Estudo do Método de Diferenças Finitas para Equação da Onda unidimencional

Lucas Amaral Taylor

Neste trabalho, aplicaremos o Método de Diferenças Finitas (MDF) para resolução numérica de equações de onda unidimensional em três exemplos pré-selecionados no enunciado.

I. Introdução

No enunciado do trabalho foram passados três exemplos, exercícios-problema, em cada um deles, vamos desenvolver o que está sendo solicitado. De maneira, geral, os três exemplos, seguem como estrutura a plotagem de dados e o cálculo do erro. Para essas duas tarefas, utilizaremos o Método de Diferenças Finitas (MDF) aplicado junto à linguagem de `Python` com auxílo das bibliotecas `numpy` e `matplotlib.pyplot`.

II. Descrição da parte teórica do trabalho

No presente relatório, aplicaremos o Método de Diferenças Finitas (MDF) para resolução da *equação da velocidade da onda* dada por:

$$u_{tt} = c^2 u_{xx}, \quad c > 0. \tag{1}$$

em três problemas distintos. Em cada um deles, são fornecidas condições iniciais:

$$u(x,0) = \Phi(x) \quad \text{e} \quad u_t(x,0) = \Psi(x) \tag{2}$$

Para a resolução, utilizaremos o esquema de diferenças finitas dado por:

$$U_m^{n+1} = c^2\lambda^2(U_{m-1}^n + U_{m+1}^n) + 2(1 - c^2\lambda^2)U_m^n - U_m^{n-1} \tag{3}$$

$$U_m^0 = \Phi(x_m) \tag{4}$$

$$U_m^1 = \frac{c^2\lambda^2}{2}(\Phi_{m-1} + \Phi_{m+1}) + (1 - c^2\lambda^2)\Phi_m + \tau\Psi_m \tag{5}$$

As condições de fronteira consideradas são:

i. Dirichlet: $u(a,t) = \alpha(t)$ ou $u(b,t) = \beta(t)$

ii. Neumann: $u_x(a,t) = \phi(t)$ ou $u_x(b,t) = \psi(t)$

iii. Combinação das anteriores

Utilizamos as seguintes aproximações para cada condição de fronteira:

i. Para Dirichlet:

$$\text{Se } u(a,t) = \alpha(t) : U_0^{n+1} = \alpha(t_{n+1}) \tag{6}$$

$$\text{Se } u(b,t) = \beta(t) : U_M^{n+1} = \beta(t_{n+1}) \tag{7}$$

ii. Para Neumann de ordem 1:

$$\text{Se } u_x(a,t) = \phi(t) : U_0^{n+1} = U_1^{n+1} - h\phi(t_{n+1}) \tag{8}$$

$$\text{Se } u_x(b,t) = \psi(t) : U_M^{n+1} = U_{M-1}^{n+1} + h\psi(t_{n+1}) \tag{9}$$

iii. Para Neumann de ordem 2:

$$\text{Se } u_x(a,t) = \phi(t) : U_0^{n+1} = \frac{4U_1^{n+1} - U_2^{n+1} - 2h\phi(t_{n+1})}{3} \tag{10}$$

$$\text{Se } u_x(b,t) = \psi(t) : U_M^{n+1} = \frac{4U_{M-1}^{n+1} - U_{M-2}^{n+1} + 2h\psi(t_{n+1})}{3} \tag{11}$$

onde:

- $u(x,t)$ é a solução exata no ponto x e no instante t
- U_m^n é a aproximação numérica de $u(x_m, t_n)$
- c é a velocidade da onda
- h é o espaçamento da malha espacial
- τ é o espaçamento da malha temporal
- $\lambda = \tau/h$ é a razão entre os espaçamentos
- M é o número de pontos da malha espacial
- $x_m = a + mh$ são os pontos da malha espacial
- $t_n = n\tau$ são os pontos da malha temporal
- $\Phi(x)$ é a condição inicial de posição
- $\Psi(x)$ é a condição inicial de velocidade

III. Implementação e explicação da resolução da tarefa

A respeito da implementação da tarefa, temos que foi utilizado o ambiente *Python* e as bibliotecas `matplotlib.pyplot` e `numpy`. Por questão de simplicidade, o programa foi desenvolvido em um único arquivo, `main.py`. O programa `main.py` é constituído por uma função principal, a função `main()` e outras cinco funções.

No que diz respeito à função `main()`, trata-se de um controle de fluxo para o usuário selecionar o exemplo trabalhado. No que diz respeito às outras, temos duas funções principais: `resolve_eq_onda()` e a função `aplica_cond_contorno()`, e três funções dedicadas a cada exemplo proposto pelo enunciado: `exemplo01()`, `exemplo02()` e `exemplo03()`. No presente relatório, vamos realizar uma análise mais profunda nas duas primeiras funções citadas e pontuar as particularidades das funções dedicadas à cada exemplo.

A. Função `resolve_eq_onda()`

A função `resolve_eq_onda()` é dada por:

```
def resolve_eq_onda(a, b, T, h, lambda_val, c=1.0, phi=None, psi=None,
                    cont_esq=None, cont_dir=None,
                    tipo_cont_esq='dirichlet',
                    tipo_cont_dir='dirichlet',
                    ordem_neumann=2):

    # Discretização do domínio
    M = int((b - a) / h)
    tau = lambda_val * h
    N = int(T / tau)

    x = np.linspace(a, b, M + 1)
    t = np.linspace(0, T, N + 1)
    U = np.zeros((N + 1, M + 1))

    # Condições iniciais
    if phi is not None:
        U[0, :] = phi(x)
    if psi is not None:
        # Primeiro passo temporal (ordem 2)
        U[1, 1:-1] = (c ** 2 * lambda_val ** 2 / 2) * (U[0, :-2] + U[0, 2:]) + \
                     (1 - c ** 2 * lambda_val ** 2) * U[0, 1:-1] + \
                     tau * psi(x[1:-1])
        U[1, 0] = aplica_cond_contorno(U, t, 1, 0, cont_esq, tipo_cont_esq, h, ordem_neumann)
```

```
        U[1, -1] = aplica_cond_contorno(U, t, 1, M, cont_dir, tipo_cont_dir, h, ordem_neumann)

    # Esquema explícito para demais passos
    for n in range(1, N):
        U[n + 1, 1:-1] = c ** 2 * lambda_val ** 2 * (U[n, :-2] + U[n, 2:]) + \
                         2 * (1 - c ** 2 * lambda_val ** 2) * U[n, 1:-1] - \
                         U[n - 1, 1:-1]
        U[n + 1, 0] = aplica_cond_contorno(U, t, n + 1, 0, cont_esq, tipo_cont_esq, h,
    ordem_neumann)
        U[n + 1, -1] = aplica_cond_contorno(U, t, n + 1, M, cont_dir, tipo_cont_dir, h,
    ordem_neumann)

    return U, x, t
```

A função `resolve_eq_onda()` recebe os seguintes parâmetros:

- `a, b`: Limites do intervalo espacial $[a, b]$
- `T`: Tempo final da simulação
- `h`: Passo espacial (discretização em x)
- `lambda_val`: Razão τ/h, onde τ é o passo temporal
- `c`: Velocidade da onda (padrão = 1.0)
- `phi`: Função para condição inicial de posição $u(x, 0)$
- `psi`: Função para condição inicial de velocidade $u_t(x, 0)$
- `cont_esq`: Função para condição de contorno em $x = a$
- `cont_dir`: Função para condição de contorno em $x = b$
- `tipo_cont_esq`: Tipo da condição de contorno em $x = a$ (Dirichlet ou Neumann)
- `tipo_cont_dir`: Tipo da condição de contorno em $x = b$ (Dirichlet ou Neumann)
- `ordem_neumann`: Ordem de aproximação para condições de Neumann (1 ou 2)

Nela, definidas as variáveis `M`, `tau` e `N`, estas relacionadas com a equação de diferenças finitas; as variáveis `x`, `t` e `u`, associadas com a malha de pontos. Após a definição de variáveis, o programa começa o tratamento com a condição inicial alinhado com as equações (2) e (4). Por fim, são definidos os pontos interiores em dois laços. O primeiro é responsável pelo cálculo do primeiro passo, isto é, quando $n = 1$ usando a condição inicial proveniente da equação (5). Enquanto o segundo, é responsável pelos pontos onde $2 \leq n \leq N$ usando a equação (3). Ambos os laços levam em consideração as condições de fronteiras e suas características são tratadas via `aplica_cond_contorno()`, apresentada a seguir.

B. Função `aplica_cond_contorno()`

A função `aplica_cond_contorno()` é apresentada como:

```
def aplica_cond_contorno(U, t, n, m, func_cont, tipo_cont, h, ordem):
    if tipo_cont.lower() == 'dirichlet':
        return func_cont(t[n])
    elif tipo_cont.lower() == 'neumann':
        if m == 0:  # Contorno esquerdo
            if ordem == 1:
                return U[n, 1] - h * func_cont(t[n])
            return (4 * U[n, 1] - U[n, 2] - 2 * h * func_cont(t[n])) / 3
        else:  # Contorno direito
            if ordem == 1:
                return U[n, -2] + h * func_cont(t[n])
            return (4 * U[n, -2] - U[n, -3] + 2 * h * func_cont(t[n])) / 3
```

Os parâmetros recebidos pela função são:

- `U`: Matriz com a solução numérica, onde `U[n,m]` representa a aproximação de $u(x_m, t_n)$
- `t`: Vetor com os pontos da malha temporal $t_n = n\tau$
- `n`: Índice temporal atual
- `m`: Índice espacial do ponto de fronteira (0 para esquerda, M para direita)
- `func_cont`: Função que define o valor da condição de contorno
- `tipo_cont`: Tipo da condição de contorno ('dirichlet' ou 'neumann')
- `h`: Passo espacial da malha
- `ordem`: Ordem de aproximação para condição de Neumann (1 ou 2)

Os parâmetros `h`, `tipo_cont` e `ordem` já constam na lista anterior da função `resolve_eq_onda()`.

A função `aplica_cond_contorno()` aplica a condição de contorno conforme descrita pelo enunciado do exemplo. Para as equações de Dirichlet são seguidas as equações (6) e (7), para as de Neumman de ordem 1 são seguidas as equações (8) e (9) e , por fim, para as de Neumman de ordem 2 são seguidas as equações (10) e (11).

C. As funções `exemplo`

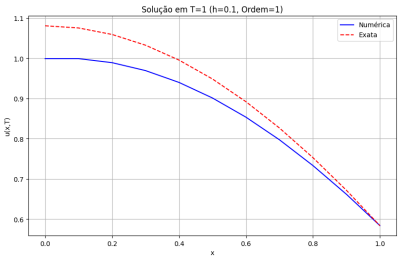
Por fim, finalizando a presente seção, cabe uma breve explicação a respeito das funções `exemplo1()`, `exemplo2()` e `exemplo3()`. Essas três funções são responsáveis por particularizar cada exemplo proposto, isto é, seguir as instruções específicas para cada exemplo.

- `exemplo1()`
  - Implementa problema com solução analítica $u(x,t) = \cos(x+t) + \cos(x-t)$
  - Testa convergência com diferentes h $(1/10, 1/20, 1/40)$ e ordens do método de Neumann $(1, 2)$
  - Plota comparação entre solução numérica e analítica
- `exemplo2()`
  - Simula onda com condição inicial $\Phi(x) = \begin{cases} 1 - |x| & \text{se } |x| \leq 1 \\ 0 & \text{caso contrário} \end{cases}$
  - Usa diferentes resoluções espaciais ($h = 1/10$ até $1/80$) com $\lambda = 0.95$
  - Aplica condições mistas (Dirichlet à esquerda, Neumann à direita)
- `exemplo3()`
  - Simula onda com condição inicial $\Phi(x) = e^{-1000(x-0.5)^2} \sin(300x)$
  - Testa $\lambda = 1.0$ e $\lambda = 0.45$ com $h = 1/300$ fixo
  - Analisa solução em $t = 0, 0.25, 2$ e 10, verificando periodicidade e erros

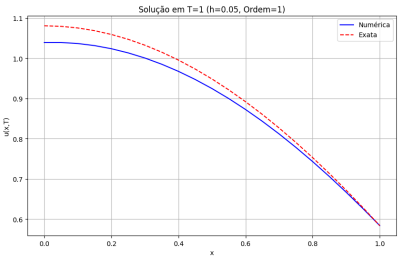
IV. Apresentação dos resultados

A. Exemplo 01

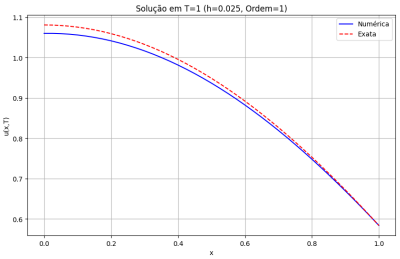
Primeiramente, cabe apresentar os gráficos gerados pelo programa:



(a) Solução com $h = 0.1$

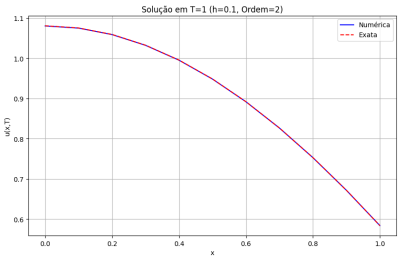


(b) Solução com $h = 0.05$

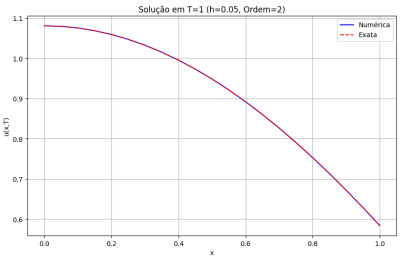


(c) Solução com $h = 0.025$

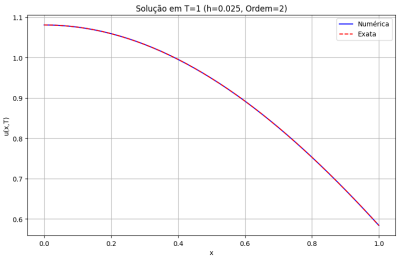
Figura 1: Soluções numéricas com aproximação de primeira ordem



(a) Solução com $h = 0.1$



(b) Solução com $h = 0.05$



(c) Solução com $h = 0.025$

Figura 2: Soluções numéricas com aproximação de segunda ordem

Note que os gráficos que utilizam aproximação de primeira ordem distanciam-se mais da solução exata ao serem comparados com os gráficos de segunda ordem, ou seja, a partir deste fato, pode-se afirmar que os gráficos de primeira

ordem apresentam um erro maior do que os gráficos de segunda ordem. Além disso, à medida que o h diminui, a solução numérica permanece cada vez mais próxima da solução exata, tanto nos gráficos de primeira ordem, tanto nos de segunda.

Por fim, nota-se que tais fatos evidenciados nas figuras, são confirmados pela tabela de cálculo do erro apresentada abaixo:

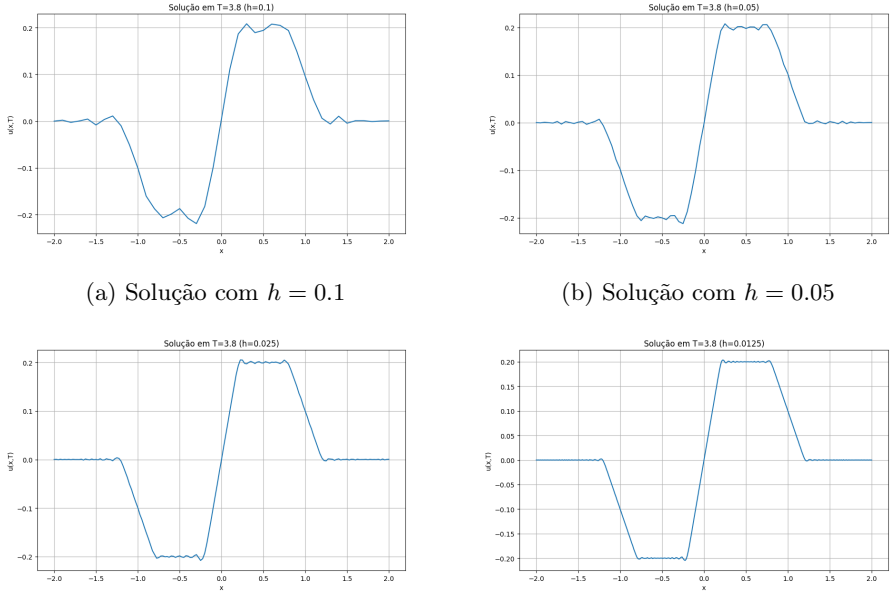
Tabela I: Erro máximo para diferentes valores de h e ordens da condição de Neumann

h	**Ordem 1**	**Ordem 2**
0.1	8.171035e-02	3.937531e-04
0.05	4.148152e-02	5.096550e-05
0.025	2.089094e-02	6.474309e-06

Por fim, analisando a tabela, podemos realizar algumas observações no que diz respeito à ordem de convergência. Para ordem 1, temos que h ao reduzir h pela metade, o erro se reduz aproximadamente pela metade, o que indica uma convergência linear. Já a ordem 2, temos que, ao reduzir h pela metade, o erro reduzir-se em um fator próximo h^3, ou seja, uma convergência cúbica.

B. Exemplo 02

Os gráficos obtidos pelo programa foram:



(a) Solução com $h = 0.1$

(b) Solução com $h = 0.05$

(c) Solução com $h = 0.025$

(d) Solução com $h = 0.0125$

Figura 3: Convergência da solução numérica em $T = 3.8$ para diferentes valores de h

Analisando os gráficos, observa-se que o refinamento da malha (redução de h) resulta em uma suavização progressiva das oscilações numéricas. Os intervalos $[-2, -1]$, $[-1, 0]$, $[0, 1]$ e $[1, 2]$ apresentam menos perturbações conforme h diminui de 0.1 para 0.0125, evidenciando uma melhor aproximação da solução contínua. As transições entre os patamares tornam-se mais suaves, e as regiões planas mostram menos ruído numérico.

Agora, analisando o erro, levando em consideração o que foi posto no enunciado do exemplo: ponto $x = 2$ como um ponto de simetria para todo t, obtivemos os seguintes resultados para análise do erro:

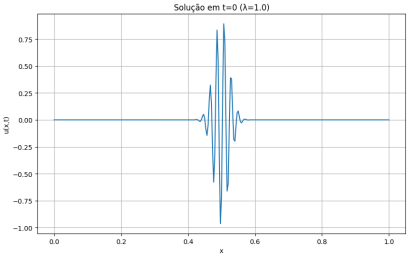
Tabela II: Análise do erro de simetria da solução em T=3.8

h	**Erro de simetria**
0.1000	4.160728e-01
0.0500	4.159548e-01
0.0250	4.130921e-01
0.0125	4.072185e-01

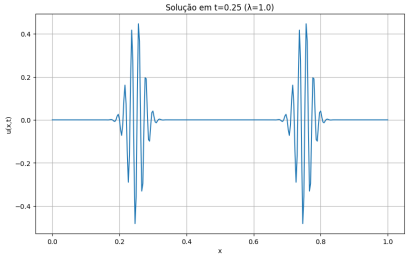
Analisando os dados apresentados na Tabela II, observa-se que mesmo com sucessivos refinamentos da malha, o erro de simetria permanece praticamente constante na ordem de 10^{-1}. Esta característica sugere que o refinamento da malha não está sendo eficiente para melhorar a precisão da solução em termos de sua propriedade de simetria, evidenciando uma convergência consideravelmente lenta do método numérico para este aspecto específico do problema.

C. Exemplo 03

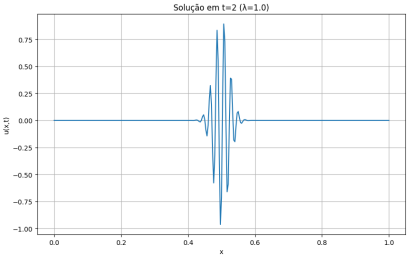
As imagens geradas pelo programa foram:



(a) Pulso inicial em $t = 0$



(b) Separação dos pulsos em $t = 0.25$



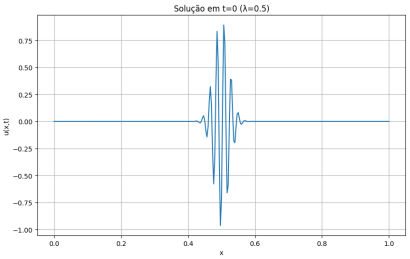
(c) Solução em $t = 2$

main.pymain.py

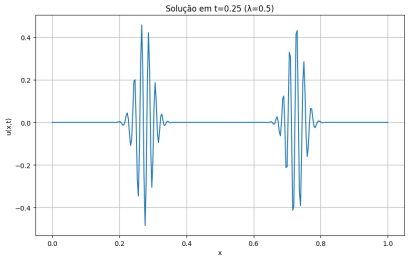
(d) Solução em $t = 10$

Figura 4: Solução numérica com $\lambda = 1.0$ e $h = 1/300$

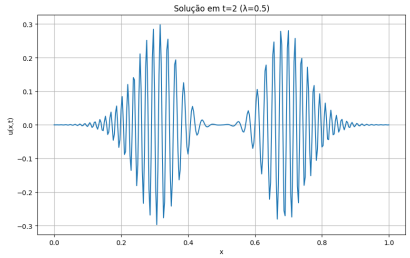
No primeiro grupo de imagens, quando $\lambda = 1$ e $h = 300$, temos que as imagens do tempo inicial de $t = 0$, $t = 2$ e $t = 10$ são semelhantes. Enquanto em $t = 0.25$ nota-se a presença de duas ondas. Este fato deve-se ao comportamento do objeto estudado



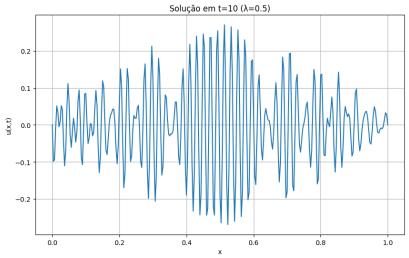
(a) Pulso inicial em $t = 0$



(b) Separação dos pulsos em $t = 0.25$



(c) Solução em $t = 2$



(d) Solução em $t = 10$

Figura 5: Solução numérica com $\lambda = 0.5$ e $h = 1/300$

As semelhanças vistas entre os gráficos do mesmo grupo não são vistas no segundo grupo. Em $t = 0$ e $t = 0.25$, pode-se notar que são semelhantes ao que foi apresentado no primeiro grupo de imagens. Já em $t = 2$ e $t = 10$, temos que além de serem extremamente diferentes entre si, também são diferentes ao serem comparadas com gráficos do mesmo tempo do grupo anterior. Por fim, é importante destacar o seguinte trecho do enunciado do exemplo 03:

> "Quando atingem a fronteira (a primeira vez em $t = 0.5$), eles refletem e então viajam na direção oposta. Nos instantes t iguais a inteiros pares a solução é igual ao seu valor em $t = 0$".

Esse fato pode ser explicado pela periodicidade em relação ao tempo da solução da equação expressa por:

$$u(x, 2n) = u(x, 0), \quad n \in \mathbb{Z}$$

Ela ocorre pelo fato de $c = 1$, ou seja, quando a velocidade da onda leva 1 unidade de tempo para percorrer o domínio $L = 1$. Além disos, vale destacar que as condições de Dirichlet homogêneas nos extremos causam reflexão com inversão de fase, e após uma ida e volta completa (2 unidades de tempo), a onda retorna à sua configuração inicial, repetindo este ciclo. A respeito à estimação do erro, obtemos a seguinte tabela.

Tabela III: Erro em relação à condiçãoUpdate main.py inicial para diferentes valores de λ

λ	**Erro em t=2**	**Erro em t=10**
1.0	1.221245e-15	3.719247e-15
0.5	9.634010e-01	1.026740e+00

Para $\lambda = 1$, os erros em $t = 2$ e $t = 10$, apesar de distintos, compartilham a mesma ordem de grandeza (10^{-15}), sendo praticamente nulos. Isto confirma a propriedade teórica do esquema ser exato quando $c\lambda = 1$. O mesmo não se pode dizer em relação à $\lambda = 0.5$: apesar de terem valores relativamente próximos (uma diferença de 6×10^{-2}), há acumulação de erros numéricos ao longo do tempo, mesmo com passo temporal menor.

V. Conclusão

Concluí-se que o Método de Diferenças Finitas (MDF) demonstrou-se uma ferramenta eficaz para a resolução numérica de equações de onda unidimensionais da forma apresentada em (1), sujeitas às condições iniciais expostas em (2).

No presente trabalho, foram desenvolvidos três exemplos distintos que, embora compartilhem a natureza de fenômenos ondulatórios unidimensionais, apresentaram características únicas. Cada exemplo permitiu explorar diferentes aspectos da aproximação numérica: o primeiro possibilitou uma análise quantitativa e visual da convergência através da comparação direta entre soluções exata e numérica; o segundo explorou propriedades de simetria em torno de $x = 2$ e sua preservação numérica; e o terceiro demonstrou a capacidade do método em capturar fenômenos físicos como a propagação, reflexão e periodicidade de pulsos ao longo do tempo.

Um aspecto crucial evidenciado foi a sensibilidade do método aos parâmetros numéricos. Em particular, a escolha de $\lambda = \tau/h$ mostrou-se determinante para a precisão e estabilidade das soluções. No primeiro exemplo, observamos como diferentes ordens de aproximação para a condição de Neumann afetam a convergência. No segundo, vimos que mesmo com refinamento da malha, certas propriedades geométricas podem ser difíceis de preservar numericamente. Já no terceiro exemplo, a comparação entre $\lambda = 1$ e $\lambda = 0.5$ demonstrou como a escolha dos parâmetros pode afetar a precisão da solução em tempos longos, com o caso $\lambda = 1$ produzindo resultados mais precisos.

VI. Referências

Strauss, W. A. (2008). *Partial Differential Equations* (2a edição). John Wiley & Sons.