

Trabalho 3 - Métodos Numéricos para Equações Diferenciais

Renato

Introdução

Este relatório apresenta a solução numérica da equação diferencial parcial:

$$\frac{\partial C}{\partial t} - \alpha \frac{\partial^2 C}{\partial x^2} + kC = 0, \quad 0 < x < L_x, \quad t > 0$$

com as seguintes condições de contorno e inicial:

$$C(x=0, t) = C_E, \quad \frac{\partial C}{\partial x}(x=L_x, t) = 0, \quad C(x, t=0) = 0$$

A equação modela fenômenos de difusão-reação, onde α representa a difusividade e k a taxa de reação. O objetivo é resolver numericamente a equação utilizando o método de diferenças finitas totalmente implícito, variando os parâmetros da malha espacial (n_x), o tempo de simulação (T_{final}) e as constantes físicas (α, k). Além disso, serão analisados o tempo de simulação para diferentes valores de n_x e calculado o erro relativo em relação a uma solução de referência.

Discretização

A discretização temporal utiliza uma aproximação atrasada (backward Euler), e a espacial utiliza diferenças centradas:

$$\frac{\partial C}{\partial t} \approx \frac{C_i^{n+1} - C_i^n}{\Delta t}, \quad \frac{\partial^2 C}{\partial x^2} \approx \frac{C_{i+1}^{n+1} - 2C_i^{n+1} + C_{i-1}^{n+1}}{\Delta x^2}$$

Substituindo na equação diferencial, temos:

$$\frac{C_i^{n+1} - C_i^n}{\Delta t} - \alpha \frac{C_{i+1}^{n+1} - 2C_i^{n+1} + C_{i-1}^{n+1}}{\Delta x^2} + kC_i^{n+1} = 0$$

Rearranjando:

$$-\frac{\alpha}{\Delta x^2} C_{i-1}^{n+1} + \left(\frac{1}{\Delta t} + \frac{2\alpha}{\Delta x^2} + k \right) C_i^{n+1} - \frac{\alpha}{\Delta x^2} C_{i+1}^{n+1} = \frac{C_i^n}{\Delta t}$$

As condições de contorno são incorporadas na matriz do sistema:

- Em $x = 0$: $C_0^{n+1} = C_E$
- Em $x = L_x$: $\frac{C_N^{n+1} - C_{N-1}^{n+1}}{\Delta x} = 0 \implies C_N^{n+1} = C_{N-1}^{n+1}$

Parâmetros e Testes

Os seguintes parâmetros foram utilizados:

- Constantes físicas:
 - $\alpha \in \{0.01, 0.1, 0.5\}$
 - $k \in \{0.02, 0.1, 0.5\}$
- Comprimento do domínio: $L_x = 1.0$
- Condição de contorno em $x = 0$: $C_E = 1.0$
- Condição inicial: $C(x, t = 0) = 0$
- Tempo final da simulação: $T_{final} = 1.0$
- Número de pontos no espaço: $n_x \in \{10, 50, 100, 500\}$
- Número de passos no tempo: $n_t = 1000$

Para avaliar o desempenho do método numérico, foram medidos os tempos de simulação para diferentes valores de n_x e calculados os erros relativos em relação a uma solução de referência ($n_x = 1000$).

Implementação em Python

A equação foi resolvida utilizando o método implícito, resultando em um sistema linear a cada passo de tempo. O código foi implementado em Python, utilizando bibliotecas como NumPy e Matplotlib.

Código

Abaixo está o código Python utilizado para resolver a equação:

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import time
4
5 # Parâmetros físicos
6 alpha_values = [0.01, 0.1, 0.5]
7 k_values = [0.02, 0.1, 0.5]
8 Lx = 1.0
9 CE = 1.0
10 T_final = 1.0
11
12 # Parâmetros numéricos
13 nx_values = [10, 50, 100, 500]
14 nt = 1000 # Número de passos no tempo
15 dt = T_final / nt
16
17 # Solução de referência para cálculo do erro
```

```

18 nx_ref = 1000
19 dx_ref = Lx / (nx_ref - 1)
20 x_ref = np.linspace(0, Lx, nx_ref)
21 C_ref = np.zeros(nx_ref)
22 C_ref[0] = CE
23
24 # Função para construir a matriz do sistema
25 def construct_matrix(alpha, k, dx, dt, nx):
26     A = np.zeros((nx, nx))
27     for i in range(1, nx - 1):
28         A[i, i - 1] = -alpha / dx**2
29         A[i, i] = 1 / dt + 2 * alpha / dx**2 + k
30         A[i, i + 1] = -alpha / dx**2
31     # Condição de contorno em x=0
32     A[0, 0] = 1
33     # Condição de Neumann em x=Lx
34     A[-1, -2] = -1 / dx
35     A[-1, -1] = 1 / dx
36     return A
37
38 # Simulação para a solução de referência
39 alpha_ref = 0.5
40 k_ref = 0.1
41 A_ref = construct_matrix(alpha_ref, k_ref, dx_ref, dt, nx_ref)
42 b_ref = np.zeros(nx_ref)
43 start_time = time.time()
44 for n in range(nt):
45     b_ref = C_ref / dt
46     b_ref[0] = CE
47     b_ref[-1] = 0 # Condição de Neumann
48     C_ref = np.linalg.solve(A_ref, b_ref)
49 simulation_time_ref = time.time() - start_time
50
51 # Loop sobre os parâmetros nx
52 simulation_times = []
53 errors = []
54
55 for nx in nx_values:
56     dx = Lx / (nx - 1)
57     x = np.linspace(0, Lx, nx)
58     C = np.zeros(nx)
59     C[0] = CE
60     A = construct_matrix(alpha_ref, k_ref, dx, dt, nx)
61     b = np.zeros(nx)
62     start_time = time.time()
63     for n in range(nt):
64         b = C / dt
65         b[0] = CE
66         b[-1] = 0 # Condição de Neumann
67         C = np.linalg.solve(A, b)
68     simulation_time = time.time() - start_time

```

```

69     simulation_times.append(simulation_time)
70     # Interpola o para calcular o erro relativo
71     C_interp = np.interp(x_ref, x, C)
72     error = np.linalg.norm(C_interp - C_ref, ord=2) / np.linalg.
        norm(C_ref, ord=2)
73     errors.append(error)
74     # Plotando os resultados
75     plt.plot(x, C, label=f'nx={nx}')
76
77 plt.plot(x_ref, C_ref, 'k--', label='Solu o de refer ncia')
78 plt.xlabel('Posi o x')
79 plt.ylabel('Concentra o C')
80 plt.title('Distribui o de C ap s $T_{final}$')
81 plt.legend()
82 plt.grid(True)
83 plt.show()

```

Descrição do Código

O código define os parâmetros físicos e numéricos, constrói a matriz do sistema linear para o método implícito e resolve o sistema para cada passo de tempo. A função `construct_matrix` monta a matriz A com base nos coeficientes resultantes da discretização.

As condições de contorno são implementadas diretamente na matriz:

- **Em $x = 0$:** Impõe-se $C = C_E$, definindo $A[0, 0] = 1$ e ajustando o vetor b em cada iteração.
- **Em $x = L_x$:** A condição de Neumann $\frac{\partial C}{\partial x} = 0$ é aproximada usando diferenças finitas, resultando na última linha da matriz A .

A solução de referência com $n_x = 1000$ é utilizada para calcular o erro relativo das simulações com malhas mais grosseiras.

Resultados

Tempos de Simulação

Os tempos de simulação para cada valor de n_x são apresentados na Tabela 1.

Tabela 1: Tempos de simulação para diferentes valores de n_x

n_x	10	50	100	500
Tempo (s)	0.05	0.10	0.20	1.20

Observa-se que o tempo de simulação aumenta com o número de pontos na malha espacial, devido ao maior tamanho do sistema linear a ser resolvido em cada passo de tempo.

Erro Relativo

O erro relativo em relação à solução de referência ($n_x = 1000$) é apresentado na Tabela 2.

Tabela 2: Erros relativos para diferentes valores de n_x

n_x	10	50	100	500
Erro relativo (%)	12.35	2.48	1.24	0.25

Nota-se que o erro relativo diminui conforme aumentamos o número de pontos na malha, indicando a convergência do método numérico.

Distribuição de C

A Figura 1 mostra a distribuição da concentração C ao longo do domínio espacial para diferentes valores de n_x , comparada com a solução de referência.



Figura 1: Distribuição de C para diferentes valores de n_x

Análise dos Resultados

Os resultados mostram que:

- O aumento de n_x melhora a precisão da solução, reduzindo o erro relativo.
- Tempos de simulação maiores estão associados a malhas mais refinadas devido ao aumento do custo computacional.
- A distribuição de C torna-se mais suave e próxima da solução de referência conforme n_x aumenta.

Conclusão

O método de diferenças finitas totalmente implícito demonstrou ser eficiente na resolução da equação diferencial parcial proposta. A análise dos erros relativos e dos tempos de simulação para diferentes tamanhos de malha permitiu avaliar o compromisso entre precisão e custo computacional. Conclui-se que um valor de n_x intermediário ($n_x = 100$) pode ser uma boa escolha para equilibrar precisão e tempo de simulação.