0.758141
17	0.850000	0.000000	0.235362	0.424940	0.487426
18	0.900000	0.000000	0.019660	0.163072	0.257071
19	0.950000	0.000000	0.000293	0.043092	0.113403

Tabela 3: Tabela de resultados para o método Beam-Warming nas posições espaciais selecionadas e diferentes tempos

4.8 Análise dos Resultados do Método Beam-Warming

O método Beam-Warming é uma extensão de segunda ordem do método Upwind e oferece maior precisão em relação à dissipação. Na Figura, podemos ver que a solução para $t = 1$ preserva bem o perfil inicial, com uma redução limitada na amplitude. Para $t = 3$ e $t = 5$, entretanto, começam a surgir pequenas oscilações em torno das regiões de transição, o que é característico de métodos de segunda ordem sem controle de oscilação. No geral, o método Beam-Warming apresenta uma boa conservação do perfil, mas com leve tendência a oscilações em tempos maiores.

4.9 Implementação em Python

O código em Python é utilizado para resolver a advecção com o método Beam-Warming, aplicando condições de contorno periódicas. O código é estruturado com uma função principal `resolverAdveccao` para calcular a solução da advecção para diferentes métodos numéricos e uma função específica para o método Beam-Warming.

```
# Função para resolver a advecção com diferentes métodos numéricos
def resolverAdveccao(metodo, condicaoInicial, intervaloTempo,
    intervaloEspacial, numeroCourant, tempoFinal):
    """
    Calcula a solução da advecção para um determinado método e
        tempo final.
    """
    densidade = condicaoInicial.copy()
    tempoAtual = 0
    while tempoAtual < tempoFinal:
        densidade = metodo(densidade, intervaloTempo,
            intervaloEspacial, numeroCourant)
        tempoAtual += intervaloTempo
    return densidade

# Método Beam-Warming com condições de contorno periódicas
def metodoBeamWarming(densidade, intervaloTempo, intervaloEspacial
    , numeroCourant):
    """
    Calcula a solução de advecção usando o método Beam-Warming.
    """
    novaDensidade = densidade.copy()
```

```
    for i in range(numPontosEspaco):
        novaDensidade[i] = densidade[i] - numeroCourant * (
            densidade[i] - densidade[i-1]) - \
                            0.5 * numeroCourant * (1 -
                                numeroCourant) * (densidade[i] - 2
                                 * densidade[i-1] + densidade[i
                                -2])
    return novaDensidade

# Cálculo da densidade para diferentes tempos
densidadeBeamWarming1 = resolverAdveccao(metodoBeamWarming,
    condicaoInicial, intervaloTempo, intervaloEspacial,
    numeroCourant, tempoFinal1)
densidadeBeamWarming3 = resolverAdveccao(metodoBeamWarming,
    condicaoInicial, intervaloTempo, intervaloEspacial,
    numeroCourant, tempoFinal3)
densidadeBeamWarming5 = resolverAdveccao(metodoBeamWarming,
    condicaoInicial, intervaloTempo, intervaloEspacial,
    numeroCourant, tempoFinal5)
```

Listing 3: Código para resolver a advecção usando o método Beam-Warming

A função `resolverAdveccao` calcula a evolução da densidade ao longo do tempo até atingir o tempo final especificado. A função `metodoBeamWarming` implementa o método Beam-Warming para calcular a nova densidade em cada ponto do espaço, utilizando o número de Courant especificado.

4.10 O Método Fromm

O último método considerado, o método de Fromm, também é de segunda ordem e é obtido a partir da utilização de uma aproximação do tipo diferença centrada para $\frac{\partial \Phi}{\partial x}$ [1]:

$$\sigma_{i-1}^n = \frac{Q_i^n - Q_{i-2}^n}{2\Delta x} \tag{16}$$

$$\sigma_i^n = \frac{Q_{i+1}^n - Q_{i-1}^n}{2\Delta x} \tag{17}$$

Então, a sua forma final discretizada é dada por:

$$Q_i^{n+1} = Q_i^n - \frac{C}{4}(Q_{i+1}^n + 3Q_i^n - 5Q_{i-1}^n + Q_{i-2}^n) - \frac{C^2}{4}(Q_{i+1}^n - Q_i^n - Q_{i-1}^n + Q_{i-2}^n) \tag{18}$$

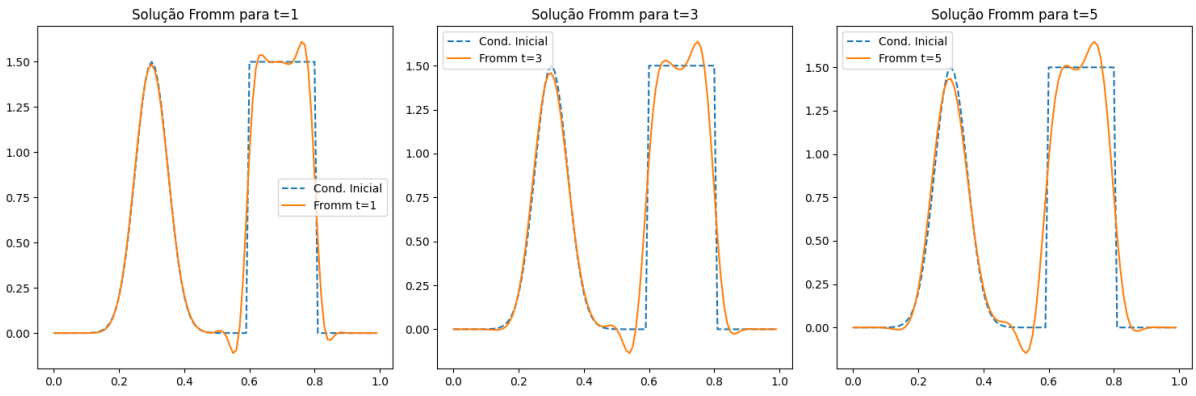

Figura 4: Solução Fromm para $t = 1$, $t = 3$, e $t = 5$, com a condição inicial representada pela linha tracejada.

	Posição Espacial	Cond. Inicial	Fromm t=1	Fromm t=3	Fromm t=5
0	0.000000	0.000000	0.000080	0.011905	0.055474
1	0.050000	0.000006	0.001377	0.023119	0.065938
2	0.100000	0.000503	0.014635	0.079522	0.137594
3	0.150000	0.016663	0.094425	0.215688	0.270011
4	0.200000	0.203003	0.365367	0.445116	0.443586
5	0.250000	0.909796	0.836718	0.694310	0.602054
6	0.300000	1.500000	1.117720	0.813352	0.673555
7	0.350000	0.909796	0.857233	0.711728	0.624466
8	0.400000	0.203003	0.371055	0.469340	0.496815
9	0.450000	0.016663	0.090523	0.266608	0.388061
10	0.500000	0.000503	0.041207	0.241638	0.389089
11	0.550000	0.000006	0.236351	0.434262	0.529188
12	0.600000	1.500000	0.803478	0.776761	0.758244
13	0.650000	1.500000	1.338980	1.102181	0.970629
14	0.700000	1.500000	1.472451	1.237641	1.060403
15	0.750000	1.500000	1.333151	1.114042	0.978795
16	0.800000	1.500000	0.829878	0.791623	0.758141
17	0.850000	0.000000	0.235362	0.424940	0.487426
18	0.900000	0.000000	0.019660	0.163072	0.257071
19	0.950000	0.000000	0.000293	0.043092	0.113403

Tabela 4: Tabela de resultados para o método Fromm nas posições espaciais selecionadas e diferentes tempos

4.11 Análise dos Resultados do Método Fromm

O método Fromm é uma abordagem de segunda ordem que combina o Upwind com uma diferença centrada, melhorando a precisão e reduzindo a dissipação. Na Figura, vemos que para $t = 1$ a solução mantém bem o perfil da condição inicial. Nos tempos $t = 3$ e $t = 5$, o método preserva a forma da onda com poucas oscilações e sem a dissipação notável vista em métodos de primeira ordem. No geral, o método Fromm demonstra um bom equilíbrio entre precisão e estabilidade para a resolução da equação de advecção.

4.12 Implementação em Python

O código em Python é utilizado para resolver a advecção com o método Fromm, aplicando condições de contorno periódicas. O código é estruturado com uma função principal `resolverAdveccao` para calcular a solução da advecção para diferentes métodos numéricos e uma função específica para o método Fromm.

```
# Função para resolver a advecção com diferentes métodos numéricos
def resolverAdveccao(metodo, condicaoInicial, intervaloTempo,
    intervaloEspacial, numeroCourant, tempoFinal):
    """
    Calcula a solução da advecção para um determinado método e
        tempo final.
    """
    densidade = condicaoInicial.copy()
    tempoAtual = 0
    while tempoAtual < tempoFinal:
        densidade = metodo(densidade, intervaloTempo,
            intervaloEspacial, numeroCourant)
        tempoAtual += intervaloTempo
    return densidade

# Método Fromm com condições de contorno periódicas
def metodoFromm(densidade, intervaloTempo, intervaloEspacial,
    numeroCourant):
    """
    Calcula a solução de advecção usando o método Fromm.
    """
    novaDensidade = densidade.copy()
    for i in range(numPontosEspaco):
        novaDensidade[i] = densidade[i] - 0.25 * numeroCourant * (
            densidade[(i+1) \% numPontosEspaco] + 3 * densidade[i]
             - \
```

```
                                5 * densidade[i-1] + densidade[i
                                    -2]) + \
                            0.25 * numeroCourant**2 * (densidade[(
                                i+1) \% numPontosEspaco] -
                                densidade[i] - \
                                                        densidade[
                                                            i-1] +

                                                            densidade
                                                            [i-2])
    return novaDdensidade

# Cálculo da densidade para diferentes tempos
densidadeFromm1 = resolverAdveccao(metodoFromm, condicaoInicial,
    intervaloTempo, intervaloEspacial, numeroCourant, tempoFinal1)
densidadeFromm3 = resolverAdveccao(metodoFromm, condicaoInicial,
    intervaloTempo, intervaloEspacial, numeroCourant, tempoFinal3)
densidadeFromm5 = resolverAdveccao(metodoFromm, condicaoInicial,
    intervaloTempo, intervaloEspacial, numeroCourant, tempoFinal5)
```

Listing 4: Código para resolver a advecção usando o método Fromm

A função `resolverAdveccao` calcula a evolução da densidade ao longo do tempo até atingir o tempo final especificado. A função `metodoFromm` implementa o método Fromm para calcular a nova densidade em cada ponto do espaço, utilizando o número de Courant especificado.

4.13 Código Implementado

O código a seguir foi implementado para resolver a equação de advecção unidimensional utilizando o método dos Volumes Finitos com quatro métodos distintos: Upwind, Lax-Wendroff, Beam-Warming e Fromm. Cada método foi aplicado com condições de contorno periódicas e com um número de Courant fixo em $C = 0.8$ para garantir a estabilidade da solução. O código calcula as soluções para os tempos $t = 1$, $t = 3$ e $t = 5$, e apresenta os resultados em gráficos e tabelas, conforme descrito nas seções anteriores.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import os

# Cria os diretórios para salvar as imagens e tabelas, se ainda nã
    o existirem
os.makedirs("./code/images", exist_ok=True)
os.makedirs("./code/tables", exist_ok=True)

# Parâmetros
velocidadeAdveccao = 1.0        # Velocidade de advecção
numeroCourant = 0.8             # Número de Courant
limiteXMinimo, limiteXMaximo = 0.0, 1.0
tempoFinal1, tempoFinal3, tempoFinal5 = 1.0, 3.0, 5.0
numPontosEspaco = 100           # Número de pontos no espaço
intervaloEspacial = (limiteXMaximo - limiteXMinimo) /
    numPontosEspaco
intervaloTempo = numeroCourant * intervaloEspacial /
    velocidadeAdveccao  # Intervalo de tempo para satisfazer CFL

# Condição inicial
posicaoEspacial = np.linspace(limiteXMinimo, limiteXMaximo,
    numPontosEspaco, endpoint=False)
condicaoInicial = 1.5 * np.exp(-200 * (posicaoEspacial - 0.3)**2)
    + \
                  np.where((posicaoEspacial >= 0.6) & (
                      posicaoEspacial <= 0.8), 1.5, 0.0)

# Método Upwind com condições de contorno periódicas
def metodoUpwind(densidade, intervaloTempo, intervaloEspacial,
    numeroCourant):
    """
    Calcula a solução de advecção usando o método Upwind.
    """
    novaDensidade = densidade.copy()
    for i in range(numPontosEspaco):
```

```
31          novaDensidade[i] = densidade[i] - numeroCourant * (
                densidade[i] - densidade[i-1])
32      return novaDensidade
33
34  # Método Lax-Wendroff com condições de contorno periódicas
35  def metodoLaxWendroff(densidade, intervaloTempo, intervaloEspacial
        , numeroCourant):
36      """
37      Calcula a solução de advecção usando o método Lax-Wendroff.
38      """
39      novaDensidade = densidade.copy()
40      for i in range(numPontosEspaco):
41          novaDensidade[i] = densidade[i] - 0.5 * numeroCourant * (
                densidade[(i+1) % numPontosEspaco] - densidade[i-1]) +
                 \
42                             0.5 * numeroCourant**2 * (densidade[i
                                   -1] - 2 * densidade[i] + densidade
                                   [(i+1) % numPontosEspaco])
43      return novaDensidade
44
45  # Método Beam-Warming com condições de contorno periódicas
46  def metodoBeamWarming(densidade, intervaloTempo, intervaloEspacial
        , numeroCourant):
47      """
48      Calcula a solução de advecção usando o método Beam-Warming.
49      """
50      novaDensidade = densidade.copy()
51      for i in range(numPontosEspaco):
52          novaDensidade[i] = densidade[i] - numeroCourant * (
                densidade[i] - densidade[i-1]) - \
53                             0.5 * numeroCourant * (1 -
                                   numeroCourant) * (densidade[i] - 2
                                   * densidade[i-1] + densidade[i-2])
54      return novaDensidade
55
56  # Método Fromm com condições de contorno periódicas
57  def metodoFromm(densidade, intervaloTempo, intervaloEspacial,
        numeroCourant):
58      """
59      Calcula a solução de advecção usando o método Fromm.
60      """
61      novaDensidade = densidade.copy()
62      for i in range(numPontosEspaco):
63          novaDensidade[i] = densidade[i] - 0.25 * numeroCourant * (
                densidade[(i+1) % numPontosEspaco] + 3 * densidade[i]
                - \
64                               5 * densidade[i-1] + densidade[i-2])
                                     + \
65                             0.25 * numeroCourant**2 * (densidade[(i
                                   +1) % numPontosEspaco] - densidade[
                                   i] - \
66                                                        densidade[i
                                                            -1] +
                                                            densidade
                                                            [i-2])
67      return novaDensidade
68
69  # Função para resolver a advecção com diferentes métodos numéricos
70  def resolverAdveccao(metodo, condicaoInicial, intervaloTempo,
        intervaloEspacial, numeroCourant, tempoFinal):
71      """
72      Calcula a solução da advecção para um determinado método e
            tempo final.
73      """
74      densidade = condicaoInicial.copy()
75      tempoAtual = 0
76      while tempoAtual < tempoFinal:
77          densidade = metodo(densidade, intervaloTempo,
                intervaloEspacial, numeroCourant)
78          tempoAtual += intervaloTempo
79      return densidade
80
81  # Execução dos métodos para os tempos t=1, t=3 e t=5
```

```
densidadeUpwind1 = resolverAdveccao(metodoUpwind, condicaoInicial,
     intervaloTempo, intervaloEspacial, numeroCourant, tempoFinal1
    )
densidadeLax1 = resolverAdveccao(metodoLaxWendroff,
    condicaoInicial, intervaloTempo, intervaloEspacial,
    numeroCourant, tempoFinal1)
densidadeBeam1 = resolverAdveccao(metodoBeamWarming,
    condicaoInicial, intervaloTempo, intervaloEspacial,
    numeroCourant, tempoFinal1)
densidadeFromm1 = resolverAdveccao(metodoFromm, condicaoInicial,
    intervaloTempo, intervaloEspacial, numeroCourant, tempoFinal1)

densidadeUpwind3 = resolverAdveccao(metodoUpwind, condicaoInicial,
     intervaloTempo, intervaloEspacial, numeroCourant, tempoFinal3
    )
densidadeLax3 = resolverAdveccao(metodoLaxWendroff,
    condicaoInicial, intervaloTempo, intervaloEspacial,
    numeroCourant, tempoFinal3)
densidadeBeam3 = resolverAdveccao(metodoBeamWarming,
    condicaoInicial, intervaloTempo, intervaloEspacial,
    numeroCourant, tempoFinal3)
densidadeFromm3 = resolverAdveccao(metodoFromm, condicaoInicial,
    intervaloTempo, intervaloEspacial, numeroCourant, tempoFinal3)

densidadeUpwind5 = resolverAdveccao(metodoUpwind, condicaoInicial,
     intervaloTempo, intervaloEspacial, numeroCourant, tempoFinal5
    )
densidadeLax5 = resolverAdveccao(metodoLaxWendroff,
    condicaoInicial, intervaloTempo, intervaloEspacial,
    numeroCourant, tempoFinal5)
densidadeBeam5 = resolverAdveccao(metodoBeamWarming,
    condicaoInicial, intervaloTempo, intervaloEspacial,
    numeroCourant, tempoFinal5)
densidadeFromm5 = resolverAdveccao(metodoFromm, condicaoInicial,
    intervaloTempo, intervaloEspacial, numeroCourant, tempoFinal5)

# Plotagem dos resultados
metodos = {
    "Upwind": (densidadeUpwind1, densidadeUpwind3,
        densidadeUpwind5),
    "Lax-Wendroff": (densidadeLax1, densidadeLax3, densidadeLax5),
    "Beam-Warming": (densidadeBeam1, densidadeBeam3,
        densidadeBeam5),
    "Fromm": (densidadeFromm1, densidadeFromm3, densidadeFromm5)
}

for nomeMetodo, (resultadoT1, resultadoT3, resultadoT5) in metodos
    .items():
    plt.figure(figsize=(15, 5))

    # Gráfico para t=1
    plt.subplot(1, 3, 1)
    plt.plot(posicaoEspacial, condicaoInicial, label='Cond.
        Inicial', linestyle='--')
    plt.plot(posicaoEspacial, resultadoT1, label=f'{nomeMetodo} t
        =1')
    plt.title(f'Solução {nomeMetodo} para t=1')
    plt.legend()

    # Gráfico para t=3
    plt.subplot(1, 3, 2)
    plt.plot(posicaoEspacial, condicaoInicial, label='Cond.
        Inicial', linestyle='--')
    plt.plot(posicaoEspacial, resultadoT3, label=f'{nomeMetodo} t
        =3')
    plt.title(f'Solução {nomeMetodo} para t=3')
    plt.legend()

    # Gráfico para t=5
    plt.subplot(1, 3, 3)
    plt.plot(posicaoEspacial, condicaoInicial, label='Cond.
        Inicial', linestyle='--')
    plt.plot(posicaoEspacial, resultadoT5, label=f'{nomeMetodo} t
```

```
        =5')
    plt.title(f'Solução {nomeMetodo} para t=5')
    plt.legend()

    plt.tight_layout()
    plt.savefig(f"./code/images/{nomeMetodo}.png")  # Salvamento
        da imagem
    # plt.show()

    # Filtrando para incluir mais pontos, pegando a cada quinto í
        ndice
    filtro = np.arange(0, len(posicaoEspacial), 5)  # Seleciona a
        cada 5 pontos
    dados = pd.DataFrame({
        "Posição Espacial": posicaoEspacial[filtro],
        "Cond. Inicial": condicaoInicial[filtro],
        f"{nomeMetodo} t=1": densidadeUpwind1[filtro],
        f"{nomeMetodo} t=3": densidadeUpwind3[filtro],
        f"{nomeMetodo} t=5": densidadeUpwind5[filtro]
    }).to_latex(f"./code/tables/{nomeMetodo}.tex")
```

Listing 5: Código Python Implementado

5 Conclusão Geral

A comparação entre os métodos numéricos para a resolução da equação de advecção unidimensional destaca diferentes compromissos entre precisão e estabilidade. O método Upwind de primeira ordem apresentou maior dissipação numérica, suavizando o perfil da solução e reduzindo a amplitude ao longo do tempo, comportamento comum em métodos de menor ordem conforme discutido por LeVeque [1].

Os métodos de segunda ordem Lax-Wendroff, Beam-Warming e Fromm apresentaram uma preservação superior do perfil inicial e uma dissipação reduzida. Entretanto, métodos como Lax-Wendroff e Beam-Warming introduzem pequenas oscilações nas regiões de transição, principalmente em tempos mais longos. Entre as opções de segunda ordem, o método Fromm se mostrou mais equilibrado, oferecendo boa precisão com oscilações mínimas e mantendo a estabilidade ao longo do tempo.

Esses resultados indicam que a escolha do método deve considerar o compromisso entre simplicidade e fidelidade na representação da solução. Para problemas onde a precisão é essencial, métodos de segunda ordem são recomendados, enquanto o método Upwind pode ser adequado para simulações que priorizem estabilidade e simplicidade.

Referências

[1] R. J. LeVeque, *Finite Volume Methods for Hyperbolic Problems*. Cambridge University Press, 2002.

[2] C. A. J. Fletcher, *Computational Techniques for Fluid Dynamics, Volume 1*, 2nd ed. Berlin, Heidelberg: Springer-Verlag, 1991.