Trabalho 2 de Métodos Numéricos para Equações Diferenciais II

Ariel Nogueira Kovaljski

Nova Friburgo, 1 de dezembro 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 três métodos numéricos — Forward Time-Backward Space (FTBS), Superbee e Van Albada — 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 três métodos numéricos baseados no método dos volumes finitos — Forward Time-Backward Space (FTBS), Superbee, Van Albada — 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 três métodos utilizados neste trabalho.

Como base para todos os métodos, será utilizada a equação 2.4, modificada ligeiramente, conforme descrito na proposta do trabalho. Esta versão modificada pode ser vista abaixo,

$$Q_i^{n+1} = Q_i^n - C\left(Q_i^n - Q_{i-1}^n\right) - \frac{1}{2}C(1-C)\left[\psi\left(\theta_{i+1/2}^n\right)\left(Q_{i+1/2}^n - Q_i^n\right) - \psi\left(\theta_{i-1/2}^n\right)\left(Q_i^n - Q_{i-1}^n\right)\right] \tag{3.6}$$

onde C é o número de Courant, $\theta_{i\pm1/2}^n$ são funções adicionais, definidas como,

$$\theta_{i-1/2}^n = \frac{Q_{i-1}^n - Q_{i-2}^n}{Q_i^n - Q_{i-1}^n} \qquad \text{e} \qquad \theta_{i+1/2}^n = \frac{Q_i^n - Q_{i-1}^n}{Q_{i+1}^n - Q_i^n}$$

e ψ é um dos métodos numéricos a serem adotados.

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 forma como função ψ é:

$$\psi(\theta) = 0 \tag{3.7}$$

3.2.2 Métodos TVD

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 TVD introduzem um termo corretivo, de maneira a minimizar a influência da difusão numérica sobre o resultado final. Os métodos TVD escolhidos para a resolução da equação da advecção são o Superbee e Van Albada.

Superbee

Para o Superbee, sua forma como função ψ é:

$$\psi(\theta) = \max(0, \min(1, 2\theta), \min(2, \theta)) \tag{3.8}$$

Van Albada

Para Van Albada, sua forma como função ψ é:

$$\psi(\theta) = \frac{\theta^2 + \theta}{\theta^2 + 1} \tag{3.9}$$

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,

$$\Delta t \leq C \left(\frac{1}{\frac{2\alpha}{\Delta x^2} + \frac{\bar{u}}{\Delta x}} \right) \tag{3.10}$$

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

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

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

$$\therefore \ \Delta t \leq C \frac{\Delta x}{\bar{u}} \tag{3.11}$$

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 0,8 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 (upwind)
calculateQ(Q_old, Q_new, upwind);
```

```
// Imprime na tela e salva os resultados no arquivo de texto
printAndSaveResults(Q_new, NX, UPWIND);

// Método Superbee
initializeArray(Q_old, NX, A, B, C, D, E)
initializeArray(Q_new, NX, A, B, C, D, E)
calculateQ(Q_old, Q_new, superbee);
printAndSaveResults(Q_new, NX, SUPERBEE);

// Método Van Albada
initializeArray(Q_old, NX, A, B, C, D, E)
initializeArray(Q_new, NX, A, B, C, D, E)
calculateQ(Q_old, Q_new, vanAlbada);
printAndSaveResults(Q_new, NX, VAN_ALBADA);
```

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
int i;
double t = 0;

do {
    // Cálculo do volume da fronteira esquerda
    i = 0;
    new[i] = old[i] - COURANT/2 * (1-COURANT) * (
                psi(thetaPlusHalf(old, i)) * (old[i+1] - old[i])
             );

    // Cálculo dos volumes do centro da malha
    for (i = 1; i < NX - 1; ++i) {
        new[i] = old[i] - COURANT * (
                     old[i] - old[i-1]
                 ) - COURANT/2 * (1-COURANT) * (
                       psi(thetaPlusHalf(old, i))  * (old[i+1] - old[i]  )
                     - psi(thetaMinusHalf(old, i)) * (old[i]   - old[i-1])
                 );
    }

    // Cálculo do volume da fronteira direita
    new[i] = old[i] - COURANT * (
                 old[i] - old[i-1]
             ) - COURANT/2 * (1-COURANT) * (
                 - psi(thetaMinusHalf(old, i)) * (old[i] - old[i-1])
             );
```

```
    // 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 passo de tempo
} while ( (t += DELTA_T) <= T_FINAL);
```

Cada função `thetaMinusHalf` / `thetaPlusHalf` possui a seguinte estrutura:

```
int a, b, c;

// Aplica condição de contorno / volume fantasma
if (method == "thetaMinusHalf") {
    a = (i - 2) < 0 ? 0 : (i - 2);
    b = (i - 1) < 0 ? 0 : (i - 1);
    c = i;
} else if (method == "thetaPlusHalf") {
    a = (i - 1) < 0 ? 0 : (i - 1);
    b = i;
    c = i + 1;
}

// Workaround para divisão por zero
if (arr[c] - arr[b] == 0) {
    return (arr[b] - arr[a]) / 1e-10;
    /* return 0; */
}

return (arr[b] - arr[a]) / (arr[c] - arr[b]);
```

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 $e^{-A(x-b)^2} + s(x)$.

Ao ser chamada, a função `calculateQ` recebe um ponteiro para a função `psi` correspondente. Isto permite o reuso de código, pois os passos do cálculo de Q são idênticos para todas as funções.

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[]`. Para o cálculo de cada função `psi`, antes é calculado o valor de $\theta^n_{i\pm1/2}$ por suas respectivas funções `thetaMinusHalf` e `thetaPlusHalf`.

O cálculo de $\theta^n_{i\pm1/2}$ aplica as condições de contorno, evitando assim o acesso de índices fora do domínio do vetor. Para alguns casos onde o denominador resultaria em zero, é aplicado um *workaround*, isto é, uma medida mitigadora, onde o denominador é definido como um número pequeno $\sim 10^{-10}$.

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:

- $L_x = 20\text{m}$ (comprimento do domínio)
- $nx = 400$ (número de células)
- $\bar{u} = 1\text{m/s}$ (velocidade de escoamento)
- $t_{\text{final}} = 2\text{s}$ (tempo final de simulação)
- $\Delta x = \frac{L_x}{nx} = 0,05\text{m}$ (passo no espaço)
- $\Delta t = 0,8\left(\frac{\Delta_x}{\bar{u}}\right) = 0,04\text{s}$ (passo de tempo)
- $A = 100$
- $B = 1,5$
- $C = 4,0$
- $D = 6,0$
- $E = 2,0$

4.1 Resultados para variações de nx

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

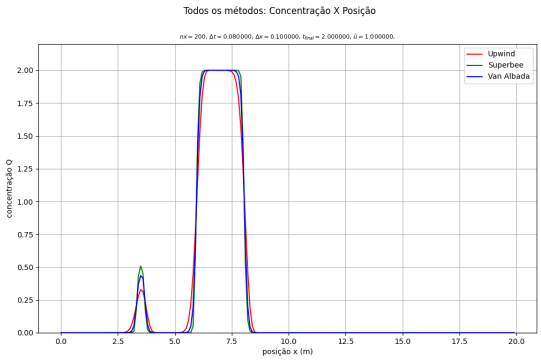

Figura 4.1: FTBS, Superbee, Van Albada: $nx = 200$

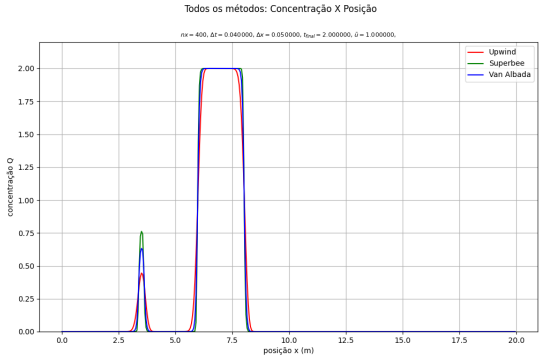

Figura 4.2: FTBS, Superbee, Van Albada: $nx = 400$

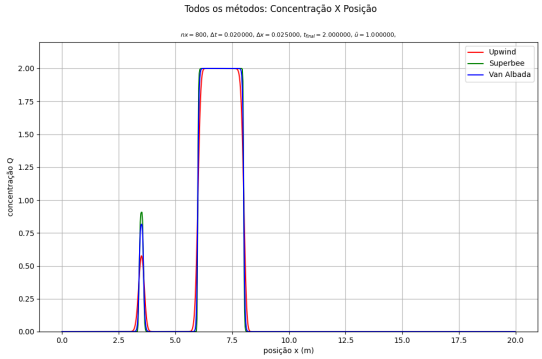

Figura 4.3: FTBS, Superbee, Van Albada: $nx = 800$

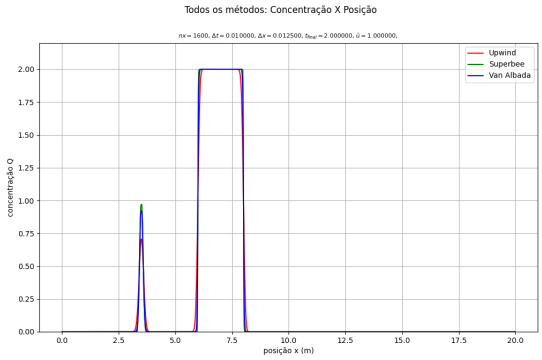

Figura 4.4: FTBS, Superbee, Van Albada: $nx = 1600$

Nota-se que o refinamento da malha resulta em dois fenômenos: a primeira onda se torna mais "pontiaguda" e a segunda mais "quadrada"; tais características não podiam ser representadas em uma malha de baixa resolução. Sendo

assim, determina-se que o aumento no número de nós torna a solução discreta mais próxima da analítica (solução real).

4.2 Resultados para variações de t_{final}

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

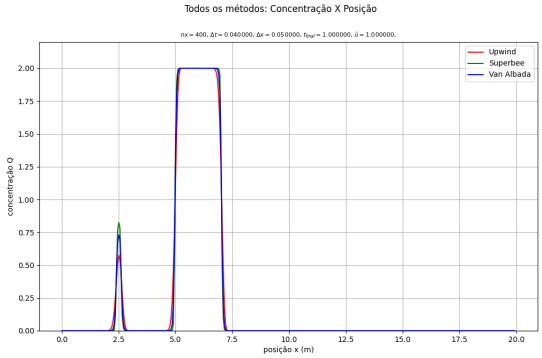

Figura 4.5: FTBS, Superbee, Van Albada: $t_{\text{final}} = 1$s

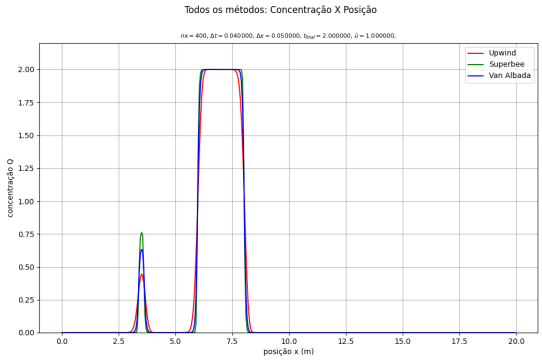

Figura 4.6: FTBS, Superbee, Van Albada: $t_{\text{final}} = 2$s

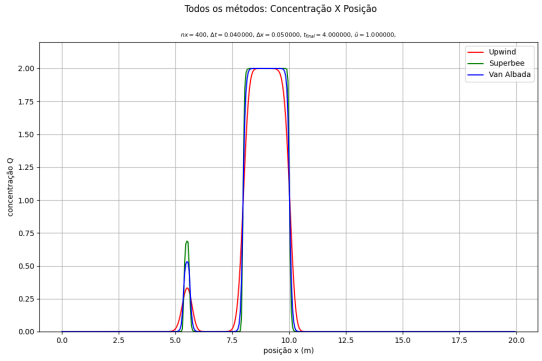

Figura 4.7: FTBS, Superbee, Van Albada: $t_{\text{final}} = 4\text{s}$

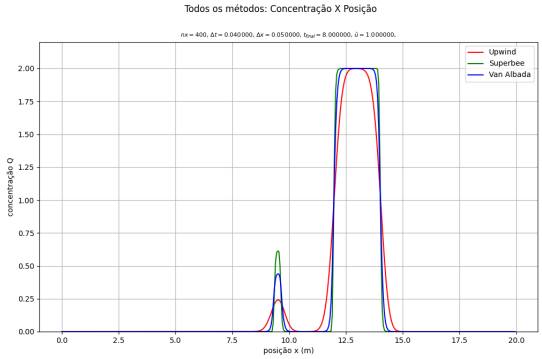

Figura 4.8: FTBS, Superbee, Van Albada: $t_{\text{final}} = 8\text{s}$

Nota-se que com o avanço no tempo, dentre todos os métodos, o Superbee é o que menos apresentou difusão numérica, mantendo-se assim mais próximo da solução analítica (real). Comparado com os métodos de alta resolução, os métodos TVD apresentam as oscilações espúrias, o que os torna mais eficazes para este tipo de problema.

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, possuem erro de truncamento na ordem de $\frac{\partial^2 c}{\partial x^2}$ o que introduz um grau mais elevado de difusão numérica. Métodos TVD, possuem as vantagens dos Métodos de Alta Resolução — baixa difusão numérica, devido ao erro de truncamento na ordem de $\frac{\partial^3 c}{\partial x^3}$ — sem os seus defeitos — oscilações espúrias.

Em relação aos refinamentos da malha, nota-se que todos os métodos se comportam de maneira similar. Uma diminuição no número de nós nx, prejudica a acurácia da solução, pois a baixa resolução esconde as variações abruptas nas curvas. O aumento de nx apresenta um resultado mais fiel ao real, porém demanda mais poder computacional para o cálculo da malha.

Em relação às mudanças em t_{final}, observa-se que 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 $\bar{u} > 0$. Erros causados pela difusão numérica mostram que métodos de primeira ordem como o FTBS não são os mais adequados para esta análise. Dentre os métodos TVD, o que apresentou melhor resultado foi o Superbee, pois manteve-se mais próximo da curva analítica comparado com o Van Albada, mesmo para valores de t_{final} elevados.

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 são de simples implementação, mas, devido a elevada difusão numérica, divergem do resultado analítco. Métodos TVD como Superbee e Van Albada possuem melhor acurácia, tendo pequena difusão numérica ao longo do tempo.

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 3      *
 *                          Ariel Nogueira Kovaljski                         *
 *****************************************************************************/

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

#define LX      (20.0)              /* comprimento do domínio (em m)          */
#define NX      (400)               /* número de células                      */
#define DELTA_X (LX/NX)             /* largura de cada célula (em m)          */
#define U_BAR   (1.0)               /* velocidade de escoamento (em m/s)      */
#define T_FINAL (2.0)               /* tempo final da simulação (em segundos) */
#define COURANT (0.8)               /* 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)                     /*                                */

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

#define MIN(a,b) (((a) < (b)) ? (a) : (b))
#define MAX(a,b) (((a) > (b)) ? (a) : (b))
#define MAX3(a,b,c) (MAX(MAX((a),(b)), (c)))

/* Métodos utilizados para o cálculo de Q neste trabalho */
enum methods {UPWIND, SUPERBEE, VAN_ALBADA};

void listParameters();
void initializeArray(double arr[], int len, double a, double b, double c,
                     double d, double e);
double thetaPlusHalf(double arr[], int i);
double thetaMinusHalf(double arr[], int i);
void calculateQ(double old[], double new[], double (*psi)(double theta));
double upwind(double theta);
double superbee(double theta);
double vanAlbada(double theta);
void printAndSaveResults(double arr[], int len, int method);
```

Arquivo `main.c`

```
/*****************************************************************************
 *        Métodos Numéricos para Equações Diferenciais II -- Trabalho 3      *
 *                           Ariel Nogueira Kovaljski                        *
 *****************************************************************************/

#include "main.h"

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.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 3\n"
         "====================\n"
         "por Ariel Nogueira Kovaljski\n");

    listParameters();

    /* Cálculo de Q via método Upwind */
    puts("Calculando Q via metodo Upwind...");
    initializeArray(Q_old, NX, A, B, C, D, E);
    initializeArray(Q_new, NX, A, B, C, D, E);
    calculateQ(Q_old, Q_new, upwind);
    printAndSaveResults(Q_new, NX, UPWIND);

    /* Cálculo de Q via método Superbee */
    puts("Calculando Q via metodo Superbee...");
    initializeArray(Q_old, NX, A, B, C, D, E);
    initializeArray(Q_new, NX, A, B, C, D, E);
    calculateQ(Q_old, Q_new, superbee);
    printAndSaveResults(Q_new, NX, SUPERBEE);

    /* Cálculo de Q via método Van Albada */
    puts("Calculando Q via metodo Van Albada...");
    initializeArray(Q_old, NX, A, B, C, D, E);
    initializeArray(Q_new, NX, A, B, C, D, E);
    calculateQ(Q_old, Q_new, vanAlbada);
    printAndSaveResults(Q_new, NX, VAN_ALBADA);

    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("LX = %f, NX = %d, T_FINAL = %f\n\n", LX, 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;
    }
}

double thetaPlusHalf(double arr[], int i)
{
    int a, b, c;

    /* Aplica condição de contorno / volume fantasma */
    a = (i - 1) < 0 ? 0 : (i - 1);
    b = i;
    c = i + 1;

    /* Workaround para divisão por zero */
    if (arr[c] - arr[b] == 0) {
        return (arr[b] - arr[a]) / 1e-10;
    }

    return (arr[b] - arr[a]) / (arr[c] - arr[b]);
}

double thetaMinusHalf(double arr[], int i)
{
    int a, b, c;

    /* Aplica condição de contorno / volume fantasma */
    a = (i - 2) < 0 ? 0 : (i - 2);
    b = (i - 1) < 0 ? 0 : (i - 1);
    c = i;

    /* Workaround para divisão por zero */
    if (arr[c] - arr[b] == 0) {
        return (arr[b] - arr[a]) / 1e-10;
    }

    return (arr[b] - arr[a]) / (arr[c] - arr[b]);
}

/* calcula Q, recebendo como ponteiro a função ‘psi‘ */
void calculateQ( double old[], double new[], double (*psi)(double
theta) )
{
    int i;
    double t = 0;

    do {
        /*
```

```
         *                    Fronteira esquerda
         *               |=@=|=@=|=@=|=@=|=@=|=@=|=@=|=@=|
         *                 ^
         */
        i = 0;
        new[i] = old[i] - COURANT/2 * (1-COURANT) * (
                    psi(thetaPlusHalf(old, i)) * (old[i+1] - old[i])
                );

        /*
         *                           Centro
         *               |=@=|=@=|=@=|=@=|=@=|=@=|=@=|=@=|
         *                     ^   ^   ^   ^   ^
         */
        for (i = 1; i < NX - 1; ++i) {
            new[i] = old[i] - COURANT * (
                        old[i] - old[i-1]
                    ) - COURANT/2 * (1-COURANT) * (
                          psi(thetaPlusHalf(old, i))  * (old[i+1] - old[i]  )
                        - psi(thetaMinusHalf(old, i)) * (old[i]   - old[i-1])
                    );
        }

        /*
         *                    Fronteira direita
         *               |=@=|=@=|=@=|=@=|=@=|=@=|=@=|=@=|
         *                                           ^
         */
        new[i] = old[i] - COURANT * (
                    old[i] - old[i-1]
                ) - COURANT/2 * (1-COURANT) * (
                    - psi(thetaMinusHalf(old, i)) * (old[i] - old[i-1])
                );

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

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

/* Método Upwind */
double upwind(double theta)
{
    return (theta = 0);
}

/* Método Superbee */
double superbee(double theta)
{
    return MAX3(0, MIN(1, 2*theta), MIN(2, theta));
}

/* Método Van Albada */
double vanAlbada(double theta)
{
    return ( (theta*theta) + theta ) / ( (theta*theta) + 1 );
}

/* 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[100], file1[100];
    FILE *results_file;     /* Ponteiro para o arquivo de resultados */

    /* Prepara nome do arquivo de saída */
    switch (method) {
        case UPWIND:
            sprintf(filename, "%s%d%s", "results", method, ".txt");
            break;
        case SUPERBEE:
            sprintf(filename, "%s%d%s", "results", method, ".txt");
            break;
        case VAN_ALBADA:
            sprintf(filename, "%s%d%s", "results", method, ".txt");
            break;
    }

    /* Imprime os resultados no console */
    printf("Q[%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 */
    sprintf(file0, "%s%s", "./results/",    filename);
    sprintf(file1, "%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 3      #
#                            Ariel Nogueira Kovaljski                        #
##############################################################################

import matplotlib.pyplot as plt


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

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

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

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


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

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

    # Plota todos os gráficos na mesma janela
    plot_all_graphs()


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: 'Upwind', 1:'Superbee', 2:'Van Albada'}
    colors  = {0: 'red',    1: 'green',   2: 'blue'}
    fig,ax  = plt.subplots()
    fig.set_size_inches(12, 7)
    ax.grid(True)
    ax.set(ylim=(0, 2.2))

    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')
    plt.draw()

def plot_all_graphs():
    methods = {0: 'Upwind', 1:'Superbee', 2:'Van Albada'}
    colors  = {0: 'red',    1: 'green',   2: 'blue'}
    fig,ax  = plt.subplots()
    fig.set_size_inches(12, 7)
    ax.grid(True)
    ax.set(ylim=(0, 2.2))

    plt.suptitle(f"Todos os métodos: 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(x0, y0, f'{colors[0][0]}-', markerfacecolor=colors[0],
             label=methods[0])
    plt.plot(x1, y1, f'{colors[1][0]}-', markerfacecolor=colors[1],
             label=methods[1])
    plt.plot(x2, y2, f'{colors[2][0]}-', markerfacecolor=colors[2],
             label=methods[2])
    plt.legend(loc='upper right')
    plt.savefig(f"./results/fig/methods_t={parameters['t_final']}.png")
    plt.show()

if __name__ == "__main__":
    main()
```