Métodos numéricos para EDPs – trabalho final

Leonardo de Oliveira Carvalho

5 de julho de 2010

1 Solução numérica da Equação de Poisson

Considere a equação de Poisson $\Delta u(x,y) = f(x,y)$ para $(x,y) \in \Omega$, com condição de contorno de Dirichlet $u(x,y) = g(x,y)$ para $(x,y) \in \partial\Omega$. Vamos encontrar uma solução discreta para esta equação em um domínio Ω retangular, usando três métodos distintos.

1.1 Discretização da equação de Poisson

Vamos inicialmente descrever um operador de diferenças aproximando o operador Laplaciano numa grade no plano xy, com N_x pontos na direção x e N_y pontos na direção y.

Através das equações

$$u_{xx}(x_i,y_j) = \frac{u(x_{i-1},y_j) - 2u(x_i,y_j) + u(x_{i+1},y_j)}{\Delta x^2} + \mathcal{O}(\Delta x^2),$$
$$u_{yy}(x_i,y_j) = \frac{u(x_i,y_{j-1}) - 2u(x_i,y_j) + u(x_i,y_{j+1})}{\Delta y^2} + \mathcal{O}(\Delta y^2),$$

podemos definir o Laplaciano discreto Δ_D como

$$(\Delta_D(u))_{i,j} = \frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{\Delta x^2} + \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{\Delta y^2}, \quad i = 1,\ldots,N_x, j = 1,\ldots,N_y.$$

Onde $u_{0,j}$,$u_{i,0}$,$u_{Nx+1,j}$ e $u_{i,Ny+1} = 0$ são definidos pela condição de contorno.

Quando $\Delta x = \Delta y = \Delta s$ teremos

$$(\Delta_D(u))_{i,j} = \frac{1}{\Delta s^2}\left(u_{i-1,j} + u_{i+1,j} - 4u_{i,j} + u_{i,j-1} + u_{i,j+1}\right) \quad i = 1,\ldots,N_x, j = 1,\ldots,N_y.$$

Para a equação de Poisson discreta $\Delta_D u = f$, tem-se para cada $i = 1,\ldots,N_x, j = 1,\ldots,N_y$

$$(\Delta_D(u))_{i,j} = f_{i,j}$$
$$\frac{1}{\Delta s^2}\left(u_{i-1,j} + u_{i+1,j} - 4u_{i,j} + u_{i,j-1} + u_{i,j+1}\right) = f_{i,j}$$
$$u_{i-1,j} + u_{i+1,j} - 4u_{i,j} + u_{i,j-1} + u_{i,j+1} = \Delta s^2 f_{i,j}.$$

Sejam as condições de contorno dadas por

$$u_{0,j} = l_j,$$
$$u_{N_x+1,j} = r_j,$$
$$u_{i,0} = b_i,$$
$$u_{i,N_y+1} = t_i.$$

Para pontos vizinhos ao bordo é possível separar os valores conhecidos das variáveis no sistema de equações visto acima. Por exemplo para $i = 1$ e $j = 1$ tem-se

$$\begin{aligned} u_{0,1} + u_{2,1} - 4u_{1,1} + u_{1,0} + u_{1,2} &= \Delta s^2 f_{1,1} \\ l_1 + u_{2,1} - 4u_{1,1} + b_1 + u_{1,2} &= \Delta s^2 f_{1,1} \\ u_{2,1} - 4u_{1,1} + u_{1,2} &= \Delta s^2 f_{1,1} - l_1 - b_1. \end{aligned}$$

Vamos reescrever este sistema no formato matricial, para isso os valores de u são colocados numa matriz coluna v, onde teremos $v_k = u_{i,j}$ seguindo a relação $k = i + (j-1)Ny$. Desta forma teremos

$$\begin{aligned} u_{i-1,j} &= v_{k-1}, && \text{se } i > 1 \\ u_{i+1,j} &= v_{k+1}, && \text{se } i < Nx \\ u_{i,j-1} &= v_{k-Ny}, && \text{se } j > 1 \\ u_{i,j+1} &= v_{k+Ny}, && \text{se } j < Ny. \end{aligned}$$

O sistema pode ser escrito então como $Av = b$, onde b corresponde aos valores de $\Delta s^2 f$ subtraidos das condições de contorno em pontos vizinhos, e A é uma matrix quadrada ($N_x N_y \times N_x N_y$), com o valor -4 na diagonal principal, e cada linha k de A tem o valor 1 nas posições $k-1$, $k+1$, $k-Ny$ e $k+Ny$, com exceção das posições correspondentes à borda do domínio de u. Por exemplo, para $Nx = 3$ e $Ny = 4$ teremos

$$A = \begin{pmatrix} -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -4 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & -4 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & -4 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -4 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 \end{pmatrix}.$$

Como A é uma matriz esparsa, ao implementarmos o método no computador, podemos utilizar uma estrutura apropriada para este tipo de dado, evitando o armazenamento desnecessário de muitos valores nulos, e otimizando os cálculos que serão feitos. A matriz A pode ser gerada no MATLAB através do seguinte código:

```
function A = Laplacian(Nx,Ny)
     T = Nx*Ny;
     D = sparse(1:T,1:T,-4*ones(1,T),T,T);
     E = sparse(2:T,1:T-1,ones(1,T-1),T,T);
     for i = Nx:Nx:T-1
          E(i+1, i) = 0; % Eliminate boundary values.
     end
     F = sparse(Nx+1:T,1:T-Nx,ones(1,T-Nx),T,T);
     A = F+E+D+E'+F';
end
```

A seguir serão discutidos alguns métodos para resolução de sistemas lineares.

1.2 Solução direta

Neste método calcula-se a solução exata do sistema. No MATLAB isto pode ser feito através do comando `mldivide`, ou, da mesma forma, pelo operador *backslash* através do comando `v = A\b`.

1.3 Solução por métodos iterativos

Métodos iterativos para resolução de um sistema linear $Av = b$ podem ser descritos por

$$Qv^n = (Q - A)v^{n-1} + b,$$

para alguma matriz Q, escolhida de forma que seja simples encontrar v^n a cada iteração, como uma matriz diagonal ou triangular, e que quando $n \to \infty$ $v^n \to v$, e portanto no limite tem-se

$$Qv = (Q - A)v + b$$
$$Av = b$$

Ou seja, v^n converge à solução do sistema.

1.3.1 Gauss-Seidel

O método iterativo de Gauss-Seidel consiste em escolher Q como sendo a matriz triangular inferior da matriz A [3].

Este método pode ser implementado em MATLAB através do seguinte código:

```
function x = GaussSeidel(A, b, e)
    Q = tril(A);
    QA = Q-A;
    x = zeros(size(A,1), 1);
    err = e+1;
    while err > e
        lastx = x;
        x = Q\(QA*x + b);
        err = norm(x - lastx, inf)/norm(lastx, inf);
    end
end
```

Como Q é uma matriz triangular inferior, o MATLAB resolve o sistema `x = Q\(QA*x + b)` em cada iteração através de uma simples substituição para frente (*forward substitution*).

As iterações se repetem até que o erro relativo seja menor ou igual a `e`, quanto menor for este parâmetro, a solução se torna mais próxima da solução exata.

1.3.2 SOR

O método SOR (*Successive Over Relaxation*) utiliza $Q = \alpha D - C$, onde α é um valor real, D é qualquer matriz Hermitiana definida positiva e C qualquer matriz satisfazendo $C + C^* = D - A$, onde C^* é a conjugada transposta de C. Se A for Hermitiana definida positiva, Q for não-singular e $\alpha > \frac{1}{2}$ o método converge para qualquer dado inicial [3].

É comum escolher $C = -L$, onde L é a parte triangular inferior de A (sem a diagonal), e D como a diagonal de A. Na literatura, é costume utilizar o parâmetro $\omega = 1/\alpha$, escrevendo o método como

$$(D + \omega L)v^{n+1} = \omega b - [\omega U + (\omega - 1)D]v^n,$$

onde U é a parte triangular superior de A (sem a diagonal).

O parâmetro α é um fator de relaxamento, que influencia o resíduo $r^n = b - Av^n$, dependendo do valor escolhido, r^n tende a zero mais rapidamente. É possível ver que quando $\alpha = 1$ o método

é equivalente ao Gauss-Seidel. Em certos casos é possível determinar o valor ótimo para este parâmetro, isto é, um valor que torne a convergência mais rápida.

O método pode ser implementado em MATLAB com o código:

```
function x = SOR(A, b, alpha, e)
     Q = alpha*diag(diag(A)) + tril(A,-1);
     QA = Q-A;
     x = zeros(size(A,1), 1);
     err = e+1;
     while err > e
          lastx = x;
          x = Q\(QA*x + b);
          err = norm(x - lastx, inf)/norm(lastx, inf);
     end
end
```

Assim como no método de Gauss-Seidel, a matriz Q é triangular inferior, logo o sistema `x = Q\(QA*x + b)` é resolvido pelo MATLAB através de substituição para frente.

1.4 Experimentos

Foram feitos testes comparando a solução da equação de Poisson utilizando os três métodos descritos acima para resolver o sistema linear.

Para gerar os dados do problema, foi considerada uma equação suave $h(x, y)$, de onde calculou-se $f(x, y) = \Delta h(x, y)$ e os valores de contorno da grade retangular.

Foi utilizado $h(x, y) = \frac{1}{3}x^3 - \frac{1}{2}x^2 + \frac{3}{16}x + \frac{1}{3}y^3 - \frac{1}{2}y^2 + \frac{3}{16}y$ com $x \in [0, 1]$ $y \in [0, 1]$, de onde $f(x, y) = 2x + 2y - 2$.

O domínio $[0, 1] \times [0, 1]$ é discretizado numa grade com $(N + 2)^2$ pontos, com espaçamento $\Delta s = 1/(N + 1)$. O seguinte código em MATLAB calcula h e f nesta grade

```
ds = 1/(N+1);
i = 1:N+2;
x = ds*(i-1);
h = zeros(N+2, N+2);
f = h;
for j = 1:N+2
     y = ds*(j-1);
     h(i,j) = x.^3/3 - x.^2/2 + 3/16*x + y.^3/3 - y.^2/2 + 3/16*y;
     f(i,j) = 2*x + 2*y - 2;
end
```

A solução u pode ser inicializada copiando os valores do bordo de h, o que pode ser feito com os comandos

```
u = h;
u(2:N+1, 2:N+1) = 0;
```

A matriz A do sistema é calculada fazendo

```
A = Laplacian(N, N);
```

Para calcular b basta tomar os valores de $\Delta s^2 f$ e subtrair os valores das bordas vizinhas, fazendo

```
b = ds*ds*f(2:N+1, 2:N+1);
b(1:N, 1) = b(1:N, 1) - u(2:N+1, 1);
b(1:N, N) = b(1:N, N) - u(2:N+1, N+2);
b(1, 1:N) = b(1, 1:N) - u(1, 2:N+1);
b(N, 1:N) = b(N, 1:N) - u(N+2, 2:N+1);
b = matrix2column(b);
```

Onde o comando `matrix2column` transforma dados matriciais em uma matriz coluna.

Assim, os dados do sistema estão prontos. O sistema é resolvido usando os três métodos descritos acima. O método de Gauss-Seidel foi utilizado com o comando

```
v = GaussSeidel(A,b, 1E-4);
```

Para utilizar o SOR é preciso informar um valor para o parâmetro α, dependendo da escolha deste parâmetro o método converge mais lentamente ou mais rapidamente. Segundo [2], o valor ótimo para α neste caso é

$$\alpha = \frac{2 + \sqrt{4 - \left(\cos\left(\frac{\pi}{N_x}\right) + \cos\left(\frac{\pi}{N_y}\right)\right)^2}}{4}.$$

Como estamos trabalhando com $N_x = N_y = N$, temos

$$\begin{aligned}
\alpha &= \frac{2 + \sqrt{4 - \left(\cos\left(\frac{\pi}{N}\right) + \cos\left(\frac{\pi}{N}\right)\right)^2}}{4} \\
&= \frac{2 + \sqrt{4 - \left(2\cos\left(\frac{\pi}{N}\right)\right)^2}}{4} \\
&= \frac{2 + \sqrt{4 - 4\cos^2\left(\frac{\pi}{N}\right)}}{4} \\
&= \frac{2 + \sqrt{4\,\mathrm{sen}^2\left(\frac{\pi}{N}\right)}}{4} \\
&= \frac{2 + 2\,\mathrm{sen}\left(\frac{\pi}{N}\right)}{4} \\
&= \frac{1 + \mathrm{sen}\left(\frac{\pi}{N}\right)}{2}.
\end{aligned}$$

Como a convergência do método SOR é garatida para qualquer dado inicial para matrizes definidas positivas, trabalhamos então com o sistema equivalente $-Av = -b$, pois a matriz A é definida negativa. Então o SOR é executado fazendo-se

```
v = SOR(-A,-b, 0.5*(1 + sin(pi/N)), 1E-4);
```

O problema foi resolvido para diferentes valores de N. A tabela abaixo indica o tempo em segundos gasto por cada método para resolver o sistema.

N	10	50	100	150	200
Solução direta	0.000517s	0.015082s	0.083636s	0.157438s	0.267260s
Gauss-Seidel	0.002537s	0.123603s	1.159065s	3.568958s	8.022997s
SOR	0.001192s	0.016260s	0.120293s	0.396335s	1.011407s

Tabela 1: Tempo gasto por cada método para resolver o sistema para diferentes valores de N.

Percebe-se que em geral a solução direta obtém o resultado mais rápido, pois o MATLAB utiliza algoritmos otimizados para resolver sistemas lineares. Entre os métodos iterativos, o método SOR é sempre mais rápido do que o Gauss-Seidel, evidenciando o efeito do parâmetro de relaxamento na aceleração do método.

Também é interessante analisar o número de iterações realizadas por cada método, o que é mostrado na Tabela 2. Verifica-se que a quantidade de iterações realizadas pelo método SOR é significativamente menor do que as efetuadas pelo método de Gauss-Seidel.

N	10	50	100	150	200
Gauss-Seidel	79	901	2171	3051	3423
SOR	24	105	206	306	405

Tabela 2: Número de iterações realizadas por cada método.

Pode-se também analisar o erro nas aproximações, calculando a norma da diferença entre o valor obtido e a solução exata (dada por h). Esta informação pode ser vista na Tabela 3.

N	10	50	100	150	200
Solução direta	8.3267e-17	1.0755e-16	2.0123e-16	1.3184e-16	1.5266e-16
Gauss-Seidel	4.4403e-05	0.0011	0.0041	0.0090	0.0148
SOR	8.0175e-06	5.9056e-06	5.9214e-06	6.0767e-06	6.2857e-06

Tabela 3: Erro obtido por cada método.

Observa-se que a solução direta e o método SOR apresentaram pouca variação no erro. A solução direta obteve um erro bem menor do que os métodos iterativos. É possível obter um erro menor diminuindo a cota para o erro relativo nos métodos iterativos, porém isto prejudica o tempo de execução, já que é necessário calcular mais iterações. O método SOR, além de ser mais rápido, também apresenta um erro menor do que o obtido pelo método Gauss-Seidel. Executando o SOR com `e = 1E-14` para $N = 100$ obtém-se um resultado com erro 2.2343e-15 em aproximadamente 0,3 segundos, o que ainda é mais rápido do que o Gauss-Seidel com `e = 1E-4` (que levou mais do que 1 segundo, conforme Tabela 1).

2 Ondas aquáticas lineares

2.1 Uma relação para a aplicação Dirichlet-Neumann

Consideremos um problema relativo a uma função harmônica $\phi(x,y)$ definida na faixa $\Omega = \{(x,y), y \in (-1,0)\}$, com condição de contorno de Dirichlet no bordo superior ($\phi(x,0) = \psi(x)$), e de Neumann no bordo inferior (homogênea). Seja a aplicação Dirichlet-Neumann dada por $DtN[\psi](x) = \frac{d\phi}{dn}(x,0)$.

Podemos escrever $\phi(x,y) = \int_{-\infty}^{\infty} F_k(y)e^{ikx}dk$, de onde teremos

$$\Delta\phi(x,y) = \int_{-\infty}^{\infty} \left(-k^2F_k(y)e^{ikx} + F_k''(y)e^{ikx}\right) dk.$$

Como $\Delta\phi(x,y) = 0$, então $-k^2F_k(y)e^{ikx} + F_k''(y)e^{ikx} = 0$, de onde temos

$$-k^2F_k(y) + F_k''(y) = 0,$$

que é uma EDO cuja solução é:

$$F_k(y) = C_ke^{ky} + D_ke^{-ky},$$

onde C_k e D_k são constantes definidas pelas condições de contorno.

Teremos então

$$F_k'(y) = kC_ke^{ky} - kD_ke^{-ky}.$$

Pela condição no bordo inferior teremos $0 = \frac{d\phi}{dy}(x,-1) = \int_{-\infty}^{\infty} F_k'(-1)e^{ikx}dk$, logo

$$0 = F_k'(-1) = kC_ke^{-k} - kD_ke^{k},$$

de onde tiramos que $C_k = D_ke^{2k}$. Usando esta relação podemos reescrever F_k e F_k' como

$$F_k(y) = D_k\left(e^{k(y+2)} + e^{-ky}\right),$$
$$F_k'(y) = D_k\left(ke^{k(y+2)} - ke^{-ky}\right).$$

Usando agora as condições do bordo superior e a transformada de Fourier inversa para ψ teremos

$$\int_{-\infty}^{\infty} F_k(0)e^{ikx}dk = \phi(x,0) = \psi(x) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} \widehat{\psi}(k)e^{ikx}dk,$$

de onde $\frac{1}{\sqrt{2\pi}}\widehat{\psi}(k) = F_k(0) = D_k\left(e^{2k}+1\right)$, e então

$$D_k = \frac{\widehat{\psi}(k)}{\left(e^{2k}+1\right)\sqrt{2\pi}}.$$

Desta forma podemos calcular o operador Dirichlet-Neumann:

$$\begin{aligned}
DtN[\psi](x) = \frac{d\phi}{dy}(x,0) &= \int_{-\infty}^{\infty} F_k'(0)e^{ikx}dk \\
&= \int_{-\infty}^{\infty} D_k\left(ke^{2k} - k\right)e^{ikx}dk \\
&= \int_{-\infty}^{\infty} \left(\frac{\widehat{\psi}(k)}{\left(e^{2k}+1\right)\sqrt{2\pi}}\right)\left(ke^{2k} - k\right)e^{ikx}dk \\
&= \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} \left(\frac{e^{2k}-1}{e^{2k}+1}\right)k\widehat{\psi}(k)e^{ikx}dk \\
&= \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} \tanh(k)k\widehat{\psi}(k)e^{ikx}dk.
\end{aligned}$$

Como $ik\widehat{\psi}(k) = \widehat{\psi_x}(k)$ então $k\widehat{\psi}(k) = -i\widehat{\psi_x}(k)$, e chegamos então à seguinte relação:

$$DtN[\psi](x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} -i \tanh(k) \widehat{\psi_x}(k) e^{ikx} dk. \tag{1}$$

2.2 Ondas aquáticas dispersivas - solução numérica

Seja o problema de ondas aquáticas dispersivas dado por

$$\beta\phi_{xx} + \phi_{yy} = 0, \text{ em } \Omega_\beta = \left\{(x, y) \in \mathbb{R}^2; -1 \leq y \leq 0\right\},$$

com condição de Neumann $d\phi/dn = 0$ em $y = -1$, e na reta $y = 0$ temos

$$\phi_t = -\eta, \qquad \eta_t = \frac{1}{\beta}\phi_y.$$

$\eta(x, t)$ indica a elevação da superfície do mar. Podemos reescrever este problema como

$$\psi_t = -\eta, \qquad \eta_t = \frac{1}{\beta} DtN[\psi],$$

onde $\psi(x, t) = \phi(x, 0, t)$.

Para obter uma solução numérica vamos utilizar as relações

$$\frac{\psi(x, t^{n+1}) - \psi(x, t^n)}{\Delta t} = -\left(\eta(x, t^n) + \frac{\Delta t}{2\beta}\phi_y(x, 0, t^n)\right) + \mathcal{O}(\Delta t^2)$$
$$\frac{\eta(x, t^{n+1}) - \eta(x, t^n)}{\Delta t} = \frac{1}{\beta}\frac{\phi_y(x, 0, t^{n+1}) + \phi_y(x, 0, t^n)}{2} + \mathcal{O}(\Delta t^2)$$

para obter o esquema de evolução

$$\psi^{n+1} = \psi^n - \Delta t\left(\eta^n + \frac{\Delta t}{2\beta}\phi_y^n\right)$$
$$\eta^{n+1} = \eta^n + \frac{\Delta t}{2\beta}\left(\phi_y^{n+1} + \phi_y^n\right)$$

onde η^n, ψ^n e ϕ_y^n são aproximações respectivamente de η, ψ e ϕ_y no instante de tempo t^n.

Para calcular $\phi_y^n = DtN[\psi^n]$ vamos utilizar uma relação para $DtN[\psi]$ adaptada para este caso. Através de cálculos semelhantes aos usados para obter a equação 1 (bastando substituir $\Delta\phi$ por $\beta\phi_{xx} + \phi_{yy}$), encontramos

$$DtN[\psi](x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} k\sqrt{\beta} \tanh(k\sqrt{\beta}) \widehat{\psi}(k) e^{ikx} dk. \tag{2}$$

Assim, através desta equação, podemos calcular $DtN[\psi^n]$ utilizando FFT e IFFT para trabalharmos no domínio da frequência. O seguinte código em MATLAB executa esta operação

```
function v = DtN(psi_n, beta)
    M = size(psi_n,2);
    k = -M/2:M/2-1;

    ui = fftshift(fft(psi_n));
    sk = sqrt(beta)*k;
    vi = sk.*tanh(sk).*ui;
    v = real(ifft(ifftshift(vi)));
end
```

O método para encontrar ψ^n e η^n foi implementado com o código a seguir, onde devem ser informados os dados iniciais ψ^0 e η^0, a profundidade β, o número de passos no tempo `Nt`, e a variação no tempo Δt.

```
function [psi,eta] = water_wave(psi0, eta0, beta, Nt, deltat)
     % Number of points.
     N = size(psi0, 2);

     % Initialization.
     psi = zeros(Nt,N);
     eta = zeros(Nt,N);
     psi(1,:) = psi0;
     eta(1,:) = eta0;

     % Iterations.
     lDtN = DtN(psi0, beta);
     for n = 2:Nt
          lpsi = psi(n-1, :);
          leta = eta(n-1, :);

          psi(n,:) = lpsi - deltat*(leta + 0.5*deltat/beta*lDtN);

          nDtN = DtN(psi(n,:), beta);
          eta(n,:) = leta + 0.5*deltat/beta*(nDtN + lDtN);
          lDtN = nDtN;
     end
end
```

Para testes, utilizamos como perfil inicial a gaussiana

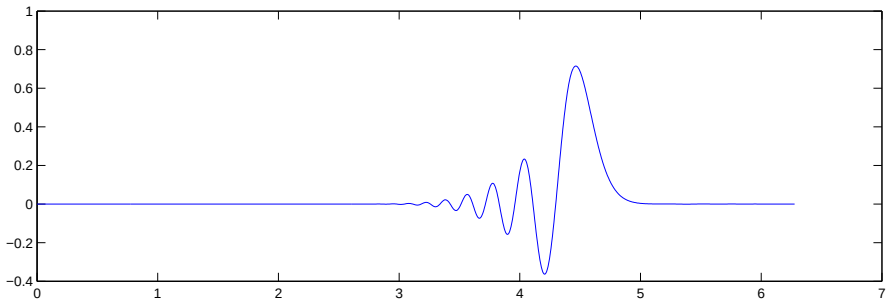
$$\psi(x,0) = \psi_0(x) = e^{-(x-x_0)^2/\sigma^2},$$

com $x_0 = \pi/2$ e $\sigma = \sqrt{0.01}$. Para a outra condição inicial precisamos definir $\psi_t(x,0) = -\eta(x,0)$, seja $\xi(x) \equiv \psi_t(x,0)$, escolhendo $\widehat{\xi}(k) = -i\omega\widehat{\psi_0}(k)$, onde

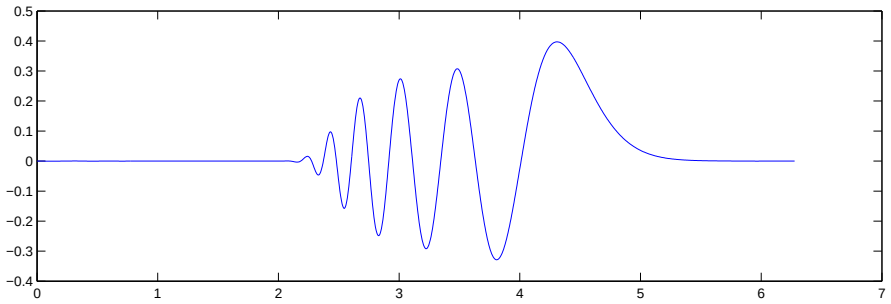
$$\omega^2 = \frac{k}{\sqrt{\beta}}\tanh(k\sqrt{\beta}),$$

fazemos com que todos os modos de oscilação no momento inicial estejam viajando para a direita [1].

A Figura 1 ilustra resultados obtidos para o potencial de velocidade em $N = 1024$ pontos, com $\Delta t = 0,01$, após 300 passos no tempo utilizando diferentes valores para β.

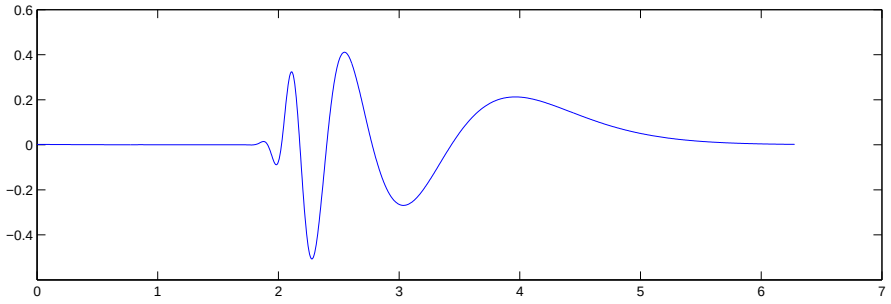


(a) $\beta = 0.001$

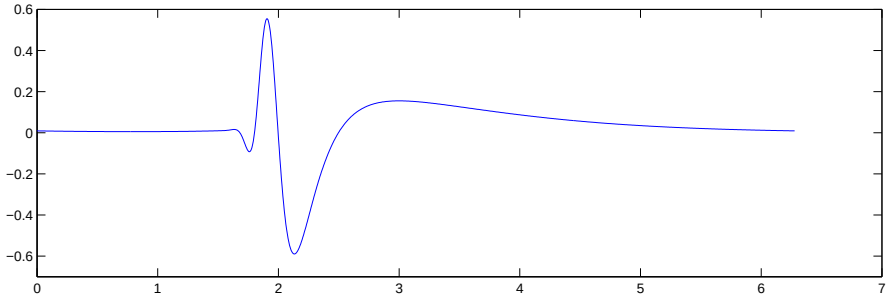


(b) $\beta = 0.01$

Figura 1: Propagação de um potencial de velocidade: regimes fracamente dispersivos.



(a) $\beta = 0.1$



(b) $\beta = 1$

Figura 2: Propagação de um potencial de velocidade: dispersão completamente desenvolvida.

2.3 Comparação entre métodos numéricos

Utilizando o operador Dirichlet-Neumann vimos que é possível encontrar a solução $\psi(x,t)$ e $\eta(x,t)$ de forma eficiente utilizando apenas os dados das condições de contorno. Se fosse utilizada uma estratégia de diferenças finitas seria necessário encontrar a solução de ϕ dentro da faixa Ω_β a cada instante de tempo, o que seria bastante custoso e pouco eficiente. Outra estratégia é usar BIEM (*Boundary Integral Equation Method*) onde os valores de ϕ_y na borda superior podem ser encontrados resolvendo equações de Fredholm, o problema se limita então à borda, mas a cada instante é preciso resolver um sistema denso, não-simétrico, o que também é custoso se comparado ao cálculo de FFT utilizado para o operador Dirichlet-Neumann. Além de mais eficiente, o cálculo de FFT possui precisão espectral, enquanto que o sistema do BIEM pode ser mal-condicionado, levando a soluções pouco precisas. Logo o operador DtN se mostra uma ferramenta útil na resolução numérica do problema.

2.4 Regime de águas rasas

Quando $\beta \ll 1$ estamos no regime de águas rasas (ou ondas longas). Vamos analisar o que ocorre quando β tende a zero.

Inicialmente podemos eliminar o η do sistema fazendo:

$$\begin{aligned}\psi_{tt} = \eta_t &= -\frac{1}{\beta} DtN[\psi] \\ &= -\frac{1}{\beta\sqrt{2\pi}} \int_{-\infty}^{\infty} -i\sqrt{\beta}\tanh(k\sqrt{\beta})\widehat{\psi_x}(k)e^{ikx}dk \\ &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} i\frac{1}{\sqrt{\beta}}\tanh(k\sqrt{\beta})\widehat{\psi_x}(k)e^{ikx}dk\end{aligned}$$

Escrevendo a tangente hiperbólica como série de potências teremos:

$$\begin{aligned}\frac{1}{\sqrt{\beta}}\tanh(k\sqrt{\beta}) &= \frac{1}{\sqrt{\beta}} \sum_{n=1}^{\infty} \frac{2^{2n}(2^{2n-1})B_{2n}(k\sqrt{\beta})^{2n-1}}{(2n)!} \\ &= \sum_{n=1}^{\infty} \frac{2^{2n}(2^{2n-1})B_{2n}k^{2n-1}\beta^{n-1}}{(2n)!} \\ &= k + \sum_{n=2}^{\infty} \frac{2^{2n}(2^{2n-1})B_{2n}k^{2n-1}\beta^{n-1}}{(2n)!},\end{aligned}$$

onde B_n é o n-ésimo número de Bernoulli

$$B_n = \sum_{k=0}^{n}\sum_{v=0}^{k}(-1)^v \binom{k}{v}\frac{v^n}{k+1}.$$

Quando $\beta \to 0$ teremos então $\frac{1}{\sqrt{\beta}}\tanh(k\sqrt{\beta}) \to k$, logo neste limite teremos

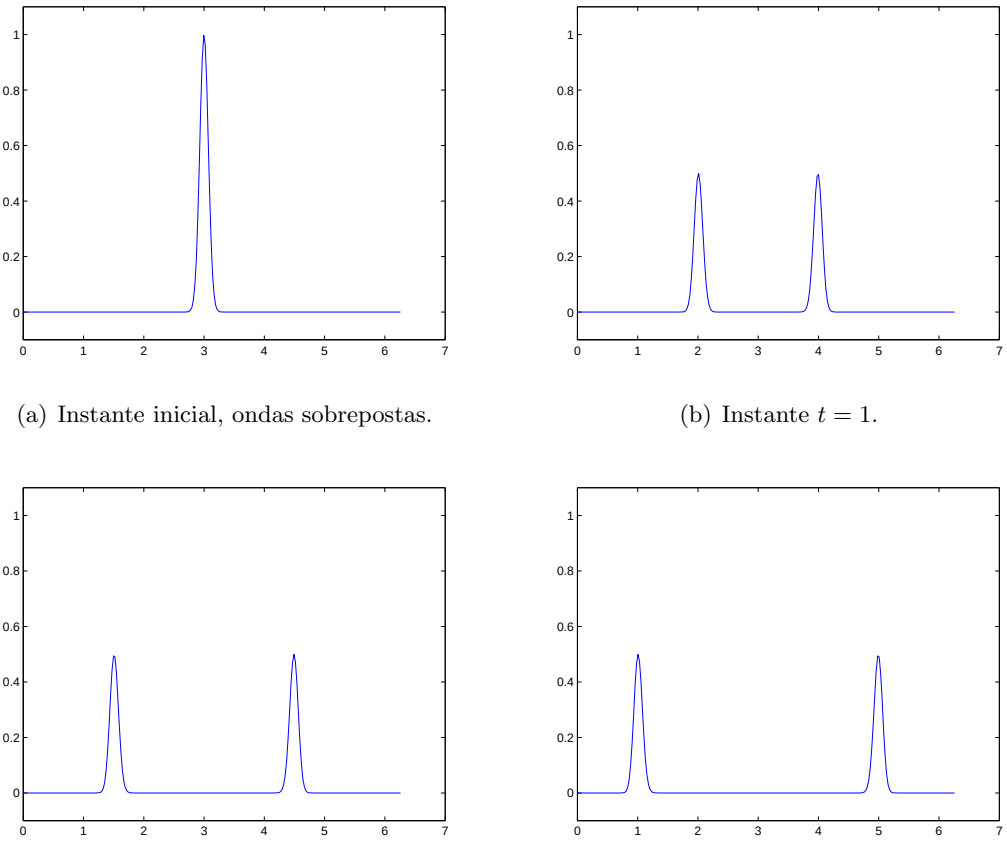
$$\begin{aligned}\psi_{tt} &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} ik\widehat{\psi_x}(k)e^{ikx}dk \\ &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \widehat{\psi_{xx}}(k)e^{ikx}dk \\ &= \psi_{xx}.\end{aligned}$$

Portanto o sistema tende à equação da onda $\psi_{tt} - \psi_{xx} = 0$.

Sabemos que a solução desta equação é

$$\psi(x,t) = \frac{1}{2}\left(\psi_0(x+t) + \psi_0(x-t)\right) + \frac{1}{2}\int_{x-t}^{x+t} \xi(s)ds,$$

onde $\psi_0(x) = \psi(x,0)$ e $\xi(x) = \psi_t(x,0)$ são as condições iniciais. Em particular para $\xi(x) = 0$ temos duas ondas viajando para lados opostos. Podemos verificar isto numericamente com o programa descrito anteriormente. Definindo $\beta = 0,00001$ usando $N = 256$ pontos, $\eta_0 = 0$, utilizando a mesma função gaussiana usada anteriormente, mas com $x_0 = 3$ obtemos o resultado ilustrado na Figura 3.



(a) Instante inicial, ondas sobrepostas.

(b) Instante $t = 1$.

(c) $t = 1,5$.

(d) $t = 2$.

Figura 3: Regime de águas rasas.

Referências

[1] William Artiles and André Nachbin. Asymptotic nonlinear wave modeling through the dirichlet-to-neumann operator. *Methods and Applications of Analysis*, 11(4):475–492, December 2004.

[2] Richard L. Burden and Douglas J. Faires. *Numerical Analysis*. Brooks Cole, 7 edition, December 2000.

[3] David Kincaid and Ward Cheney. *Numerical analysis: mathematics of scientific computing*. Brooks/Cole Publishing Co., Pacific Grove, CA, USA, second edition, 1996.