Trabalho 2 de Métodos Numéricos para Equações Diferenciais II

Ariel Nogueira Kovaljski

Nova Friburgo, XX de novembro de 2020

Sumário

Lista de Figuras

1. Resumo

A obtenção de solução de equações diferenciais parciais (EDPs) é muitas vezes extremamente difícil ou até mesmo impossível, pois a mesma pode não possuir solução analítica. Através do Método dos Volumes Finitos, é possível discretizá-la, o que permite a obtenção de uma solução numérica aproximada ao se utilizar métodos numéricos computacionais.

Neste trabalho, foram utilizados quatro métodos numéricos — Forward Time-Backward Space (FTBS), Lax-Friedrichs, Lax-Wendroff e Beam-Warming — visando resolver a EDP da advecção. A partir da solução obtida por cada método, foram variados parâmetros como o número de células e o tempo de simulação, o que permitiu observar o comportamento de cada e compará-los, revelando suas vantagens e desvantagens para cada situação.

2. Introdução

Neste trabalho foi implementado um método computacional de maneira a resolver a equação de advecção de forma numérica.

Para melhor entender o desenvolvimento, é necessária introdução dos conceitos-chave utilizados, alguns dos quais já foram apresentados no primeiro trabalho.

2.1 A Equação de Advecção

A equação de advecção é obtida a partir da equação de advecção-difusão, introduzida no primeiro trabalho como exemplo de modelagem do escoamento de um contaminante em um córrego. A parte advectiva desta trata apenas do carregamento da substância devido a velocidade da correnteza. A forma mais geral da equação de advecção é

$$\frac{\partial c}{\partial t} + \frac{\partial}{\partial x}(uc) = 0 \tag{2.1}$$

onde c indica a concentração e u a velocidade. Para este trabalho assume-se um u constante e maior que zero, denotado como $\bar{u}$. Sendo assim, a forma final equação da advecção a ser utilizada neste trabalho é

$$\frac{\partial c}{\partial t} + \bar{u}\frac{\partial c}{\partial x} = 0 \tag{2.2}$$

2.2 Método dos Volumes Finitos

O método dos volumes finitos tem como finalidade a discretização do domínio espacial. Este é subdividido em um conjunto de volumes finitos e as variáveis dependentes são determinadas como médias volumétricas sobre estes volumes, avaliadas nos centros dos mesmos. Partindo de um problema unidimensional, temos um caso particular da lei de conservação discretizada,

$$Q_i^{n+1} = Q_i^n - \frac{\Delta t}{\Delta x}\left(F_{i+1/2}^n - F_{i-1/2}^n\right) \tag{2.3}$$

onde F indica o fluxo, definido como,

$$F_{i\pm1/2}^n \approx \frac{1}{\Delta t}\int_{t_n}^{t_{n+1}} f(x_{i\pm1/2}, t)dt$$

e Q indica as concentrações na malha, definido como

$$Q_i^n \approx \frac{1}{\Delta x} \int_{x_{i-1/2}}^{i_{i+1/2}} \phi(x, t_n) dx$$

Considerando que, para problemas hiperbólicos, a velocidade de propagação é finita, é possível definir uma representação para os fluxos nas faces do volume de controle em função de Q^n, isto é,

$$F_{i-1/2}^n = \mathcal{F}(Q_{i-1}^n, Q_i^n)$$

$$F_{i+1/2}^n = \mathcal{F}(Q_i^n, Q_{i+1}^n)$$

Reescrevendo a Eq. 2.3 em função de $\mathcal{F}$ obtém-se,

$$Q_i^{n+1} = Q_i^n - \frac{\Delta t}{\Delta x} \left[\mathcal{F}(Q_i^n, Q_{i+1}^n) - \mathcal{F}(Q_{i-1}^n, Q_i^n) \right] \tag{2.4}$$

de forma que o valor futuro Q_i^{n+1} depende explicitamente dos valores anteriores Q^n na vizinhança do volume i e das funções $\mathcal{F}$.

Neste trabalho, serão utilizados quatro métodos numéricos baseados no método dos volumes finitos — Forward Time-Backward Space (FTBS), Lax-Friedrichs, Lax-Wendroff e Beam-Warming — visando resolver a equação da advecção.

3. Metodologia

Neste capítulo serão abordados os passos e métodos utilizados para se obter a solução numérica do problema proposto.

3.1 Condições Inicial e de Contorno

A resolução de qualquer equação diferencial parcial (EDP) requer a determinação de sua condição(ões) inicial(ais) e de contorno. Como proposto pelo trabalho, a concentração inicial da malha é dada pela seguinte equação

$$c(x,0) = e^{-A(x-B)} + s(x) \tag{3.1}$$

e as condições de contorno são dadas pelas seguintes equações

$$\left(\frac{\partial c}{\partial x}\right)^t_{x=0} = 0 \tag{3.2}$$

$$\left(\frac{\partial c}{\partial x}\right)^t_{x=L_x} = 0 \tag{3.3}$$

para o contorno esquerdo e direito, respectivamente.

Usando o conceitos de volumes fantasmas, é possível determinar o valor dos volumes no contorno, através das seguintes aproximações, para o contorno esquerdo,

$$\left(\frac{\partial c}{\partial c}\right)^t_{x=0} \approx \frac{Q^n_1 - Q^n_0}{\Delta x} = 0$$

$$\therefore\ Q^n_1 = Q^n_0 \tag{3.4}$$

e para o contorno direito,

$$\left(\frac{\partial c}{\partial c}\right)^t_{x=L_x} \approx \frac{Q^n_{nx+1} - Q^n_{nx}}{\Delta x} = 0$$

$$\therefore\ Q^n_{nx+1} = Q^n_{nx} \tag{3.5}$$

3.2 Métodos Numéricos

Os métodos numéricos utilizados para a resolução da equação de advecção são diversos. Nesta seção serão tratados os quatro métodos utilizados neste trabalho.

3.2.1 Métodos *Upwind*

Problemas hiperbólicos, como a equação da advecção, possuem informação (ondas) que se propagam com uma velocidade e sentido característico. A utilização de métodos *upwind* leva em conta essa característica, permitindo uma modelagem mais acurada do fenômeno tratado. O método *upwind* escolhido para a resolução da equação da advecção é o *forward time-backward space* (FTBS).

Forward Time-Backward Space (FTBS)

O FTBS trata da ideia de que, para a equação da advecção unidimensional, há apenas uma única onda que se propaga. O método *upwind* determina o valor de Q_i^{n+1}, onde, para um $\bar{u} > 0$, resulta em um fluxo da esquerda para a direita, de forma que a concentração de cada volume Q_i^{n+1} depende dos volumes atual Q_i^n e anterior Q_{i-1}^n. Sua discretização é:

$$Q_i^{n+1} = Q_i^n - \frac{\bar{u}\Delta t}{\Delta x}\left(Q_i^n - Q_{i-1}^n\right) \tag{3.6}$$

aplicando as condições de contorno, obtém-se,

$$Q_i^{n+1} = Q_i^n - \frac{\bar{u}\Delta t}{\Delta x}\left(Q_1^n - Q_1^n\right)$$

$$\therefore\ Q_1^{n+1} = Q_1^n \tag{3.7}$$

para o primeiro elemento da malha (contorno esquerdo). O último elemento da malha (contorno direito) no método FTBS não necessita do volume fantasma, pois depende apenas do elemento atual Q_i^n e do anterior Q_{i-1}^n, portanto, a Eq. 3.6 aplica-se ao contorno direito.

3.2.2 Lax-Friedrichs (L-F)

Para Lax-Friedrichs (L-F), tem-se a seguinte discretização

$$Q_i^{n+1} = \frac{Q_{i+1}^n + Q_{i-1}^n}{2} - \frac{\bar{u}\Delta t}{2\Delta x}\left(Q_{i+1}^n - Q_{i-1}^n\right) \tag{3.8}$$

aplicando as condições de contorno, obtém-se,

$$Q_1^{n+1} = \frac{Q_2^n + Q_1^n}{2} - \frac{\bar{u}\Delta t}{2\Delta x}\left(Q_2^n - Q_1^n\right) \tag{3.9}$$

para o primeiro elemento da malha (contorno esquerdo). Enquanto para o último elemento da malha (contorno direito), obtém-se,

$$Q_{nx}^{n+1} = \frac{Q_{nx}^n + Q_{nx-1}^n}{2} - \frac{\bar{u}\Delta t}{2\Delta x}\left(Q_{nx}^n - Q_{nx-1}^n\right) \tag{3.10}$$

3.2.3 Métodos de Alta Resolução

A utilização de métodos *upwind* de primeira ordem, apesar de simples, acaba por introduzir significativa difusão numérica, impactando negativamente a acurácia da solução. Os métodos de alta resolução introduzem um termo corretivo, de maneira a minimizar a influência da difusão numérica sobre o resultado final. Os métodos de alta resolução escolhidos para a resolução da equação da advecção são o Lax-Wendroff (L-W) e Beam-Warming (B-W).

Lax-Wendroff (L-W)

Para Lax-Wendroff (L-W), tem-se a seguinte discretização

$$Q_i^{n+1} = Q_i^n - \frac{\bar{u}\Delta t}{2\Delta x}\left(Q_{i+1}^n - Q_{i-1}^n\right) + \frac{\bar{u}^2\Delta t^2}{2\Delta x^2}\left(Q_{i+1}^n - 2Q_i^n + Q_{i-1}^n\right) \tag{3.11}$$

aplicando as condições de contorno, obtém-se,

$$Q_1^{n+1} = Q_1^n - \frac{\bar{u}\Delta t}{2\Delta x}\left(Q_2^n - Q_1^n\right) + \frac{\bar{u}^2\Delta t^2}{2\Delta x^2}\left(Q_2^n - 2Q_1^n + Q_1^n\right)$$

$$Q_1^{n+1} = Q_1^n - \frac{\bar{u}\Delta t}{2\Delta x}\left(Q_2^n - Q_1^n\right) + \frac{\bar{u}^2\Delta t^2}{2\Delta x^2}\left(Q_2^n - Q_1^n\right) \tag{3.12}$$

para o primeiro elemento da malha (contorno esquerdo). Enquanto para o último elemento da malha (contorno direito), obtém-se,

$$Q_{nx}^{n+1} = Q_{nx}^n - \frac{\bar{u}\Delta t}{2\Delta x}\left(Q_{nx}^n - Q_{nx-1}^n\right) + \frac{\bar{u}^2\Delta t^2}{2\Delta x^2}\left(Q_{nx}^n - 2Q_{nx}^n + Q_{nx-1}^n\right)$$

$$Q_{nx}^{n+1} = Q_{nx}^n - \frac{\bar{u}\Delta t}{2\Delta x}\left(Q_{nx}^n - Q_{nx-1}^n\right) + \frac{\bar{u}^2\Delta t^2}{2\Delta x^2}\left(Q_{nx-1}^n - Q_{nx}^n\right) \tag{3.13}$$

Beam-Warming (B-W)

Para Beam-Warming (B-W), tem-se a seguinte discretização

$$Q_i^{n+1} = Q_i^n - \tfrac{\bar{u}\Delta t}{2\Delta x}\left(3Q_i^n - 4Q_{i-1}^n + Q_{i-2}^n\right) + \tfrac{\bar{u}^2\Delta t^2}{2\Delta x^2}\left(Q_i^n - 2Q_{i-1}^n + Q_{i-2}^n\right) \tag{3.14}$$

onde, para o primeiro elemento (contorno esquerdo), devido ao termo Q_{i-2}^n, não é possível aplicar as condições de contorno diretamente. Sendo assim, apenas para o primeiro elemento da malha, será utilizado o método L-W. Desta forma, para $i = 1$ a discretização utilizada é a Eq. 3.12. Para o segundo elemento, aplica-se a condição de contorno esquerda, resultando na seguinte discretização,

$$Q_2^{n+1} = Q_2^n - \tfrac{\bar{u}\Delta t}{2\Delta x}\left(3Q_2^n - 4Q_1^n + Q_1^n\right) + \tfrac{\bar{u}^2\Delta t^2}{2\Delta x^2}\left(Q_2^n - 2Q_1^n + Q_1^n\right)$$

$$Q_2^{n+1} = Q_2^n - \frac{3\bar{u}\Delta t}{2\Delta x}\left(Q_2^n - Q_1^n\right) + \frac{\bar{u}^2\Delta t^2}{2\Delta x^2}\left(Q_2^n - Q_1^n\right) \tag{3.15}$$

Assim como no método FTBS, o último elemento da malha (contorno direito) do método B-W não necessita do volume fantasma, pois depende apenas do elemento atual Q_i^n e dos anteriores Q_{i-1}^n e Q_{i-2}^n, portanto, a Eq. 3.14 aplica-se ao contorno direito.

3.3 Estabilidade

Se trata do comportamento do algoritmo e seus valores numéricos frente aos parâmetros de entrada. Um algoritmo estável se comporta de maneira esperada frente a uma faixa específica de valores de entrada.

A partir da condição de estabilidade para a equação da advecção-difusão, utilizada no primeiro trabalho,

$$\Delta t \leq C \left(\frac{1}{\frac{2\alpha}{\Delta x^2} + \frac{\bar{u}}{\Delta x}} \right) \tag{3.16}$$

considerando que não há componente difusivo α na equação de advecção, o mesmo pode ser igualado a 0.

$$\Delta t \leq C \left(\frac{1}{\frac{2(0)}{\Delta x^2} + \frac{\bar{u}}{\Delta x}} \right)$$

$$\Delta t \leq C \left(\frac{1}{\frac{\bar{u}}{\Delta x}} \right)$$

$$\therefore \ \Delta t \leq C \frac{\Delta x}{\bar{u}} \tag{3.17}$$

obtém-se assim a condição de estabilidade para os métodos numéricos para a equação da advecção, onde C trata-se do número de Courant: adota-se um valor de 0,9 para estre trabalho, visando satisfazer a condição de estabilidade e evitar possíveis erros numéricos durante o cálculo computacional. O método obtido trata-se de um método *condicionalmente estável*, ou seja que depende de uma faixa de valores para garantir a estabilidade de seu funcionamento.

3.4 Programação

Aliado destes conceitos, foi possível construir um programa em linguagem C que calcula as concentrações para cada célula ao longo do tempo, para cada método aqui citado, exportando arquivos de texto com os resultados; estes arquivos, então, são lidos por um *script* Python que gera os gráficos correspondentes.

O programa principal possui a seguinte estrutura, descrita em C:

```
// Vetores para concentração no tempo 'n' e no tempo 'n+1', respectivamente
double Q_old[];
double Q_new[];

// Para cada método, os vetores são inicializados e as concentrações são
// calculadas

// Método FTBS
// Inicializa ambos os vetores com a função de concentração inicial
initializeArray(Q_old, nx, A, B, C, D, E)
initializeArray(Q_new, nx, A, B, C, D, E)

// Calcula Q através do método FTBS
calculateQ_FTBS(Q_old, Q_new);

// Imprime na tela e salva os resultados no arquivo de texto
printAndSaveResults(Q_new, nx, FTBS);

// Método L-F
initializeArray(Q_old, nx, A, B, C, D, E)
initializeArray(Q_new, nx, A, B, C, D, E)
calculateQ_LF(Q_old, Q_new);
```

```
printAndSaveResults(Q_new, nx, LF);

// Método L-W
initializeArray(Q_old, nx, A, B, C, D, E)
initializeArray(Q_new, nx, A, B, C, D, E)
calculateQ_LW(Q_old, Q_new);
printAndSaveResults(Q_new, nx, LW);

// Método B-W
initializeArray(Q_old, nx, A, B, C, D, E)
initializeArray(Q_new, nx, A, B, C, D, E)
calculateQ_BW(Q_old, Q_new);
printAndSaveResults(Q_new, nx, BW);
```

A função `initializeArray` possui a seguinte estrutura:

```
int i;
double x, s;

// Laço 'for' percorre todos os índices do vetor
for (i = 0; i < nx; ++i) {

    // Posição 'x' (m) é definida como índice do volume * largura do volume
    x = i * Delta_x;

    // 's' recebe o valor de 'E' somente se C <= x <= D
    s = (x >= C && x <= D ? E : 0);

    // Cada índice do vetor 'Q' é inicializado segundo a condição inicial
    Q[i] = exp( -A * ((x - b)*(x - b)) ) + s;
}
```

Cada função `calculateQ` possui a seguinte estrutura:

```
// Cálculo de Q, iterado ao longo do tempo
do {

    // Cálculo do volume da fronteira esquerda para o método Beam-Warming, que
    // utiliza Lax-Wendroff apenas para este caso
    if (method == BW) {leftBoudary(LW, 0);}

    // Cálculo do volume da fronteira esquerda, onde 'leftBoundary' é a função
    // da fronteira esquerda específica a cada método. Caso o método seja
    // 'Beam-Warming', calcula-se o volume logo após a fronteira.
    Q_new[0] = method == BW ? leftBoundary(method, 1) : leftBoudary(method, 0);

    // Cálculo de volumes do centro da malha, onde 'center' é a função do centro
    // da malha específica a cada método. Este laço começa em 'i = 2' caso o
    // método seja B-W.
    for (i = (method == BW ? 2 : 1); i < nx - 1; ++i) {
        Q_new[i] = center(method, i);
    }

    // Cálculo do volume da fronteira direita, onde 'rightBoundary' é a função
    // da fronteira direita específica a cada método
    Q_new[i] = rightBoundary(method, i);

    // Vetor 'Q_old' é atualizado com valores do 'Q_new' para o próximo passo
    // de tempo
    for (i = 0; i < nx; ++i) {
        Q_old[i] = Q_new[i];
    }
```

```
// Incrementa passo de tempo
} while ( (t += Delta_t) <= t_final);
```

São definidos dois vetores, `Q_old[]` e `Q_new[]`, que correspondem as concentrações Q no tempo n e $n+1$, respectivamente. Antes do cálculo das concentrações, os vetores são inicializados, em um simples laço `for`, seguindo a função de concentração inicial $e^{-A(x-b)^2} + s(x)$.

A cada iteração do laço `do-while`, o tempo `t` é incrementado por uma quantidade `Delta_t`, que obedece as regras de estabilidade descritas na seção anterior. Ao longo da iteração, o vetor `Q_new[]` é calculado para as fronteiras e para o centro da malha, em função de `Q_old[]`. Antes do fim da iteração, os vetores `Q_old[]` são atualizados com os valores de `Q_new[]`, o tempo é incrementado, e então a nova iteração é iniciada.

Ao fim da execução, o vetor `Q_new[]`, terá os resultados da concentração de cada volume da malha, correspondente a cada índice do vetor, no tempo `t = t_final`. Os pares índice-concentração são exportados em um arquivo de texto, para cada método, linha-a-linha, para serem lidos e plotados pelo *script* Python.

4. Resultados

Neste capítulo serão descritos os resultados obtidos através da simulação da EDP com a variação de diversos parâmetros. Os parâmetros iniciais escolhidos para este trabalho foram:

- $L_x = 10\text{m}$ (comprimento do domínio)
- $nx = 200$ (número de células)
- $\bar{u} = 2\text{m/s}$ (velocidade de escoamento)
- $t_{\text{final}} = 1\text{s}$ (tempo final de simulação)
- $\Delta x = \frac{L_x}{nx} = 0,05\text{m}$ (passo no espaço)
- $\Delta t = 0,9\left(\frac{\Delta_x}{\bar{u}}\right) = 0,0225\text{s}$ (passo de tempo)
- $A = 100$
- $B = 1,5$
- $C = 4,0$
- $D = 6,0$
- $E = 2,0$

4.1 Forward Time-Backward Space (FTBS)

4.1.1 Resultados para variações de nx

Com a variação de nx, obtiveram-se os seguintes resultados:

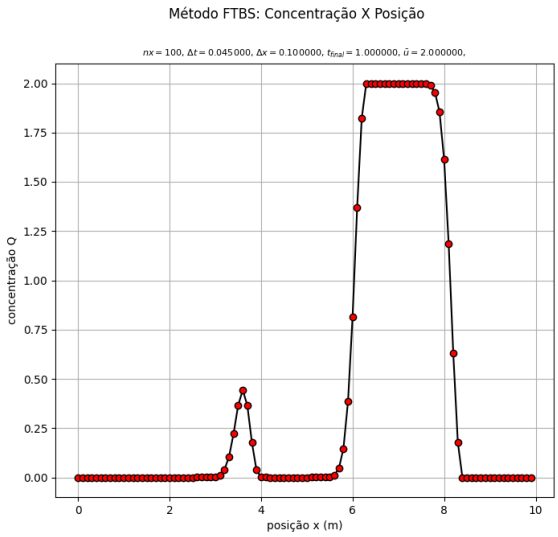

Figura 4.1: FTBS: $nx = 100$

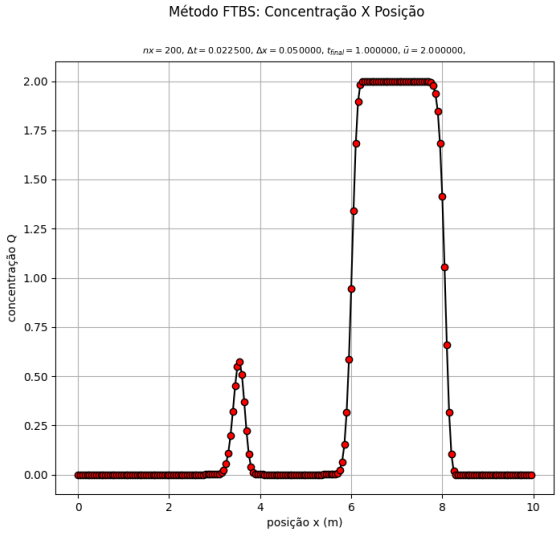

Figura 4.2: FTBS: $nx = 200$

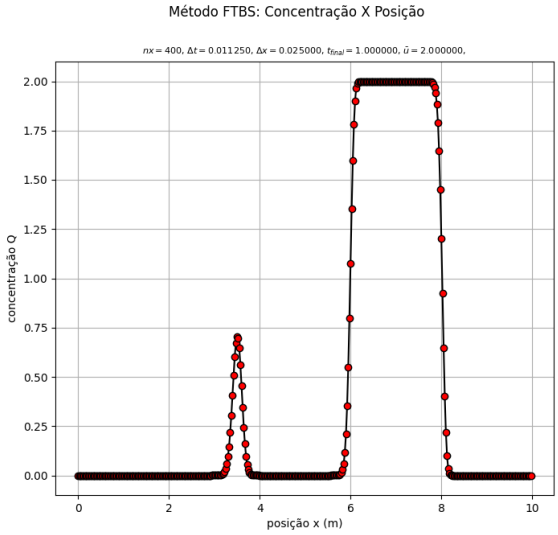

Figura 4.3: FTBS: $nx = 400$

Nota-se que o refinamento da malha resulta em uma curva mais suave, devido ao aumento da resolução. Este aumento no número de nós torna a solução discreta mais próxima da contínua (solução real).

4.1.2 Resultados para variações de t_{final}

Com a variação de t_{final}, obtiveram-se os seguintes resultados:

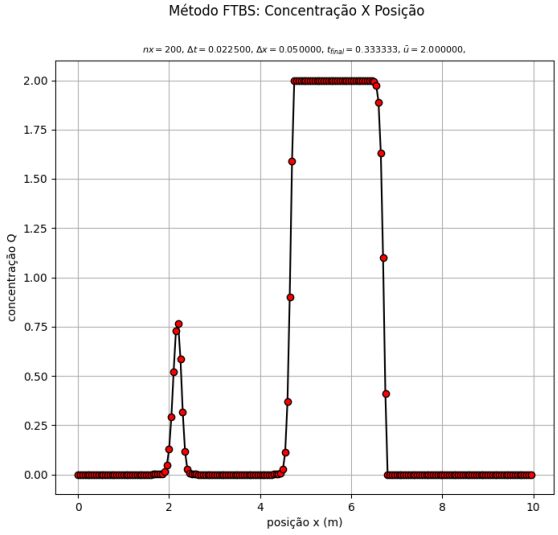

Figura 4.4: FTBS: $t_{\text{final}} = \frac{1}{3}$s

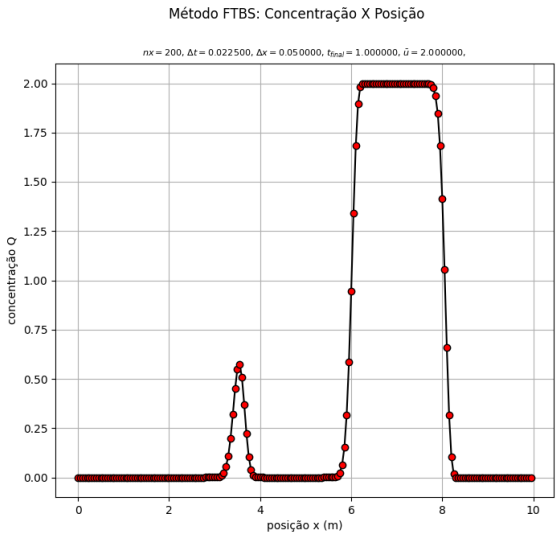

Figura 4.5: FTBS: $t_{\text{final}} = 1,0\text{s}$

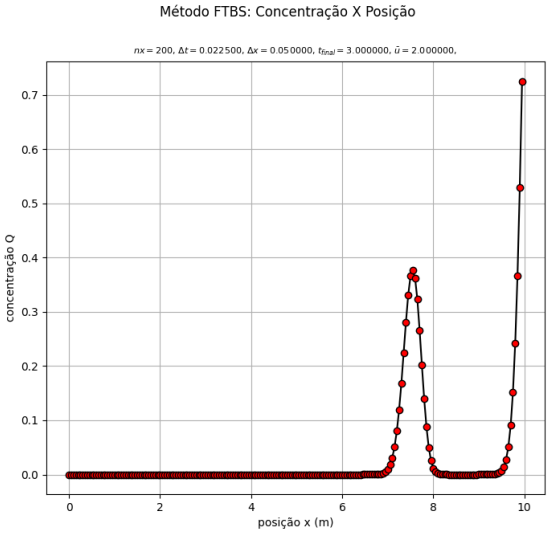

Figura 4.6: FTBS: $t_{\text{final}} = 3,0\text{s}$

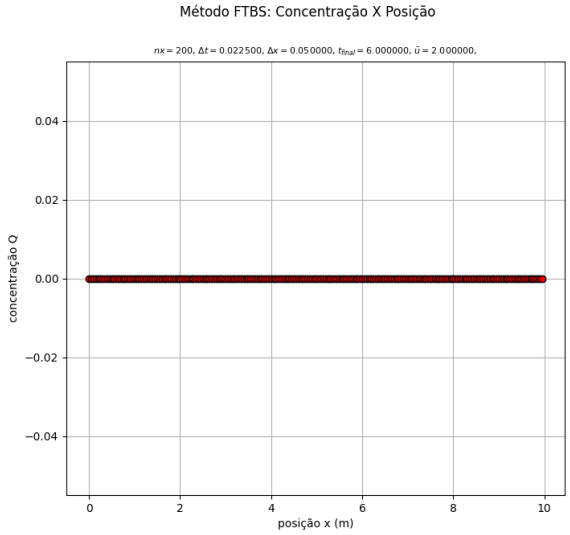

Figura 4.7: FTBS: $t_{\text{final}} = 6,0\text{s}$

Nota-se com o avanço do tempo, a onda, representada no gráfico, tende a se deslocar para a direita, devido ao valor de $\bar{u}$ positivo. Para um tempo grande o suficiente, a concentração se estabiliza em zero.

4.2 Lax-Friedrichs (L-F)

4.2.1 Resultados para variações de nx

Com a variação de nx, obtiveram-se os seguintes resultados:

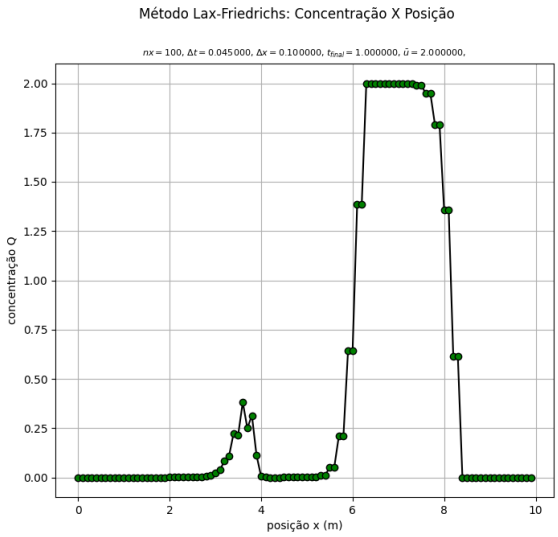

Figura 4.8: L-F: $nx = 100$

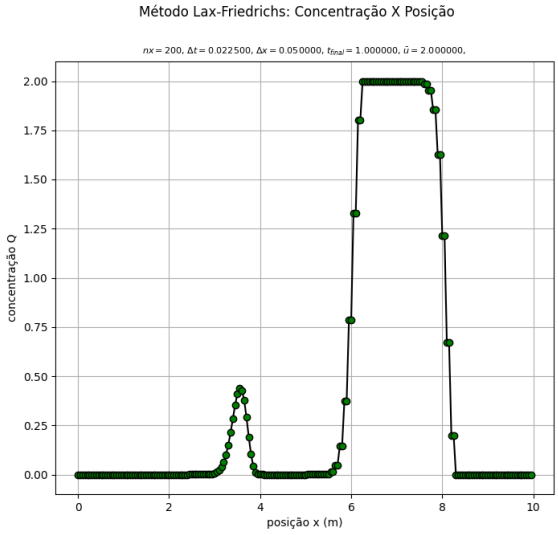

Figura 4.9: L-F: $nx = 200$

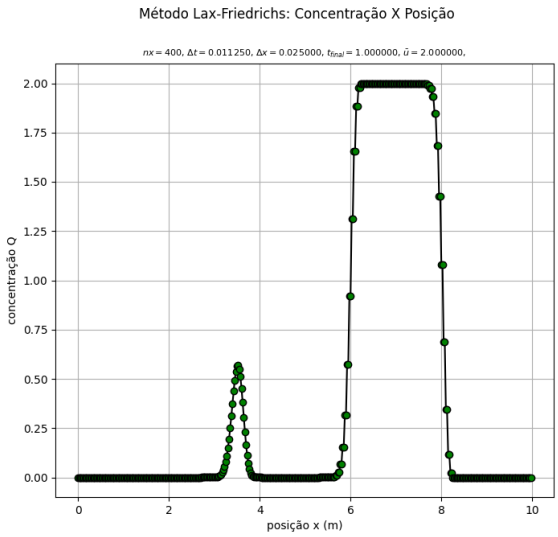

Figura 4.10: L-F: $nx = 400$

Nota-se que o refinamento da malha resulta em uma curva mais suave, devido ao aumento da resolução. Este aumento no número de nós torna a solução discreta mais próxima da contínua (solução real).

4.2.2 Resultados para variações de t_{final}

Com a variação de t_{final}, obtiveram-se os seguintes resultados:

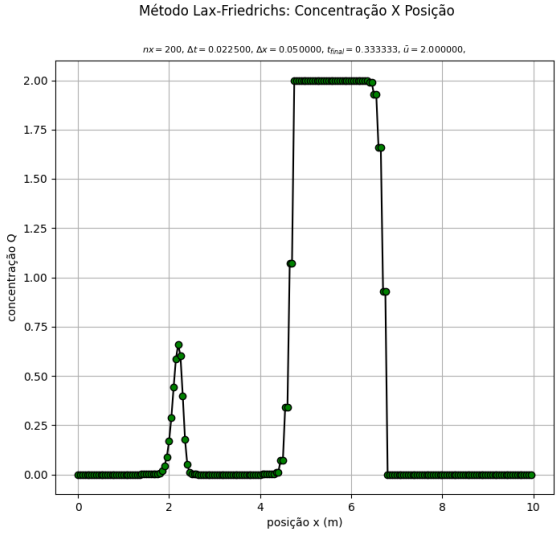

Figura 4.11: L-F: $t_{\text{final}} = \frac{1}{3}$s

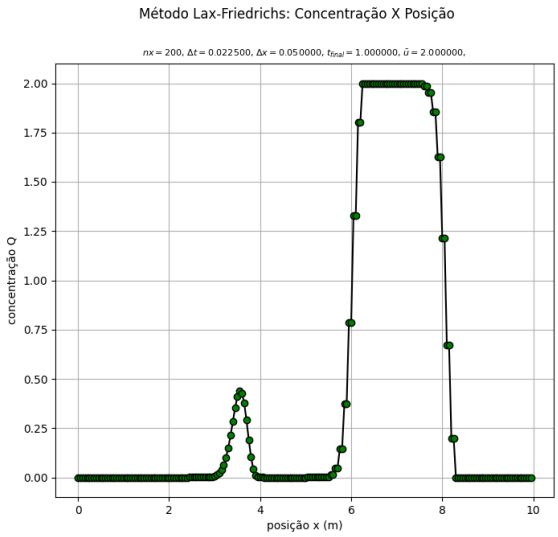

Figura 4.12: L-F: $t_{\text{final}} = 1,0\text{s}$

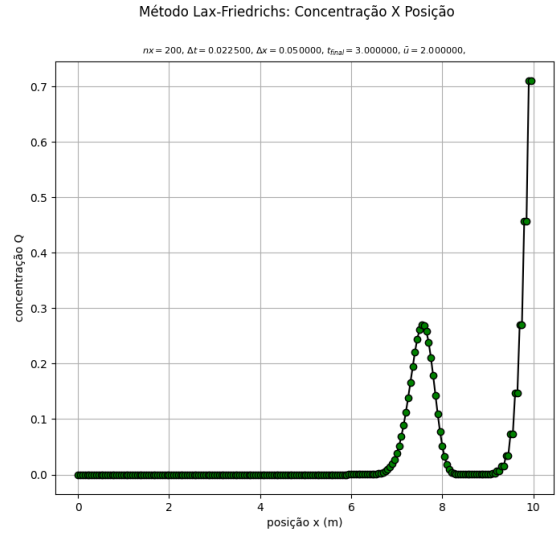

Figura 4.13: L-F: $t_{\text{final}} = 3,0\text{s}$

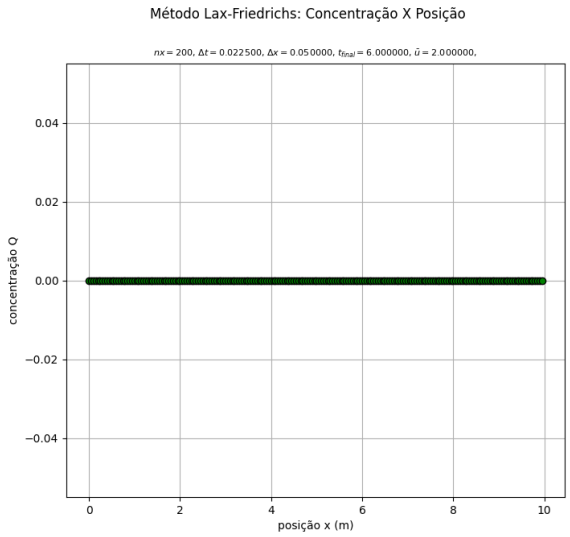

Figura 4.14: L-F: $t_{\text{final}} = 6,0\text{s}$

Nota-se com o avanço do tempo, a onda, representada no gráfico, tende a se deslocar para a direita, devido ao valor de $\bar{u}$ positivo. Para um tempo grande o suficiente, a concentração se estabiliza em zero.

4.3 Lax-Wendroff (L-W)

4.3.1 Resultados para variações de nx

Com a variação de nx, obtiveram-se os seguintes resultados:

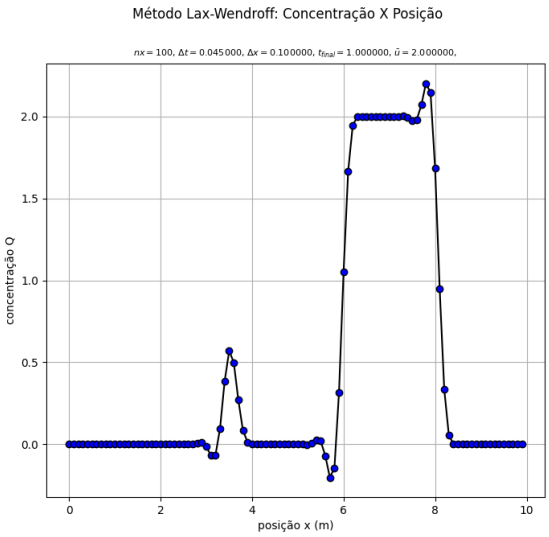

Figura 4.15: L-W: $nx = 100$

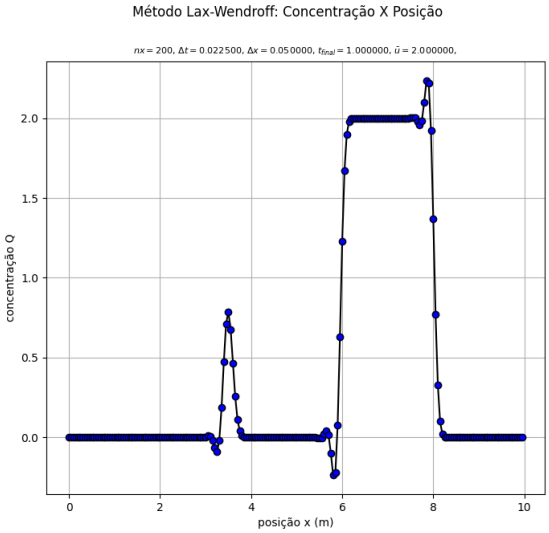

Figura 4.16: L-W: $nx = 200$

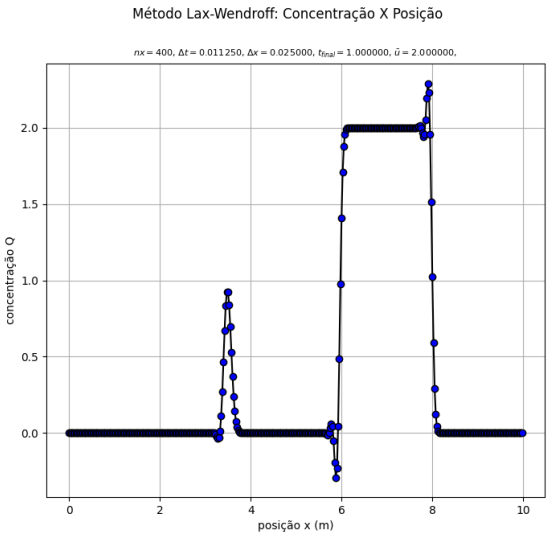

Figura 4.17: L-W: $nx = 400$

Nota-se, com o refinamento da malha, a presença das oscilações numéricas características do método.

Nota-se que o refinamento da malha resulta em uma curva mais suave, devido ao aumento da resolução, além disso, nota-se a presença de oscilações numéricas características do método.

4.3.2 Resultados para variações de t_{final}

Com a variação de t_{final}, obtiveram-se os seguintes resultados:

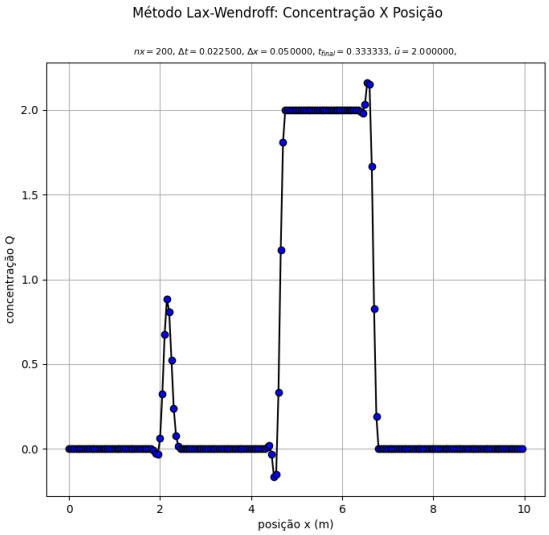

Figura 4.18: L-W: $t_{\text{final}} = \frac{1}{3}$s

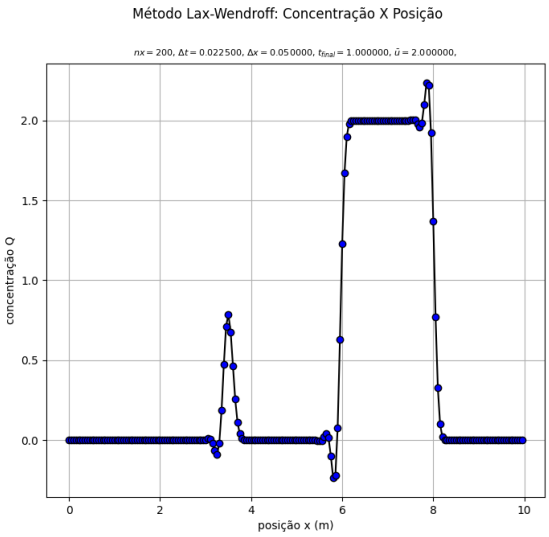

Figura 4.19: L-W: $t_{\text{final}} = 1,0\text{s}$

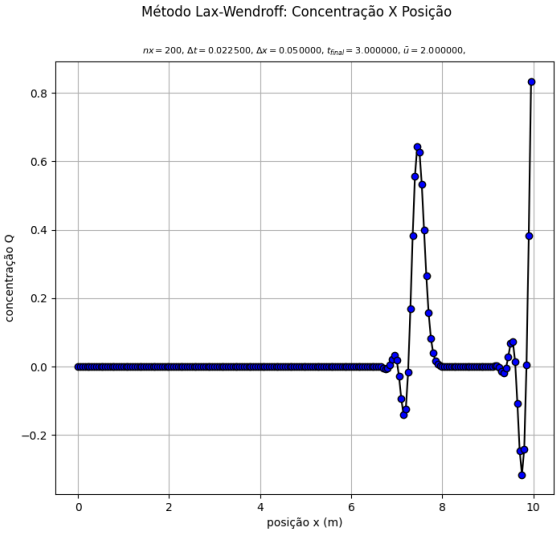

Figura 4.20: L-W: $t_{\text{final}} = 3,0\text{s}$

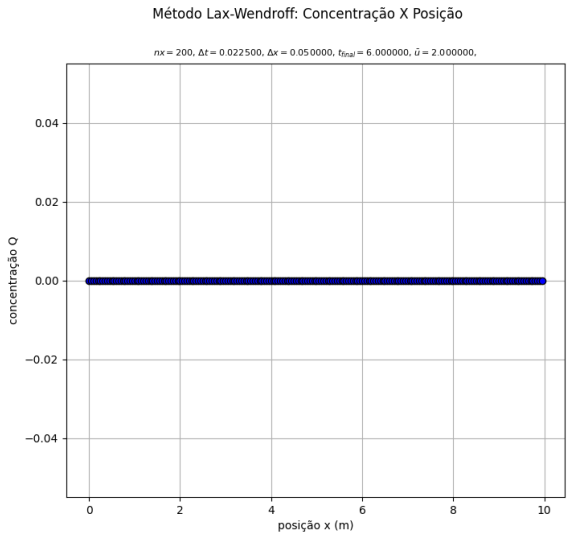

Figura 4.21: L-W: $t_{\text{final}} = 6,0\text{s}$

Nota-se com o avanço do tempo, a onda, representada no gráfico, tende a se deslocar para a direita, devido ao valor de $\bar{u}$ positivo. Para um tempo grande o suficiente, a concentração se estabiliza em zero.

4.4 Beam-Warming (B-W)

4.4.1 Resultados para variações de nx

Com a variação de nx, obtiveram-se os seguintes resultados:

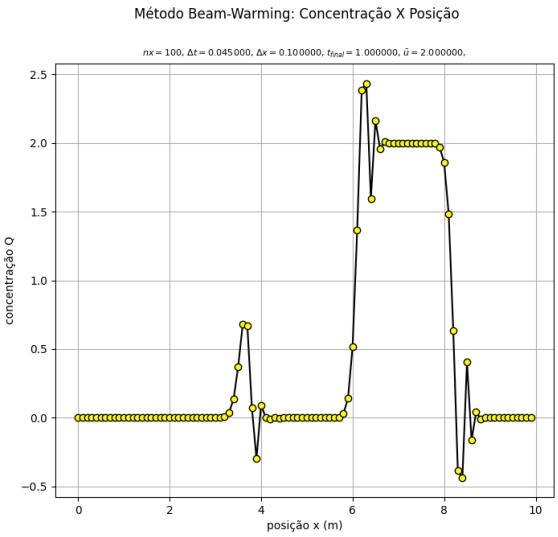

Figura 4.22: B-W: $nx = 100$

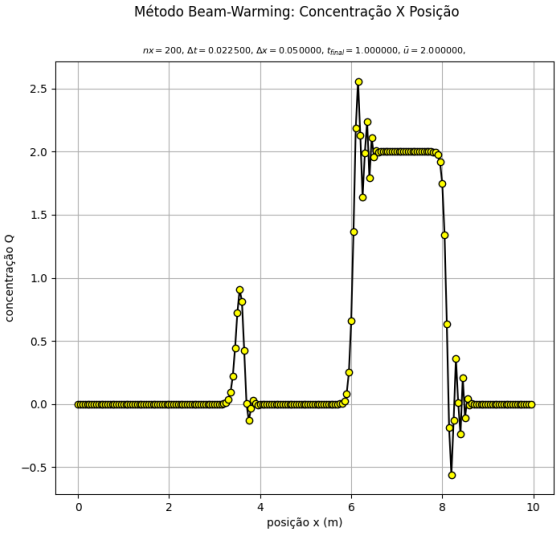

Figura 4.23: B-W: $nx = 200$

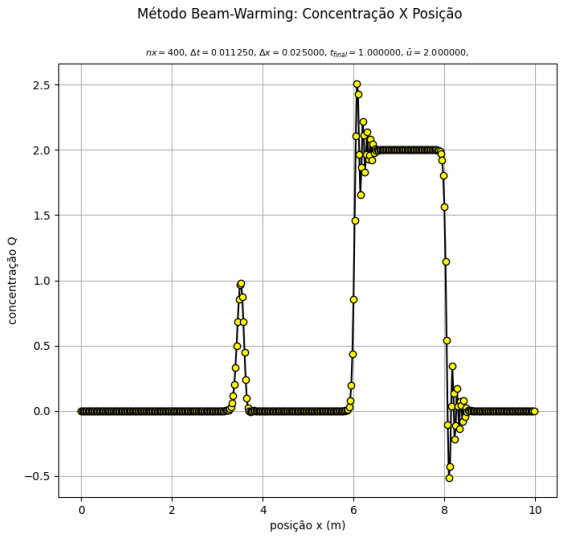

Figura 4.24: B-W: $nx = 400$

Nota-se que o refinamento da malha resulta em uma curva mais suave, devido ao aumento da resolução, além disso, nota-se a presença de oscilações numéricas características do método.

4.4.2 Resultados para variações de t_{final}

Com a variação de t_{final}, obtiveram-se os seguintes resultados:

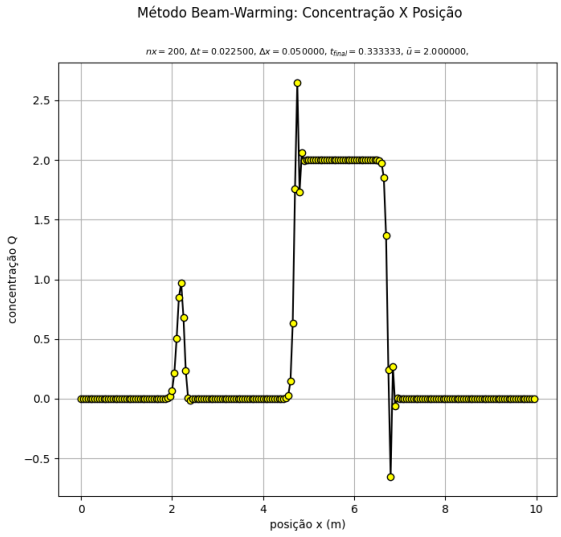

Figura 4.25: B-W: $t_{\text{final}} = \frac{1}{3}$s

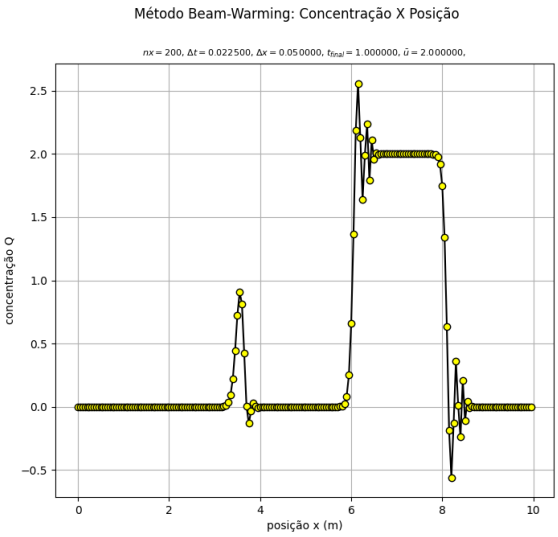

Figura 4.26: B-W: $t_{\text{final}} = 1,0\text{s}$

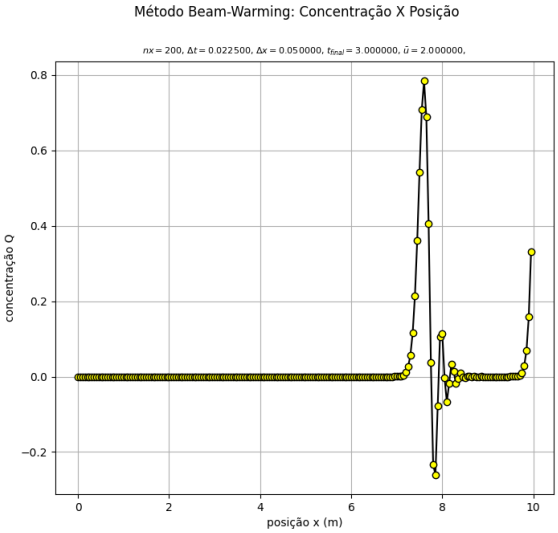

Figura 4.27: B-W: $t_{\text{final}} = 3,0\text{s}$

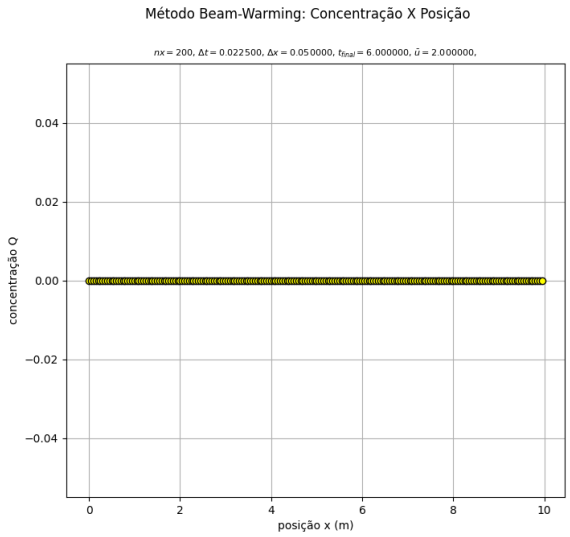

Figura 4.28: B-W: $t_{\text{final}} = 6,0\text{s}$

Nota-se com o avanço do tempo, a onda, representada no gráfico, tende a se deslocar para a direita, devido ao valor de $\bar{u}$ positivo. Para um tempo grande o suficiente, a concentração se estabiliza em zero.

5. Discussão

Observa-se que a aplicação de diferentes métodos numéricos para um mesmo problema — neste caso, a EDP da advecção — resulta em soluções bastante diferentes. Métodos de primeira ordem como FTBS e Lax-Friedrichs, possuem erro de truncamento na ordem de $\frac{\partial^2 c}{\partial x^2}$ o que introduz uma pequena difusão numérica e oscilações. Métodos de segunda ordem como Lax-Wendroff e Beam-Warming apesar de tecnicamente melhores, apresentam erro de truncamento na ordem de $\frac{\partial^3 c}{\partial x^3}$ o que resulta em dispersão numérica e maiores oscilações.

Em relação aos refinamentos da malha, nota-se que todos os métodos se comportam de maneira similar. Uma diminuição no número de nós nx resulta em uma curva mais "pontiaguda" e um aumento deste resulta em uma curva mais "suave". Como consequência, pode-se concluir que o refinamento da malha aproxima os gráficos — pertencentes ao domínio discreto — da solução contínua, dada pela EDP analítica.

Em relação às mudanças em t_{final}, observa-se que novamente os métodos apresentam comportamento similar. Conforme o avanço do tempo, o efeito advectivo faz com que as concentrações se desloquem para a direita, o que está de acordo com a velocidade $\bar{u} > 0$. Dado um tempo suficientemente longo, as concentrações se estabilizam em zero.

6. Conclusão

Equações diferenciais são uma ferramenta poderosa para a modelagem de problemas físicos. A EDP da advecção, muito utilizada na modelagem do escoamento de fluídos, pode ser difícil ou até impossível de resolver analiticamente.

Desta forma, se faz necessário o uso de métodos numéricos, os quais permitem a obtenção de uma solução aproximada para o problema. Diferentes métodos possuem diferentes comportamentos, com vantagens e desvantagens para cada situação. Neste trabalho, foi estudado o comportamento de quatro métodos, na busca da solução da equação da advecção.

Seguindo as condições de estabilidade, foi possível perceber que métodos como o FTBS e Lax-Friedrichs possuem um comportamento bastante suave, com poucas oscilações, mas não tão boa acurácia. Métodos de alta resolução como Lax-Wendroff e Beam-Warming possuem melhor acurácia, mas sofrem de oscilações espúrias que prejudicam sua precisão.

A escolha dentre diferentes métodos é necessária dependendo dos parâmetros do problema físico e do projeto de engenharia a ser estudado. É necessário estabelecer um *tradeoff* entre tempo de projeto vs. acurácia da solução, de maneira a manter os custos sob controle.

7. Referências Bibliográficas

1. **Pedro Amaral Souto**, Helio & **de Souza Boy**, Grazione, *Apostila de "Métodos Numéricos para Equações Diferenciais II"*, 2020

2. **Pedro Amaral Souto**, Helio & **de Souza Boy**, Grazione, *Notas de aula de "Métodos Numéricos para Equações Diferenciais II"*, 2020

8. Código Computacional

8.0.1 Código Principal (`main.h` & `main.c`)

Arquivo `main.h`

```
/*****************************************************************************
 *        Métodos Numéricos para Equações Diferenciais II -- Trabalho 2      *
 *                          Ariel Nogueira Kovaljski                         *
 *****************************************************************************/

/*======================== Parâmetros a serem ajustados ========================*/

#define LX      (10.0)              /* comprimento do domínio (em m)          */
#define NX      (200)               /* número de células                      */
#define DELTA_X (LX/NX)             /* largura de cada célula (em m)          */
#define U_BAR   (2.0)               /* velocidade de escoamento (em m/s)      */
#define T_FINAL (1.0)               /* tempo final da simulação (em segundos) */
#define COURANT (0.9)               /* C: número de courant                   */

#define DELTA_T (COURANT*DELTA_X/U_BAR) /* passo de tempo (em segundos)       */

#define A (100.0)                   /*                                */
#define B (1.5)                     /*                                */
#define C (4.0)                     /* Parâmetros da condição inicial */
#define D (6.0)                     /*                                */
#define E (2.0)                     /*                                */

/*============================================================================*/

/* Métodos utilizados para o cálculo de Q neste trabalho */
enum methods {FTBS, LF, LW, BW};

void listParameters();
void initializeArray(double arr[], int len, double a, double b, double c,
                     double d, double e);
void calculateQ_FTBS(double old_arr[], double new_arr[]);
void calculateQ_LF(double old_arr[], double new_arr[]);
void calculateQ_LW(double old_arr[], double new_arr[]);
void calculateQ_BW(double old_arr[], double new_arr[]);
void printAndSaveResults(double arr[], int len, int method);
```

Arquivo `main.c`

```
/*****************************************************************************
 *        Métodos Numéricos para Equações Diferenciais II -- Trabalho 2      *
 *                          Ariel Nogueira Kovaljski                         *
 *****************************************************************************/

#include <stdio.h>
```

```
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include "main.h"

int main(void)
{
    double Q_new[NX];   /* Array de Q no tempo n+1 */
    double Q_old[NX];   /* Array de Q no tempo n */

    puts("MNED II - Trabalho 2\n"
         "====================\n"
         "por Ariel Nogueira Kovaljski\n");

    listParameters();

    /* Cálculo de Q via método Forward Time-Backward Space (FTBS) */
    initializeArray(Q_old, NX, A, B, C, D, E);
    initializeArray(Q_new, NX, A, B, C, D, E);
    calculateQ_FTBS(Q_old, Q_new);
    printAndSaveResults(Q_new, NX, FTBS);

    /* Cálculo de Q via método Lax-Friedrichs */
    initializeArray(Q_old, NX, A, B, C, D, E);
    initializeArray(Q_new, NX, A, B, C, D, E);
    calculateQ_LF(Q_old, Q_new);
    printAndSaveResults(Q_new, NX, LF);

    /* Cálculo de Q via método Lax-Wendroff */
    initializeArray(Q_old, NX, A, B, C, D, E);
    initializeArray(Q_new, NX, A, B, C, D, E);
    calculateQ_LW(Q_old, Q_new);
    printAndSaveResults(Q_new, NX, LW);

    /* Cálculo de Q via método Beam-Warmimg */
    initializeArray(Q_old, NX, A, B, C, D, E);
    initializeArray(Q_new, NX, A, B, C, D, E);
    calculateQ_BW(Q_old, Q_new);
    printAndSaveResults(Q_new, NX, BW);

    return 0;
}

void listParameters()
{
    puts("Parametros\n----------");
    puts("Constantes da equacao:");
    printf("DELTA_T = %f, DELTA_X = %f, U_BAR = %3.2e\n\n",
           DELTA_T, DELTA_T, U_BAR);
    puts("Constantes da simulacao:");
    printf("NX = %d, T_FINAL = %f\n\n", NX, T_FINAL);
}

/* Inicializa um array para um valor de entrada */
void initializeArray(double arr[], int len, double a, double b, double c,
                     double d, double e)
{
    /*************************************************************
    *
    *                    Condição de contorno
    *                c(x,0) = exp(-A(x-B)²) + s(x)
    *
```

```
    *           onde s(x) = { E,    se  C <= x <= D
    *                       { 0,    c.c.
    */

    int i;
    double x, s;

    for (i = 0; i < len; ++i) {
        x = i * DELTA_X;
        s = (x >= c && x <= d ? e : 0);
        arr[i] = exp( -a * ((x - b)*(x - b)) ) + s;
    }
}

/* Calcula as concentrações na malha ao longo do tempo */
/* Método Forward Time-Backward Space (FTBS) */
void calculateQ_FTBS(double old[], double new[])
{
    int i;
    int progress = 0, progress_count = 0;
    int progress_incr = (T_FINAL/DELTA_T) * 5.0 / 100;
    double t = 0;

    puts("Calculando FTBS...");

    do {
        /*************************************************************
        *
        *            |==@==|==@==|==@==|==@==|==@==|==@==|==@==|
        *               ^
        *            Para o volume da fronteira esquerda
        *        o índice 0 refere-se ao volume nº 1 da malha
        */
        new[0] = old[0];

        /*************************************************************
        *
        *            |==@==|==@==|==@==|==@==|==@==|==@==|==@==|
        *                      ^     ^     ^     ^     ^
        *                  Para os volumes do centro
        *               e na fronteira direita da malha
        */
        for (i = 1; i < NX; ++i) {
            new[i] = old[i] - U_BAR*DELTA_T/DELTA_X * (
                old[i] - old[i-1]
            );
        }

        /* Incrementa o progresso a cada 5% */
        if (progress_count == progress_incr){
            progress_count = 0;
            ++progress;
            printf("\rCalculando... %d%% concluido", progress * 5);
            fflush(stdout);
        }

        /* Atualiza array de valores antigos com os novos para o próximo
        passo de tempo */
        for (i = 0; i < NX; ++i) {
            old[i] = new[i];
        }
```

```
        /* Incrementa contador para cada 5% */
        ++progress_count;

    } while ( (t += DELTA_T) <= T_FINAL );
}

/* Método Lax-Friedrichs */
void calculateQ_LF(double old[], double new[])
{
    int i;
    int progress = 0, progress_count = 0;
    int progress_incr = (T_FINAL/DELTA_T) * 5.0 / 100;
    double t = 0;

    puts("Calculando Lax-Friedrichs...");

    do {
        /*************************************************************
        *
        *            |==@==|==@==|==@==|==@==|==@==|==@==|==@==|
        *               ^
        *            Para o volume da fronteira esquerda
        *        o índice 0 refere-se ao volume nº 1 da malha
        */
        new[0] = (
                   old[1] + old[0]
                )/2 - U_BAR*DELTA_T/(2*DELTA_X) * (
                   old[1] - old[0]
                );

        /*************************************************************
        *
        *            |==@==|==@==|==@==|==@==|==@==|==@==|==@==|
        *                      ^     ^     ^     ^
        *             Para os volumes do centro da malha
        */
        for (i = 1; i < NX - 1; ++i) {
            new[i] = (
                       old[i+1] + old[i-1]
                    )/2 - U_BAR*DELTA_T/(2*DELTA_X) * (
                       old[i+1] - old[i-1]
                    );
        }

        /*************************************************************
        *
        *            |==@==|==@==|==@==|==@==|==@==|==@==|==@==|
        *                                                  ^
        *             Para o volume da fronteira direita
        *            i possui valor de NX - 1 nesse ponto
        */
        new[i] = (
                   old[i] + old[i-1]
                )/2 - U_BAR*DELTA_T/(2*DELTA_X) * (
                   old[i] - old[i-1]
                );

        /* Incrementa o progresso a cada 5% */
        if (progress_count == progress_incr){
            progress_count = 0;
            ++progress;
            printf("\rCalculando... %d%% concluido", progress * 5);
```

```
            fflush(stdout);
        }

        /* Atualiza array de valores antigos com os novos para o próximo
        passo de tempo */
        for (i = 0; i < NX; ++i) {
            old[i] = new[i];
        }

        /* Incrementa contador para cada 5% */
        ++progress_count;

    } while ( (t += DELTA_T) <= T_FINAL );
}

/* Método Lax-Wendroff */
void calculateQ_LW(double old[], double new[])
{
    int i;
    int progress = 0, progress_count = 0;
    int progress_incr = (T_FINAL/DELTA_T) * 5.0 / 100;
    double t = 0;

    puts("Calculando Lax-Wendroff...");

    do {
        /************************************************************
        *
        *            |==@==|==@==|==@==|==@==|==@==|==@==|==@==|
        *               ^
        *            Para o volume da fronteira esquerda
        *        o índice 0 refere-se ao volume nº 1 da malha
        */
        new[0] = old[0] - U_BAR*DELTA_T/(2*DELTA_X) * (
                   old[1] - old[0]
                ) + (U_BAR*U_BAR)*(DELTA_T*DELTA_T)/(2*DELTA_X*DELTA_X) * (
                   old[1] - old[0]
                );

        /************************************************************
        *
        *            |==@==|==@==|==@==|==@==|==@==|==@==|==@==|
        *                     ^     ^     ^     ^
        *             Para os volumes do centro da malha
        */
        for (i = 1; i < NX - 1; ++i) {
            new[i] = old[i] - U_BAR*DELTA_T/(2*DELTA_X) * (
                old[i+1] - old[i-1]
            ) + (U_BAR*U_BAR)*(DELTA_T*DELTA_T)/(2*DELTA_X*DELTA_X) * (
                old[i+1] - 2*old[i] + old[i-1]
            );
        }

        /************************************************************
        *
        *            |==@==|==@==|==@==|==@==|==@==|==@==|==@==|
        *                                              ^
        *             Para o volume da fronteira direita
        *            i possui valor de NX - 1 nesse ponto
        */
        new[i] = old[i] - U_BAR*DELTA_T/(2*DELTA_X) * (
            old[i] - old[i-1]
```

```
        ) + (U_BAR*U_BAR)*(DELTA_T*DELTA_T)/(2*DELTA_X*DELTA_X) * (
            old[i-1] - old[i]
        );

        /* Incrementa o progresso a cada 5% */
        if (progress_count == progress_incr){
            progress_count = 0;
            ++progress;
            printf("\rCalculando... %d%% concluido", progress * 5);
            fflush(stdout);
        }

        /* Atualiza array de valores antigos com os novos para o próximo
        passo de tempo */
        for (i = 0; i < NX; ++i) {
            old[i] = new[i];
        }

        /* Incrementa contador para cada 5% */
        ++progress_count;

    } while ( (t += DELTA_T) <= T_FINAL );
}

/* Método Beam-Warming */
void calculateQ_BW(double old[], double new[])
{
    int i;
    int progress = 0, progress_count = 0;
    int progress_incr = (T_FINAL/DELTA_T) * 5.0 / 100;
    double t = 0;

    puts("Calculando Beam-Warming...");

    do {
        /*************************************************************
        *
        *            |==@==|==@==|==@==|==@==|==@==|==@==|==@==|
        *               ^
        *            Para o volume da fronteira esquerda
        *        o índice 0 refere-se ao volume nº 1 da malha.
        *     O método de Lax-Wendroff é utilizado para este caso
        */
        new[0] = old[0] - U_BAR*DELTA_T/(2*DELTA_X) * (
                   old[1] - old[0]
                ) + (U_BAR*U_BAR)*(DELTA_T*DELTA_T)/(2*DELTA_X*DELTA_X) * (
                   old[1] - old[0]
                );

        /*************************************************************
        *
        *            |==@==|==@==|==@==|==@==|==@==|==@==|==@==|
        *                  ^
        *         Para o volume logo após a fronteira esquerda
        */
        new[1] = old[1] - 3 * U_BAR*DELTA_T/(2*DELTA_X) * (
                   old[1] - old[0]
                ) + (U_BAR*U_BAR)*(DELTA_T*DELTA_T)/(2*DELTA_X*DELTA_X) * (
                   old[1] - old[0]
                );

        /*************************************************************
```

```
        *
        *            |==@==|==@==|==@==|==@==|==@==|==@==|==@==|
        *                           ^      ^      ^      ^
        *                 Para os volumes do centro
        *               e na fronteira direita da malha
        */
        for (i = 2; i < NX; ++i) {
            new[i] = old[i] - U_BAR*DELTA_T/(2*DELTA_X) * (
                        3*old[i] - 4*old[i-1] + old[i-2]
                    ) + (U_BAR*U_BAR)*(DELTA_T*DELTA_T)/(2*DELTA_X*DELTA_X) * (
                        old[i] - 2*old[i-1] + old[i-2]
                    );
        }

        /* Incrementa o progresso a cada 5% */
        if (progress_count == progress_incr){
            progress_count = 0;
            ++progress;
            printf("\rCalculando... %d%% concluido", progress * 5);
            fflush(stdout);
        }

        /* Atualiza array de valores antigos com os novos para o próximo
        passo de tempo */
        for (i = 0; i < NX; ++i) {
            old[i] = new[i];
        }

        /* Incrementa contador para cada 5% */
        ++progress_count;

    } while ( (t += DELTA_T) <= T_FINAL );
}

/* Imprime na tela e salva os resultados num arquivo de saída */
void printAndSaveResults(double arr[], int len, int method)
{
    int i;
    char filename[50], file0[50], file1[50];
    FILE *results_file;     /* Ponteiro para o arquivo de resultados */

    /* Prepara nome do arquivo de saída */
    switch (method) {
        case FTBS:
            snprintf(filename, 50, "%s%d%s", "results", method, ".txt");
            break;
        case LF:
            snprintf(filename, 50, "%s%d%s", "results", method, ".txt");
            break;
        case LW:
            snprintf(filename, 50, "%s%d%s", "results", method, ".txt");
            break;
        case BW:
            snprintf(filename, 50, "%s%d%s", "results", method, ".txt");
            break;
    }

    /* Imprime os resultados no console */
    printf("\n\nQ[%d] (tempo final: %.2fs) = [", NX, T_FINAL);
    for (i = 0; i < len - 1; ++i) {
        printf("%f, ", arr[i]);
    }
```

```
    printf("%f]\n\n", arr[i]);

    /* Error Handling -- Verifica se é possível criar/escrever o arquivo de
                         resultados */
    snprintf(file0, 50, "%s%s", "./results/", filename);
    snprintf(file1, 50, "%s%s", "./../results/", filename);
    if (   ( results_file = fopen(file0, "w") ) == NULL
        && ( results_file = fopen(file1, "w") ) == NULL) {
        fprintf(stderr, "[ERR] Houve um erro ao escrever o arquivo \"%s\"! "
                        "Os resultados nao foram salvos.\n", filename);
        exit(1);
    }

    /* Adiciona os resultados no arquivo "results.txt" */
    fprintf(results_file,
            "nx=%d\n"
            "Delta_t=%f\n"
            "Delta_x=%f\n"
            "t_final=%f\n"
            "u_bar=%f\n",
            NX, DELTA_T, DELTA_X, T_FINAL, U_BAR);
    fputs("********************\n", results_file);

    for (i = 0; i < len; ++i) {
        fprintf(results_file, "%f,%f\n", i * DELTA_X, arr[i]);
    }

    fclose(results_file);   /* Fecha o arquivo */

    printf("[INFO] Os resultados foram salvos no arquivo \"%s\" "
           "no diretorio \"results/\".\n\n", filename);
}
```

8.0.2 Código do Gráfico (`plot_graph.py`)

```
################################################################################
#         Métodos Numéricos para Equações Diferenciais II -- Trabalho 2        #
#                             Ariel Nogueira Kovaljski                         #
################################################################################

import matplotlib.pyplot as plt


x0, y0 = [], []
x1, y1 = [], []
x2, y2 = [], []
x3, y3 = [], []
parameters = {}

def main():
    # Abre arquivos para leitura
    # Arquivo FTBS
    try:
        f0 = open('./results/results0.txt', 'r')
    except OSError as err:
        print("Erro ao abrir o arquivo 'results0.txt':", err)

    # Arquivo Lax-Friedrichs
    try:
        f1 = open('./results/results1.txt', 'r')
    except OSError as err:
        print("Erro ao abrir o arquivo 'results1.txt':", err)
```

```
    # Arquivo Lax-Wendroff
    try:
        f2 = open('./results/results2.txt', 'r')
    except OSError as err:
        print("Erro ao abrir o arquivo 'results2.txt':", err)

    # Arquivo Beam-Warming
    try:
        f3 = open('./results/results3.txt', 'r')
    except OSError as err:
        print("Erro ao abrir o arquivo 'results3.txt':", err)

    # Extrai informações dos arquivos
    data_extract(f0, x0, y0)
    data_extract(f1, x1, y1)
    data_extract(f2, x2, y2)
    data_extract(f3, x3, y3)

    # Plota os gráficos de cada método
    plot_graph(x0, y0, 0)
    plot_graph(x1, y1, 1)
    plot_graph(x2, y2, 2)
    plot_graph(x3, y3, 3)

    # Exibe os gráficos
    plt.show()


def data_extract(f, x, y):
    for line_number, line in enumerate(f):
        if (line_number + 1) < 6:
            # Adiciona parâmetros da simulação em um dicionário
            parameters[line.split('=')[0]] = (line.split('=')[1]).split('\n')[0]
        elif (line_number + 1) == 6:
            # Pula linha separadora
            pass
        else:
            # Separa valores na lista `x` e na lista `y`
            x.append( float(line.split(',')[0]) )
            y.append( float((line.split(',')[1]).split('\n')[0]) )


def plot_graph(x, y, id):
    methods = {0:'FTBS', 1:'Lax-Friedrichs', 2:'Lax-Wendroff', 3:'Beam-Warming'}
    colors  = {0: 'red', 1: 'green',         2: 'blue',        3: 'yellow'}
    fig,ax  = plt.subplots()
    fig.set_size_inches(8, 7)
    ax.grid(True)

    plt.suptitle(f"Método {methods[id]}: Concentração X Posição")
    plt.title(rf"$nx = {parameters['NX']}$, "
              rf"$\Delta t = {parameters['DELTA_T']}$, "
              rf"$\Delta x = {parameters['DELTA_X']}$, "
              rf"$t_{{final}} = {parameters['T_FINAL']}$, "
              rf"$\bar{{u}} = {parameters['U_BAR']}$, ", fontsize=8)

    plt.xlabel("posição x (m)")
    plt.ylabel("concentração Q")
    plt.plot(x,y,'ko-', markerfacecolor=colors[id], markeredgecolor='k')

if __name__ == "__main__":
```

```
main()
```