Trabalho 1 de Métodos Numéricos para Equações Diferenciais II

Ariel Nogueira Kovaljski

Nova Friburgo, XX de outubro de 2020

Sumário

1 Introdução 3
1.1 A Equação de Advecção-Difusão 3
1.2 Método dos Volumes Finitos 4

2 Desenvolvimento 7
2.1 Condições Inicial e de Contorno 7
2.2 Consistência, Convergência e Estabilidade 8
2.2.1 Consistência 8
2.2.2 Convergência 8
2.2.3 Estabilidade 9
2.3 Programação 10

3 Resultados 12
3.1 Resultados para variações de L_x 12
3.2 Resultados para variações de nx 15
3.3 Resultados para variações de $\bar{u}$ 16
3.4 Resultados para variações de α 17
3.5 Resultados para $\bar{u} = 0$ e variações de α 20
3.6 Resultados para variações de c_{ini} 22
3.7 Resultados para variações de c_{inj} 23
3.8 Resultados para variações de t_{final} 25

4 Conclusão 28

5 Código Computacional 29
5.0.1 Código Principal 29
5.0.2 Código do Gráfico (`plot_graph.py`) 32

Lista de Figuras

1.1 Efeitos difusivos e advectivos observados no despejo de esgoto no Rio Bengala 3
1.2 Partição do domínio da solução 4
1.3 Posições ao redor do i-ésimo volume da malha 4

3.1 $L_x = 50\text{m}$ 13
3.2 $L_x = 100\text{m}$ 13
3.3 $L_x = 200\text{m}$ 14
3.4 $L_x = 400\text{m}$ 14
3.5 $nx = 25$ 15
3.6 $nx = 50$ 15
3.7 $nx = 100$ 16
3.8 $\bar{u} = 2 \times 10^{-2}\text{m/s}$ 16
3.9 $\bar{u} = 2 \times 10^{-1}\text{m/s}$ 17
3.10 $\bar{u} = 2 \times 10^{0}\text{m/s}$ 17
3.11 $\alpha = 2.0 \times 10^{-4}$ 18
3.12 $\alpha = 2.0 \times 10^{-2}$ 18
3.13 $\alpha = 2.0 \times 10^{-1}$ 19
3.14 $\alpha = 2.0 \times 10^{0}$ 19
3.15 $\bar{u} = 0$ & $\alpha = 2.0 \times 10^{-4}$ 20
3.16 $\bar{u} = 0$ & $\alpha = 2.0 \times 10^{-2}$ 20
3.17 $\bar{u} = 0$ & $\alpha = 2.0 \times 10^{-1}$ 21
3.18 $\bar{u} = 0$ & $\alpha = 2.0 \times 10^{0}$ 21
3.19 $c_{\text{ini}} = 0.5\text{mol/m}^3$ 22
3.20 $c_{\text{ini}} = 1.0\text{mol/m}^3$ 22
3.21 $c_{\text{ini}} = 2.0\text{mol/m}^3$ 23
3.22 $c_{\text{inj}} = 0.75\text{mol/m}^3$ 24
3.23 $c_{\text{inj}} = 1.5\text{mol/m}^3$ 24
3.24 $c_{\text{inj}} = 3.0\text{mol/m}^3$ 25
3.25 $t_{\text{final}} = 150\text{s}$ 25
3.26 $t_{\text{final}} = 300\text{s}$ 26
3.27 $t_{\text{final}} = 600\text{s}$ 26
3.28 $t_{\text{final}} = 900\text{s}$ 27

1. Introdução

Neste trabalho foi implementado um método computacional de maneira a resolver a equação de Advecção-Difusão de forma numérica.

Para melhor entender o desenvolvimento, é necessária introdução de alguns conceitos-chave utilizados.

1.1 A Equação de Advecção-Difusão

A equação de advecção-difusão possibilita a solução de problemas envolvendo variações espaciais e temporais da concentração de uma substância escoando em um fluído. Um exemplo bastante didático consiste no despejo de esgoto em um afluente: o contaminante sofrerá efeitos difusivos — concentrando-se ao redor da saída — e efeitos advectivos — sendo carregado no sentido da correnteza.

Figura 1.1: Efeitos difusivos e advectivos observados no despejo de esgoto no Rio Bengala

Para um problema unidimensional tem-se a seguinte forma,

$$\frac{\partial c}{\partial t} + \frac{\partial}{\partial x}(uc) - \frac{\partial}{\partial x}\left(D\frac{\partial c}{\partial x}\right) = 0 \tag{1.1}$$

onde c indica a concentração, u a velocidade e D o coeficiente de difusão.

Considerando que para u e D constantes, tem-se $\bar{u}$ e α, respectivamente, é possível reescrever Eq. 1.1 como,

$$\frac{\partial c}{\partial t} + \bar{u}\frac{\partial c}{\partial x} - \alpha\frac{\partial^2 c}{\partial x^2} = 0 \tag{1.2}$$

1.2 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.

A partir da Eq. 1.1, para adaptá-la ao métodos dos volumes finitos, é possível reescrevê-la como

$$\frac{\partial \Phi}{\partial t} + \frac{\partial f}{\partial x} = 0 \tag{1.3}$$

onde,

$$\Phi = c \qquad (1.4) \qquad\qquad f = f(c) = uc - D\frac{\partial c}{\partial x} \qquad (1.5)$$

Para a solução, considera-se um domínio discretizado e subdividido conforme figuras abaixo:

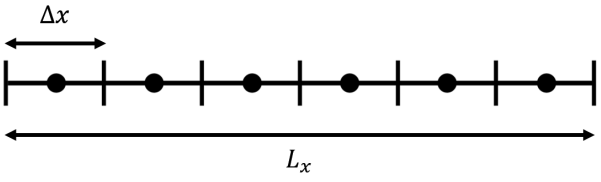

Figura 1.2: Partição do domínio da solução

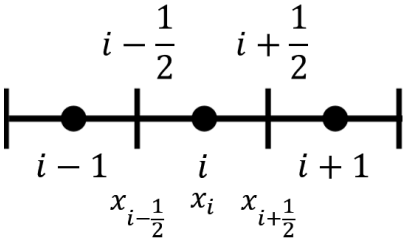

Figura 1.3: Posições ao redor do i-ésimo volume da malha

Para obter-se a discretização do domínio, integra-se a Eq. 1.3 no intervalo de tempo de t^n a t^{n+1} e no espaço de $x_{i-\frac{1}{2}}$ a $x_{i+\frac{1}{2}}$. Além disso, define-se os incrementos no tempo e no espaço como

$$\Delta t = t^{n+1} - t^n \qquad \text{e} \qquad \Delta x = x_{i+\frac{1}{2}} - x_{i-\frac{1}{2}}$$

Tem-se, assim, uma integração dupla

$$\int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} \left(\int_{t^n}^{t^{n+1}} \frac{\partial \Phi}{\partial t} dt \right) dx + \int_{t^n}^{t^{n+1}} \left(\int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} \frac{\partial f}{\partial x} dx \right) dt = 0$$

$$\int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} [\Phi(x, t^{n+1}) - \Phi(x, t^n)] dx + \int_{t^n}^{t^{n+1}} [f(x_{i+\frac{1}{2}}, t) - f(x_{i-\frac{1}{2}}, t)] dt = 0 \quad (1.6)$$

Define-se uma variável Q_i^k que consiste no valor médio aproximado de Φ no espaço, dado um tempo k onde $k = t^n$ ou $k = t^{n+1}$

$$Q_i^n \approx \frac{1}{\Delta x} \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} \Phi(x, t^n) dx \quad (1.7)$$

$$Q_i^{n+1} \approx \frac{1}{\Delta x} \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} \Phi(x, t^{n+1}) dx \quad (1.8)$$

Analogamente, define-se os fluxos como um F_k^n que consiste no valor médio aproximado de F no tempo, dada uma posição k onde $k = x_{i-\frac{1}{2}}$ ou $k = x_{i+\frac{1}{2}}$

$$F_{i-\frac{1}{2}}^n \approx \frac{1}{\Delta t} \int_{t^n}^{t^{n+1}} f(x_{i-\frac{1}{2}}, t) dt \quad (1.9)$$

$$F_{i+\frac{1}{2}}^n \approx \frac{1}{\Delta t} \int_{t^n}^{t^{n+1}} f(x_{i+\frac{1}{2}}, t) dt \quad (1.10)$$

Substituindo as Eq. 1.7, 1.8, 1.9 e 1.10 na Eq. 1.6 e dividindo todos os termos por Δx tem-se que

$$\frac{1}{\Delta x} \left\{ \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} [\Phi(x, t^{n+1}) - \Phi(x, t^n)] dx \right\} + \frac{1}{\Delta x} \left\{ \int_{t^n}^{t^{n+1}} [f(x_{i+\frac{1}{2}}, t) - f(x_{i-\frac{1}{2}}, t)] dt \right\} = 0$$

$$Q_i^{n+1} - Q_i^n + \frac{1}{\Delta x} \left\{ \int_{t^n}^{t^{n+1}} [f(x_{i+\frac{1}{2}}, t) - f(x_{i-\frac{1}{2}}, t)] dt \right\} = 0$$

$$Q_i^{n+1} = Q_i^n + \frac{\Delta t}{\Delta t \Delta x} \left\{ \int_{t^n}^{t^{n+1}} [f(x_{i+\frac{1}{2}}, t) - f(x_{i-\frac{1}{2}}, t)] dt \right\}$$

Por fim, obtém-se

$$Q_i^{n+1} = Q_i^n - \frac{\Delta t}{\Delta x} (F_{i+\frac{1}{2}}^n - F_{i-\frac{1}{2}}^n) \quad (1.11)$$

onde o valor médio de Φ no volume de controle, Q, deve ser atualizado iterativamente para o próximo passo de tempo.

Para a aproximação dos fluxos, dentre as possíveis metodologias, foi escolhido o uso de uma aproximação *upwind* para a concentração c:

$$c \approx \frac{(Q_i^n - Q_{i-1}^n)}{\Delta x} \quad (1.12)$$

e uma aproximação centrada para a derivada espacial da concentração $\frac{\partial c}{\partial x}$:

$$\frac{\partial c}{\partial x} \approx \frac{(Q_{i+1}^n - 2Q_i^n + Q_{i-1}^n)}{\Delta x^2} \tag{1.13}$$

que, ao serem substituídas na Eq. 1.11, resultam em

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

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

$$Q_i^{n+1} = Q_i^n - \Delta t\left[\bar{u}\frac{(Q_i^n - Q_{i-1}^n)}{\Delta x} - \alpha\frac{(Q_{i+1}^n - 2Q_i^n + Q_{i-1}^n)}{\Delta x^2}\right] \tag{1.14}$$

2. Desenvolvimento

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

2.1 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. No caso da EDP discretizada (Eq. 1.14), o mesmo se aplica.

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

Nota-se que há uma dependência temporal do termo futuro Q_i^{n+1} em relação aos termos Q presentes em n, e seus vizinhos espaciais i, $i \pm 1$. A partir desta relação é possível perceber que, para se calcular o primeiro termo, Q^1, é necessário um termo de partida, Q^0. Se tem, assim, a necessidade do estabelecimento de uma condição inicial. Neste trabalho, considera-se uma concentração inicial, c_{ini}, constante para toda a malha.

Mudando o foco para os volumes da malha, nos volumes $i = 0$ e $i = 6$ há uma dependência de termos localizados além do seu domínio — Q_0^n e Q_{nx+1}^n, respectivamente. Para resolver este problema, neste trabalho foram adotadas as seguintes condições de contorno:

$$c(x = 0, t) = c_{\text{inj}} \qquad (2.1) \qquad \left(\frac{\partial c}{\partial x} \right)^t_{x=L_x} = 0 \qquad (2.2)$$

Aliado a estas condições, utiliza-se o conceito de *volumes fantasmas*, para a definição destas condições no discreto. É possível redefinir as condições de contorno como uma média entre os dois volumes adjacentes. Ao se realizar tal construção para a fronteira esquerda, obtém-se,

$$c_{\text{inj}} = \frac{Q_0^n + Q_1^n}{2}$$

$$Q_0^n = 2c_{\text{inj}} - Q_1^n \qquad (2.3)$$

Analogamente, para a fronteira direita, obtém-se,

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

$$Q^n_{nx+1} = Q^n_{nx} \tag{2.4}$$

A partir de ambas as relações, definem-se as equações discretas. Para o contorno esquerdo,

$$Q^{n+1}_1 = Q^n_1 - \Delta t \left[\bar{u} \frac{(Q^n_1 - (2c_{\text{inj}} - Q^n_1))}{\Delta x} - \alpha \frac{(Q^n_2 - 2Q^n_1 + (2c_{\text{inj}} - Q^n_1))}{\Delta x^2} \right]$$

$$Q^{n+1}_1 = Q^n_1 - \Delta t \left[\bar{u} \frac{(2Q^n_1 - 2c_{\text{inj}})}{\Delta x} - \alpha \frac{(Q^n_2 - 3Q^n_1 + 2c_{\text{inj}})}{\Delta x^2} \right] \tag{2.5}$$

e para o contorno direito,

$$Q^{n+1}_{nx} = Q^n_{nx} - \Delta t \left[\bar{u} \frac{(Q^n_{nx} - Q^n_{nx-1})}{\Delta x} - \alpha \frac{(Q^n_{nx} - 2Q^n_{nx} + Q^n_{nx-1})}{\Delta x^2} \right]$$

$$Q^{n+1}_{nx} = Q^n_{nx} - \Delta t \left[\bar{u} \frac{(Q^n_{nx} - Q^n_{nx-1})}{\Delta x} - \alpha \frac{(Q^n_{nx-1} - Q^n_{nx})}{\Delta x^2} \right] \tag{2.6}$$

2.2 Consistência, Convergência e Estabilidade

A análise da consistência, convergência e estabilidade de uma EDP tem como finalidade garantir que a solução numérica do problema — calculada por algoritmos — se aproxime o máximo possível da solução real, com algumas observações.

2.2.1 Consistência

Se trata da equivalência da forma algorítmica da EDP em relação a sua forma analítica. Um método numérico é dito *consistente* quando, através de operações algébricas, é possível recuperar a EDP original.

2.2.2 Convergência

Se trata da aproximação dos valores numéricos do algoritmo à solução analítica da EDP, dado um certo número de iterações. Um método numérico é dito *convergente* quando este sempre irá tender aos valores da solução, não importando o número de iterações.

A análise direta da convergência de algoritmo é muito difícil, mesmo para os casos mais fáceis. Uma possível saída para esse problema é utilizar o Teorema da Equivalência de Lax, que diz:

> "Para um problema linear de valor inicial bem-posto e um método de discretização consistente, estabilidade é condição necessária e suficiente para a convergência." (Peter Lax)

Como a Eq. 1.14 foi obtida a partir da EDP analítica, é certa a sua consistência, restando assim, a determinação de sua estabilidade para a garantia da convergência.

2.2.3 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.

Para se determinar a estabilidade da Eq. 1.14, é possível reescrevê-la para a análise através do método de Von Neumann.

$$(Q^*)_i^{n+1} = (s+C)(Q^*)_{i-1}^n + (1-2s+C)(Q^*)_i^n + s(Q^*)_{i+1}^n \tag{2.7}$$

onde Q^* é solução numérica da EDP, sujeita aos erros de arredondamento do computador, $s = \frac{\alpha \Delta t}{\Delta x^2}$ e $C = \frac{\bar{u}\Delta t}{\Delta x}$ (número de Courant). Além disso, define-se uma nova equação,

$$\xi_i^{n+1} = (s+C)\xi_{i-1}^n + (1-2s+C)\xi_i^n + s\xi_{i+1}^n \tag{2.8}$$

onde $\xi_i^n = Q_i^n - (Q^*)_i^n$ é o erro numérico introduzido em cada ponto da malha.

Seguindo o método de Von Neumann, é possível expandir cada termo ξ_i^n em uma série de Fourier,

$$\xi_i^n = (G)^n e^{j\theta i} \tag{2.9}$$

onde j é a unidade imaginária e G é um fator de amplificação, de forma que,

$$\xi_n^{n+1} = G\xi_i^n \tag{2.10}$$

e a estabilidade é garantida se $|G| \leq 1$. Substituindo esses termos em Eq. 2.8, obtém-se a seguinte equação:

$$G^{n+1}e^{j\theta i} = (s+C)G^n e^{j\theta(i-1)} + (1-2s+C)G^n e^{j\theta i} + sG^n e^{j\theta(i+1)} \tag{2.11}$$

Dividindo-a por $G^n e^{j\theta i}$,

$$G = (s+C)e^{-j\theta} + (1-2s+C) + se^{j\theta}$$

A partir das relações de Euler, $e^{-j\theta} = \cos\theta - j\sin\theta$ & $e^{j}\theta = \cos\theta + j\sin\theta$, é possível reescrevê-la como,

$$G = (s+C)e^{-j\theta} + (1-2s+C) + se^{j\theta}$$

$$G = 1 - (2s+C)(1-\cos\theta) - jC\sin\theta$$

tendo assim,

$$|G| = \sqrt{[1-(2s+C)(1-\cos\theta)]^2 + C^2\sin^2\theta} \;\leq\; 1 \tag{2.12}$$

O caso limite para esta construção ocorre quando $\sin\theta = \pm 1$ e $\cos\theta = 0$. É necessário analisar quais são os valores de s e C que mantém a veracidade da inequação.

$$|G| = \sqrt{[1-(2s+C)(1)]^2 + C^2} \;\leq\; 1$$

Se $2s + C \leq 1$ a condição é satisfeita. O que acontece, porém, no novo caso limite de $2s + C = 1$?

$$|G| = \sqrt{(1-1)^2 + C^2} = \sqrt{C^2} = C$$

Como $2s + C = 1$, C **tem** que ser menor que 1 para um $s \neq 0$. Desta forma,

$$2\frac{\alpha\Delta t}{\Delta x^2} + \frac{\bar{u}\Delta t}{\Delta x} \leq 1$$

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

$$\Delta t \leq \frac{1}{\frac{2\alpha}{\Delta x^2} + \frac{\bar{u}}{\Delta x}} \tag{2.13}$$

Tem-se, então, um método *condicionalmente estável*, ou seja, que depende de uma faixa de valores para garantir a estabilidade de seu funcionamento.

2.3 Programação

Aliado destes conceitos, foi possível construir um programa em linguagem C que calcula as concentrações para cada célula ao longo do tempo, exportando um arquivo de texto com os resultados; este arquivo, então, é lido por um *script* Python que gera um gráfico correspondente.

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[];

// Inicializa ambos os vetores com os valores da concentração inicial (c_ini)
inicializaVetor(Q_antigo)
inicializaVetor(Q_novo)

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

// Cálculo do volume da fronteira esquerda
Q_new[0] = Q_old[0] - Delta_t/Delta_x * (
            u_bar * (2*Q_old[0] - 2*c_inj)
          - alpha * (Q_old[1] - 3*Q_old[0] + 2*c_inj) / Delta_x
         );

// Cálculo de volumes do centro da malha
for (x = 1; x < nx - 1; ++x) {
    Q_new[x] = Q_old[x] - Delta_t/Delta_x * (
                u_bar * (Q_old[x] - Q_old[x-1])
              - alpha * (Q_old[x+1] - 2*Q_old[x] + Q_old[x-1]) / Delta_x
             );

// Cálculo do volume da fronteira direita
Q_new[x] = Q_old[x] - Delta_t/Delta_x * (
            u_bar * (Q_old[x] - Q_old[x-1])
          - alpha * (Q_old[x-1] - Q_old[x]) / Delta_x
         );

// Atualiza vetores antigos para a próxima iteração
for (x = 0; x < nx; ++x) {
    Q_old[x] = Q_new[x];
}

// 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`, com os valores de concentração inicial c_{ini}.

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 célula da malha, correspondente a cada índice do vetor, no tempo `t = t_final`. Os pares índice-concentração são exportados em um arquivo de texto, linha-a-linha, para serem lidos e plotados pelo *script* Python.

3. 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:

- $L_x = 100\text{m}$ (comprimento do domínio)
- $nx = 50$ (número de células)
- $\bar{u} = 0.2\text{m/s}$ (velocidade de escoamento)
- $\alpha = 2.0 \times 10^{-4}$ (coeficiente de difusão)
- $c_{\text{ini}} = 1.0\text{mol/m}^{\mathbf{3}}$ (concentração inicial)
- $c_{\text{inj}} = 1.5\text{mol/m}^{\mathbf{3}}$ (concentração de injeção)
- $t_{\text{final}} = 300\text{s}$ (tempo final de simulação)
- $\Delta t = 0.1 \left(\dfrac{1}{\dfrac{2\alpha}{\Delta x^2} + \dfrac{\bar{u}}{\Delta x}} \right) \approx 0.999001$ (passo de tempo)

3.1 Resultados para variações de L_x

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

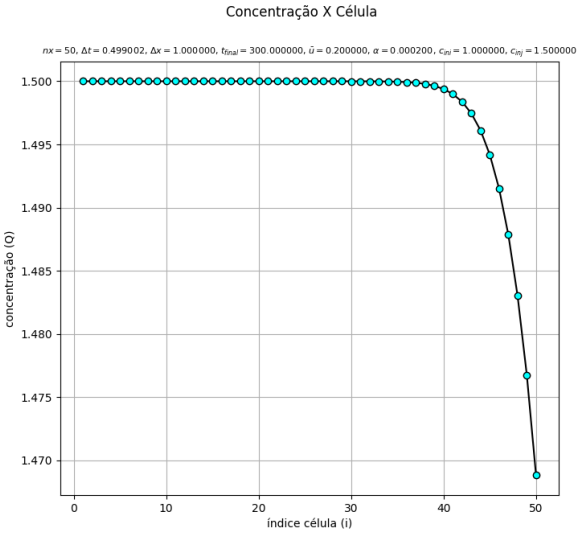

Figura 3.1: $L_x = 50\text{m}$

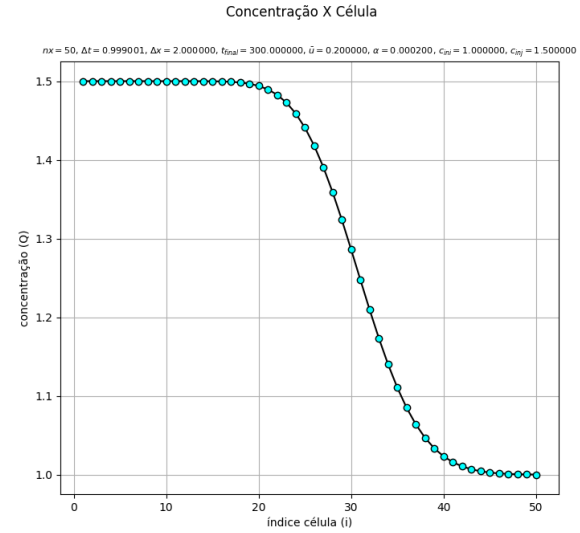

Figura 3.2: $L_x = 100\text{m}$

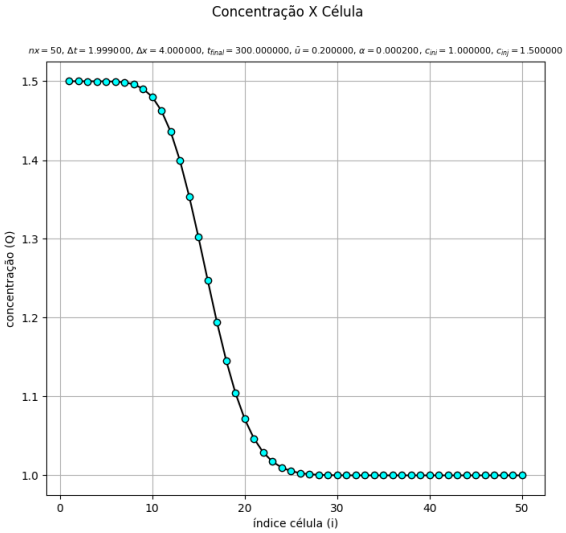

Figura 3.3: $L_x = 200\text{m}$

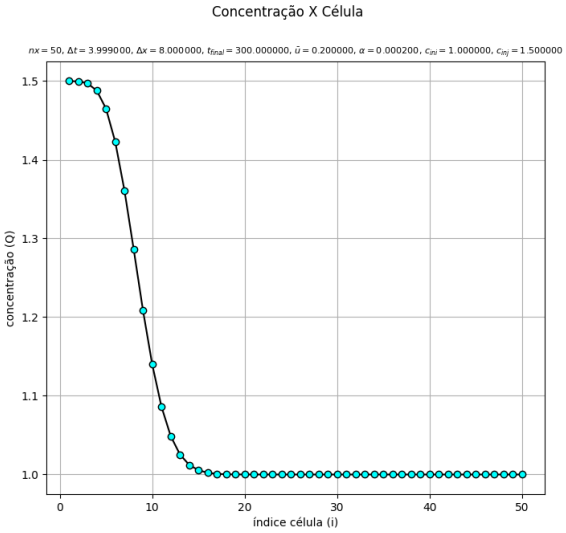

Figura 3.4: $L_x = 400\text{m}$

Ao se variar o comprimento do domínio, percebe-se que os efeitos adectivos e difusivos levam mais tempo para percorrer toda a sua extensão. Sendo assim, quanto maior o domínio, maior a quantidade de tempo necessária para este alcançar o equilíbrio.

3.2 Resultados para variações de nx

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

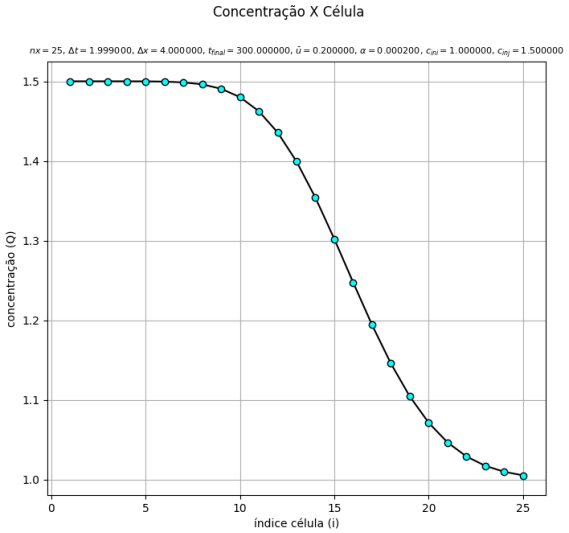

Figura 3.5: $nx = 25$

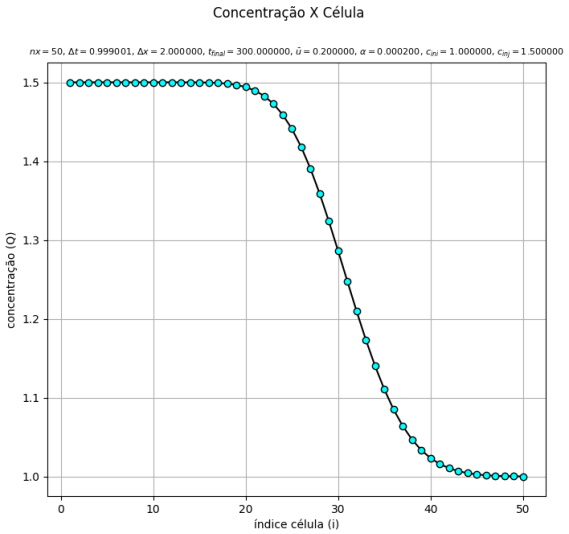

Figura 3.6: $nx = 50$

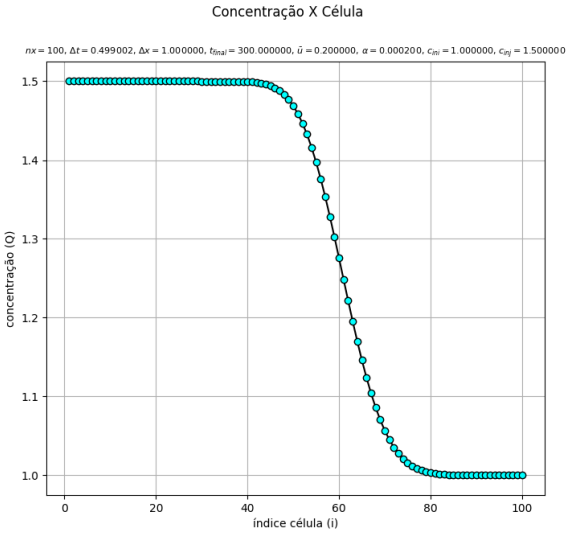

Figura 3.7: $nx = 100$

Pode-se observar que a variação do nx resulta em um aumento da resolução do gráfico. A curva se torna mais detalhada, com mais células descrevendo a concentração naquela posição do domínio.

3.3 Resultados para variações de $\bar{u}$

Com a variação de $\bar{u}$, obtiveram-se os seguintes resultados:

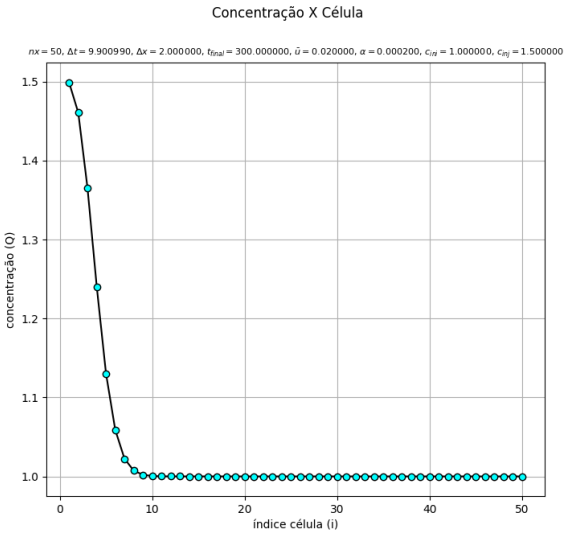

Figura 3.8: $\bar{u} = 2 \times 10^{-2}\text{m/s}$

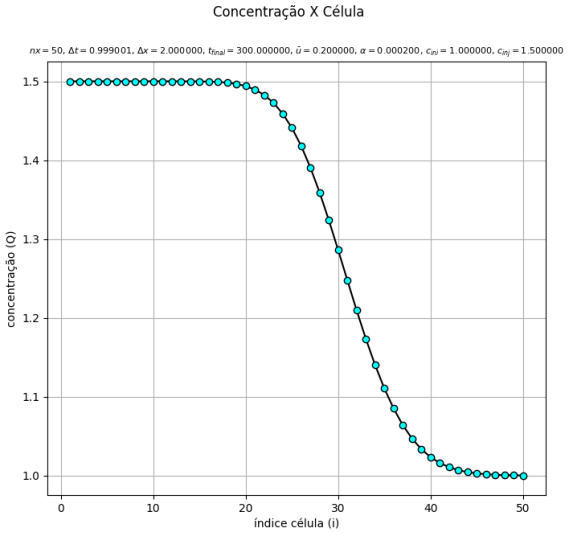

Figura 3.9: $\bar{u} = 2 \times 10^{-1}\text{m/s}$

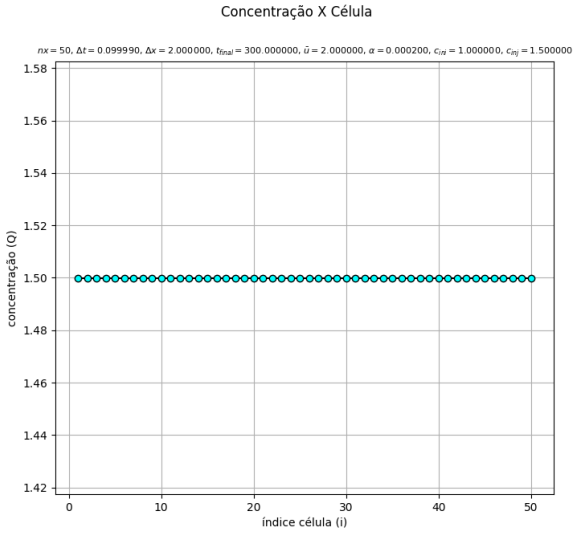

Figura 3.10: $\bar{u} = 2 \times 10^{0}\text{m/s}$

Pode-se observar que o sistema é bastante sensível em relação ao termo advectivo. Uma mudança na ordem de grandeza da velocidade faz com que este entre em equilíbrio em uma quantidade de tempo relativamente pequena.

3.4 Resultados para variações de α

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

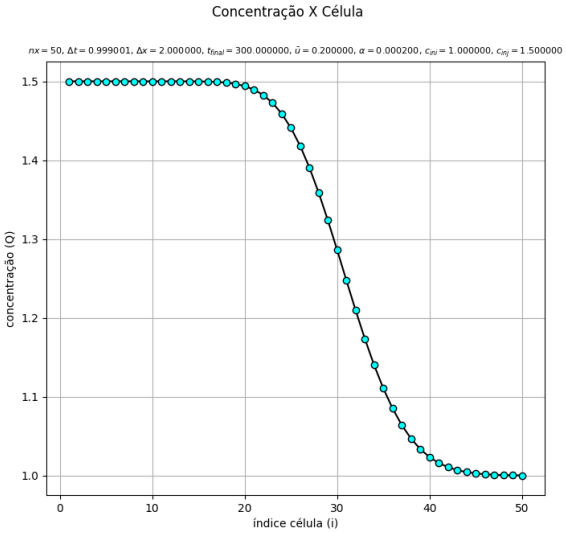

Figura 3.11: $\alpha = 2.0 \times 10^{-4}$

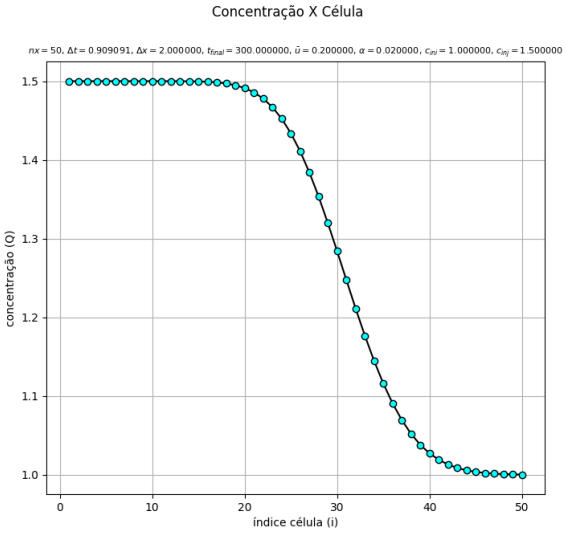

Figura 3.12: $\alpha = 2.0 \times 10^{-2}$

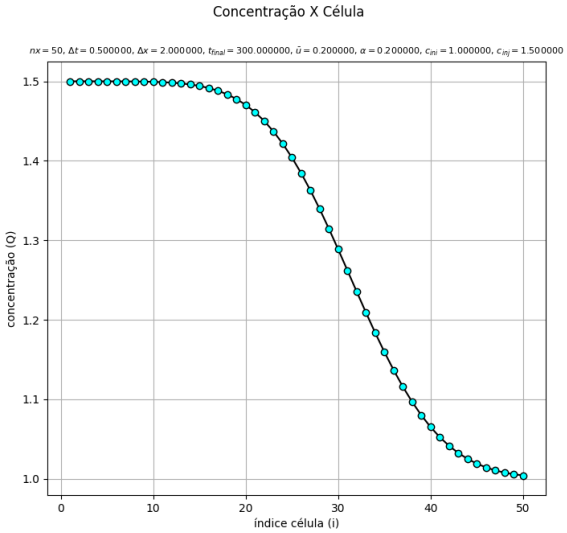

Figura 3.13: $\alpha = 2.0 \times 10^{-1}$

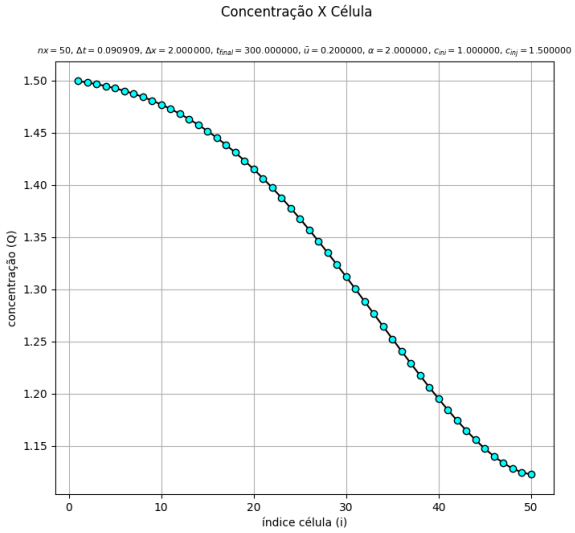

Figura 3.14: $\alpha = 2.0 \times 10^{0}$

Nota-se que a presença do efeito difusivo é quase desprezível a concentrações na ordem de 10^{-4}. Para $\alpha \geq 1$, porém, o efeito difusivo passa a ter uma presença significativa sobre o sistema, devendo assim ser levado em conta na modelagem de um problema real de engenharia.

3.5 Resultados para $\bar{u} = 0$ e variações de α

No caso especial onde há uma ausência do termo advectivo, obtiveram-se os seguintes resultados:

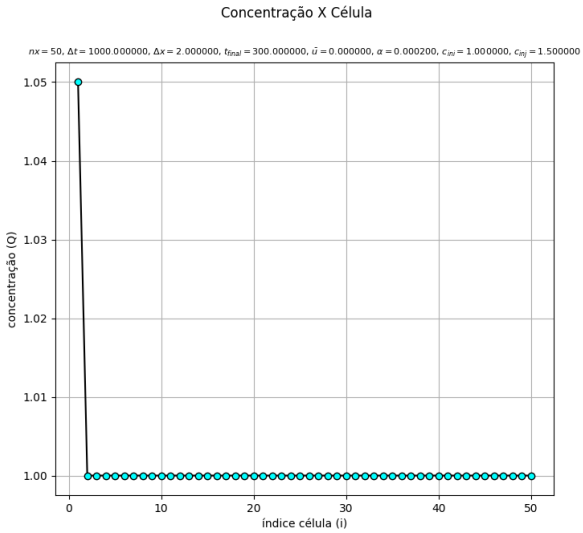

Figura 3.15: $\bar{u} = 0$ & $\alpha = 2.0 \times 10^{-4}$

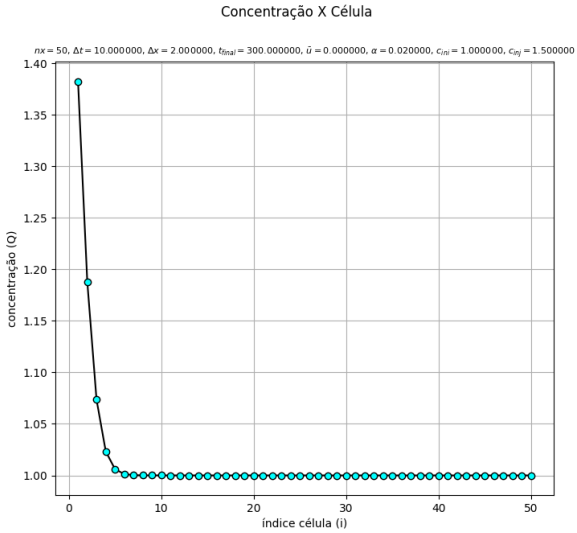

Figura 3.16: $\bar{u} = 0$ & $\alpha = 2.0 \times 10^{-2}$

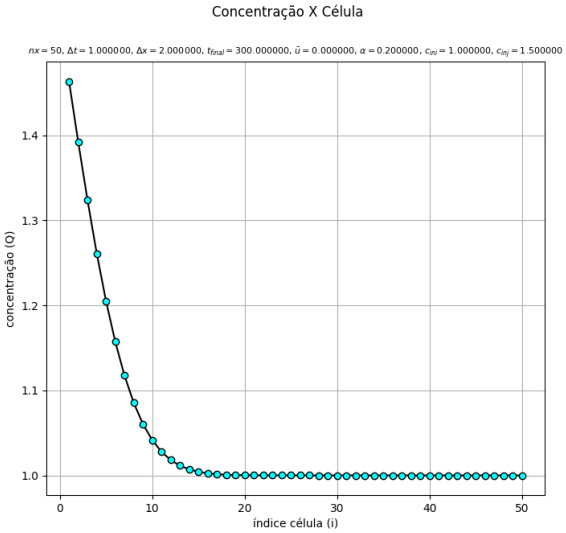

Figura 3.17: $\bar{u} = 0$ & $\alpha = 2.0 \times 10^{-1}$

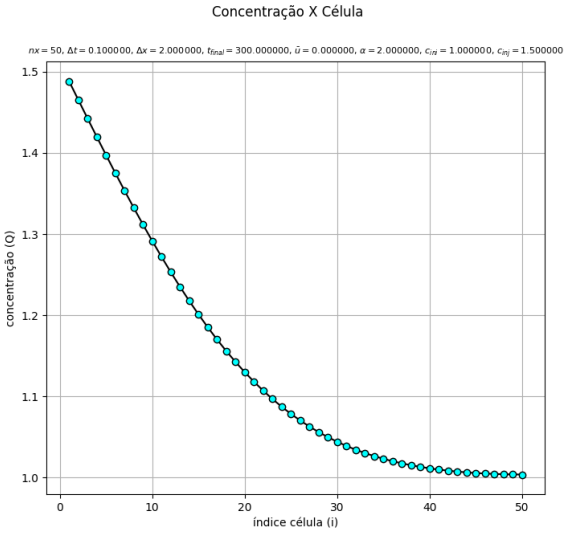

Figura 3.18: $\bar{u} = 0$ & $\alpha = 2.0 \times 10^{0}$

Com a ausência de um termo advectivo, o sistema leva bastante tempo para alcançar a estabilidade, dado valores de $\alpha \leq 10^{-2}$, pois a influência de um dado volume sobre os seus vizinhos é bastante pequeno. Com valores maiores, porém, os efeitos difusivos podem ser claramente apreciados, e a diferença de concentrações se propaga pelo sistema rapidamente.

3.6 Resultados para variações de $c_{\mathbf{ini}}$

Com a variação de c_{ini}, obtiveram-se os seguintes resultados:

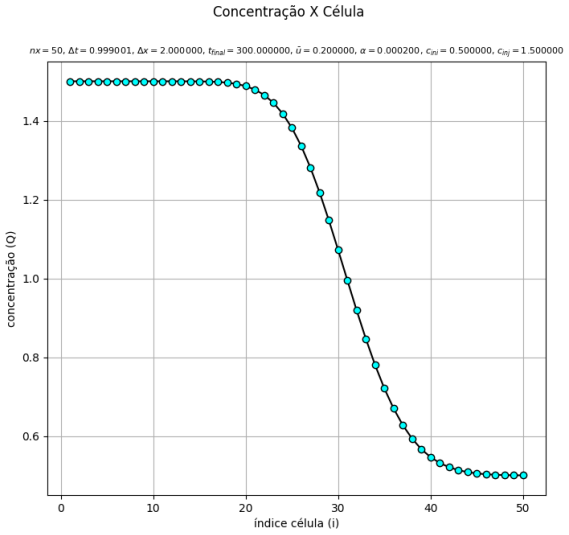

Figura 3.19: $c_{\text{ini}} = 0.5\text{mol/m}^{\mathbf{3}}$

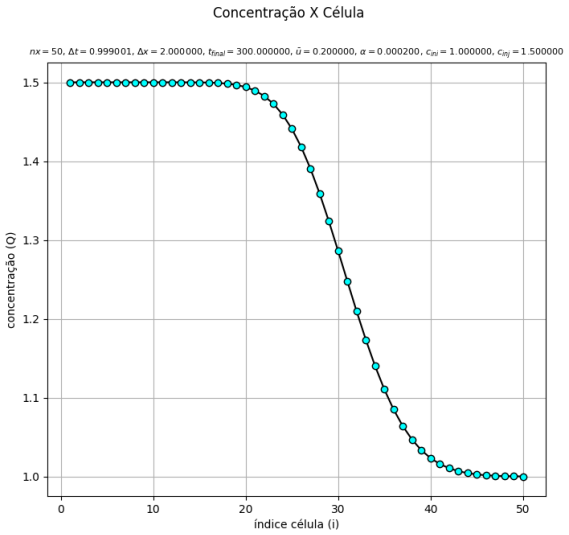

Figura 3.20: $c_{\text{ini}} = 1.0\text{mol/m}^{\mathbf{3}}$

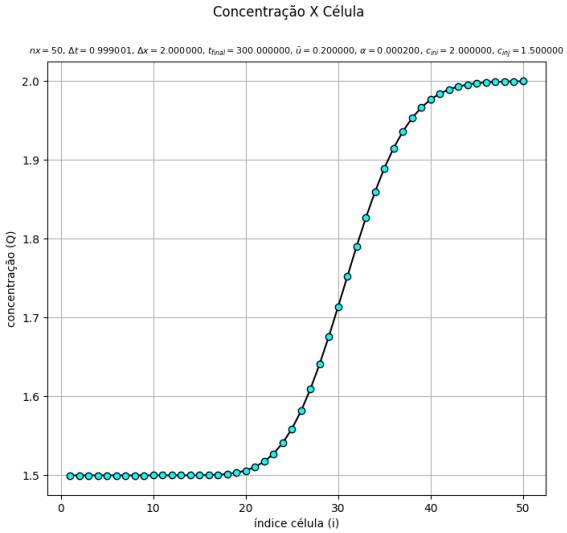

Figura 3.21: $c_{\text{ini}} = 2.0\text{mol/m}^3$

Para uma variação da concentração inicial, nota-se que os volumes mais próximos do contorno direito mantém esse valor para um tempo de simulação no qual os efeitos advectivos e difusivos não foram suficientes para influenciá-los. Sendo assim, para uma concentração inicial c_{ini} maior que a concentração de injeção c_{inj} o gráfico acaba se invertendo.

3.7 Resultados para variações de c_{inj}

Com a variação de c_{inj}, obtiveram-se os seguintes resultados:

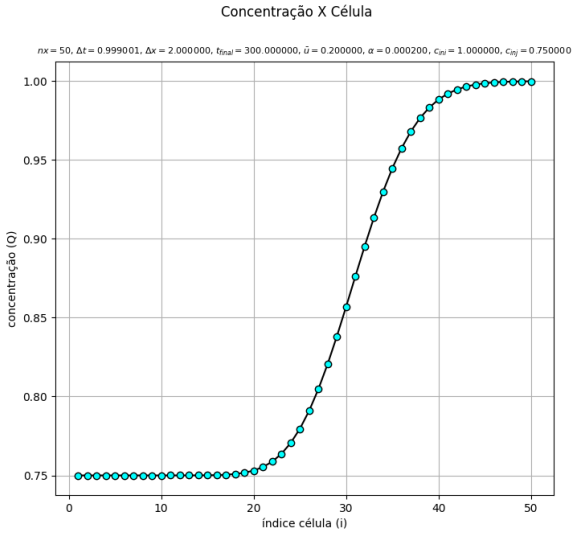

Figura 3.22: $c_{\text{inj}} = 0.75\text{mol/m}^3$

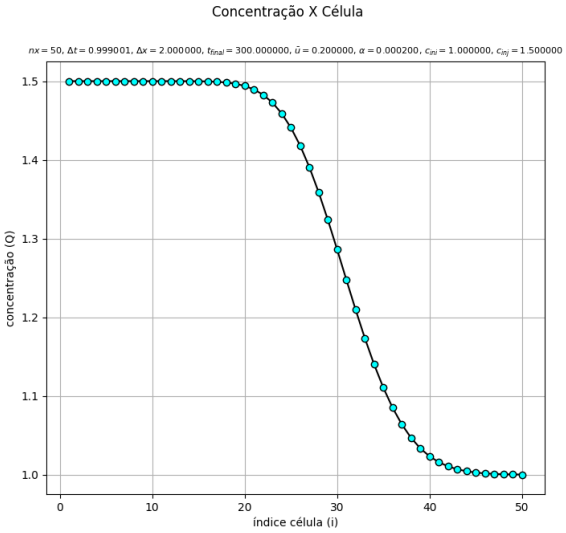

Figura 3.23: $c_{\text{inj}} = 1.5\text{mol/m}^3$

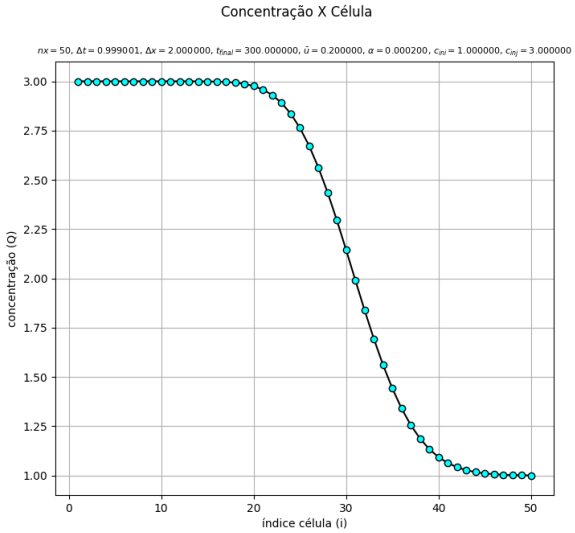

Figura 3.24: $c_{\text{inj}} = 3.0\text{mol/m}^3$

Para uma variação da concentração de injeção, tem-se a situação oposta da seção anterior: volumes próximos ao contorno esquerdo sofrem influência quase que imediata da concentração de injeção, enquanto volumes distantes levam mais tempo para serem influenciados por seus vizinhos.

3.8 Resultados para variações de t_{final}

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

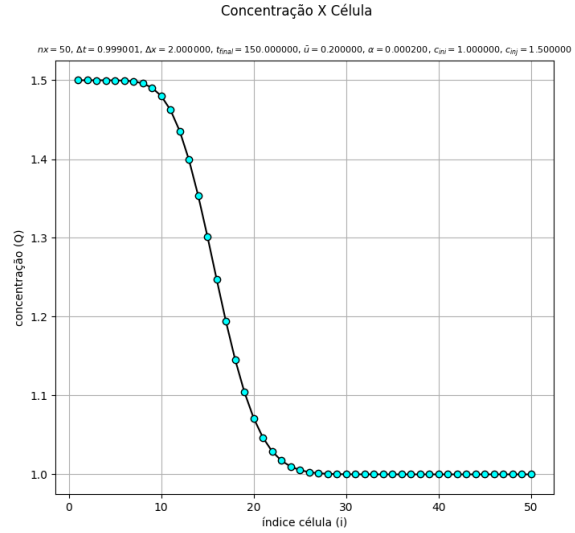

Figura 3.25: $t_{\text{final}} = 150\text{s}$

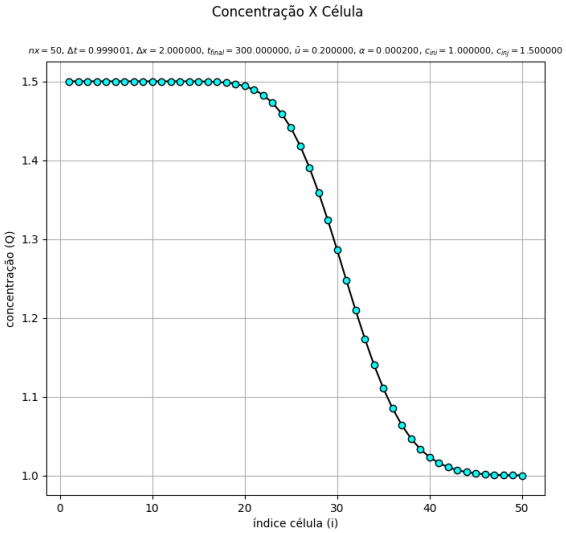

Figura 3.26: $t_{\text{final}} = 300\text{s}$

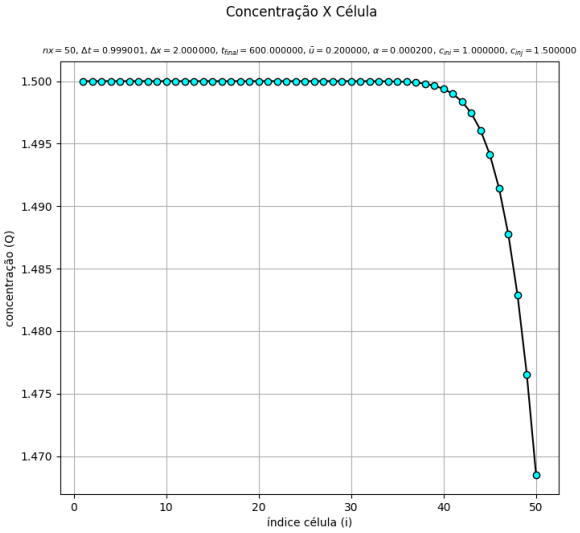

Figura 3.27: $t_{\text{final}} = 600\text{s}$

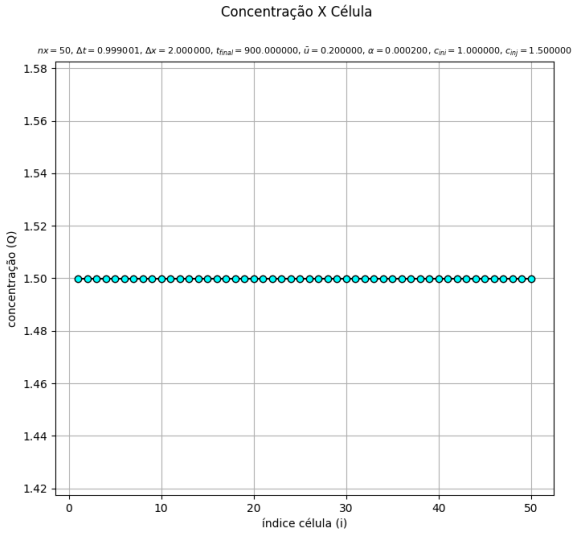

Figura 3.28: $t_{\text{final}} = 900\text{s}$

Ao variar o tempo de simulação, nota-se que o sistema tende ao equilíbrio dado um tempo longo o suficiente para os efeitos difusivos e advectivos influenciarem todos os volumes de seu domínio.

4. Conclusão

Equações diferenciais modelam quase tudo a nossa volta. A imagem introdutória, do despejo de esgoto, é exemplo disso. Ser capaz de aliar a análise matemática ao poder de processamento de um computador desbloqueia inúmeras possibilidades, principalmente no campo de engenharia.

Muitas vezes, equações diferenciais parciais não possuem solução analítica. Apesar de isto parecer um obstáculo intransponível, é possível resolvê-las usando alguns métodos alternativos.

Com o auxílio do Método dos Volumes Finitos, é viável resolver EDPs muito difíceis desde que se sua forma discretizada respeite as regras de consistência, convergência e estabilidade. Assim será possível analisar seu comportamento ao longo do tempo, e determinar o seu significado físico.

Neste trabalho foi possível perceber que cada parâmetro tem uma influência sobre o comportamento da simulação da EDP. Em alguns casos, pequenas alterações são insignificantes e, em outros, extremamente notáveis. O conhecimento do comportamento destes parâmetros aplicado a ferramenta computacional agiliza o desenvolvimento de projetos de engenharia, e viabiliza alguns que seriam impraticáveis de serem resolvidos manualmente.

5. Código Computacional

5.0.1 Código Principal

Arquivo `main.h`

```
/*****************************************************************************
 *        Métodos Numéricos para Equações Diferenciais II -- Trabalho 1       *
 *                           Ariel Nogueira Kovaljski                          *
 *****************************************************************************/

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

#define Lx      100.0       /* Lx: comprimento do domínio (em m) */
#define nx      50          /* nx: número de células */
#define Delta_x (Lx/nx)     /* Delta_x: largura de cada célula (em m) */
#define u_bar   0.2         /* u_bar: velocidade de escoamento (em m/s) */
#define alpha   2.0e-4      /* alpha: coeficiente de difusão */
#define c_ini   1.0         /* concentração inicial nos volumes da malha */
#define c_inj   1.5         /* concentração de injeção nos vol. da malha */
#define t_final 300.0       /* tempo final da simulação (em segundos) */
                            /* Delta_t: passo de tempo (em segundos) */
#define Delta_t (0.1 * (1/( (2*alpha)/(Delta_x*Delta_x) + u_bar/Delta_x ) ))

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

void listParameters();
void initializeArray(double arr[], int len, double value);
void calculateQ(double old_arr[], double new_arr[]);
void printAndSaveResults(double arr[], int len);
```

Arquivo `main.c`

```
/*****************************************************************************
 *        Métodos Numéricos para Equações Diferenciais II -- Trabalho 1       *
 *                           Ariel Nogueira Kovaljski                          *
 *****************************************************************************/

#include <stdio.h>
#include <stdlib.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("\nMNED II - Trabalho 1\n====================");
    puts("por Ariel Nogueira Kovaljski\n");
```

```
    listParameters();

    /* inicializa os arrays Q para uma concentração c_ini inicial */
    initializeArray(Q_old, nx, c_ini);
    initializeArray(Q_new, nx, c_ini);

    calculateQ(Q_old, Q_new);
    printAndSaveResults(Q_new, nx);

    return 0;
}

void listParameters()
{
    puts("Parametros\n----------");
    puts("Constantes da equacao:");
    printf("Delta_t = %f, Delta_x = %f, u_bar = %3.2e, alpha = %3.2e, "
           "c_inj = %3.2e\n\n", Delta_t, Delta_t, u_bar, alpha, c_inj);
    puts("Constantes da simulacao:");
    printf("nx = %d, t_final = %f, c_ini = %3.2e\n\n", nx, t_final, c_ini);
}

/* Inicializa um array para um valor de entrada */
void initializeArray(double arr[], int len, double value)
{
    int i;
    for (i = 0; i < len; ++i) {
        arr[i] = value;
    }
}

/* Calcula as concentrações na malha ao longo do tempo */
void calculateQ(double old[], double new[])
{
    int x;
    int progress = 0, progress_count = 0;
    int progress_incr = (t_final/Delta_t) * 5 / 100;
    double t = 0;

    do {

        /*************************************************************
        *
        *            |==@==|==@==|==@==|==@==|==@==|==@==|==@==|
        *               ^
        *            Para o volume da fronteira esquerda
        *        o índice 0 refere-se ao volume nº 1 da malha
        */
        new[0] = old[0] - Delta_t/Delta_x * (
                    u_bar * (2*old[0] - 2*c_inj)
                 - alpha * (old[1] - 3*old[0] + 2*c_inj) / Delta_x
                );

        /*************************************************************
        *
        *            |==@==|==@==|==@==|==@==|==@==|==@==|==@==|
        *                     ^      ^      ^      ^
        *             Para os volumes do centro da malha
        */
        for (x = 1; x < nx - 1; ++x) {
            new[x] = old[x] - Delta_t/Delta_x * (
```

```
                      u_bar * (old[x] - old[x-1])
                    - alpha * (old[x+1] - 2*old[x] + old[x-1]) / Delta_x
                   );
        }

        /************************************************************
        *
        *            |==@==|==@==|==@==|==@==|==@==|==@==|==@==|
        *                                                    ^
        *             Para o volume da fronteira direita
        *            x possui valor de nx - 1 nesse ponto
        */
        new[x] = old[x] - Delta_t/Delta_x * (
                    u_bar * (old[x] - old[x-1])
                  - alpha * (old[x-1] - old[x]) / Delta_x
                 );

        /* 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 a próxima
        iteração */
        for (x = 0; x < nx; ++x) {
            old[x] = new[x];
        }

        /* 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 i;
    FILE *results_file;     /* Ponteiro para o arquivo de resultados */

    /* 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 */
    if (   (results_file = fopen("./results/results.txt", "w")   ) == NULL
        && (results_file = fopen("./../results/results.txt", "w")) == NULL) {
        fputs("[ERR] Houve um erro ao escrever o arquivo \"results.txt\"! "
              "Os resultados nao foram salvos.\n", stderr);
        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"
            "alpha=%f\n"
            "c_ini=%f\n"
            "c_inj=%f\n",
            nx, Delta_t, Delta_x, t_final, u_bar, alpha, c_ini, c_inj);
    fputs("********************\n", results_file);

    for (i = 0; i < len; ++i) {
        fprintf(results_file, "%d,%f\n", i + 1, arr[i]);
    }

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

    puts("[INFO] Os resultados foram salvos no arquivo \"results.txt\" "
    "no diretorio \"results/\".");
}
```

5.0.2 Código do Gráfico (`plot_graph.py`)

```
################################################################################
#         Métodos Numéricos para Equações Diferenciais II -- Trabalho 1        #
#                             Ariel Nogueira Kovaljski                         #
################################################################################

import matplotlib.pyplot as plt

x = []
y = []
parameters = {}

# Abre arquivo para leitura
f = open('./results/results.txt', 'r')

for line_number, line in enumerate(f):
    if line_number < 8:
        # Adiciona parâmetros da simulação em um dicionário
        parameters[line.split('=')[0]] = (line.split('=')[1]).split('\n')[0]
    elif line_number == 8:
        # Pula linha separadora
        pass
    else:
        # Separa valores na lista `x` e na lista `y`
        x.append( int(line.split(',')[0]) )
        y.append( float( (line.split(',')[1]).split('\n')[0] ) )

# Configura e exibe o gráfico
fig,ax = plt.subplots()
fig.set_size_inches(8, 7)   # Size of the window (1in = 100px)
ax.grid(True)

plt.suptitle("Concentração X Célula")
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"$\bar{{u}} = {parameters['u_bar']}$, "
          rf"$\alpha = {parameters['alpha']}$, "
          rf"$c_{{ini}} = {parameters['c_ini']}$, "
```

```
         rf"$c_{{inj}} = {parameters['c_inj']}$", fontsize=8)
# Rótulo X -- descomentar depois
# plt.xlabel("célula (i)")

# Rótulo X -- resolvendo problema da célula
plt.xlabel("índice célula (i)")
plt.ylabel("concentração (Q)")
plt.plot(x,y,'ko-', markerfacecolor='cyan', markeredgecolor='k')
plt.show()
```