Vetorização e Estratégias Numéricas na Resolução das Equações de Águas Rasas Via MATLAB

ADOLFO GUILHERME SILVA CORREIA
IME – Instituto Militar de Engenharia
Praça General Tibúrcio, 80
22290-270 – Rio de Janeiro

ANDRÉ NACHBIN
IMPA – Instituto de Matemática Pura e Aplicada
Estrada Dona Castorina, 110
22460-320 – Rio de Janeiro

Resumo

Neste trabalho são aplicadas técnicas para aumentar a eficiência da resolução computacional das equações de águas rasas. As principais técnicas utilizadas foram a vetorização do código, o uso de matrizes esparsas, o emprego de permutações aos elementos das matrizes envolvidas e a aplicação de uma malha espacial irregular. A razão entre o tempo de execução do código final e inicial do exemplo estudado foi menor que 3%. As técnicas aplicadas são gerais e podem ser utilizadas em outros problemas envolvendo sistemas de EDPs lineares.

1 Introdução

Um problema freqüente na computação científica surge quando se exige do computador uma grande quantidade de operações. Não são raros programas que levam horas ou mesmo dias para serem executados. Tornam-se necessários, então, o desenvolvimento e a aplicação de métodos ou algoritmos que reduzam o tempo de execução desses programas.

Este processo de aprimoramento do código será demonstrado através do problema das equações de águas rasas, que consiste basicamente na resolução de um sistema de equações diferenciais parciais lineares.

As principais técnicas utilizadas neste trabalho para aumentar a eficiência da resolução do problema foram:

- a vetorização de trechos críticos do código;
- um melhor aproveitamento da esparsidade das matrizes empregadas;
- a elaboração e aplicação de permutações às matrizes utilizadas;
- a discretização do problema através de uma malha espacial não regular.

No contexto do MATLAB, um código *vetorizado* é aquele que trabalha diretamente com as matrizes ou vetores envolvidos, sem a necessidade de operar-se individualmente cada elemento da matriz ou vetor. A linguagem MATLAB oferece vários recursos para a criação de códigos vetorizados [Hi00].

Matrizes *esparsas* são aquelas com uma grande proporção de elementos nulos. Algoritmos que tomem proveito desta característica geralmente são mais eficientes e possuem uma menor demanda de memória. Novamente, a linguagem MATLAB fornece ferramentas para uma melhor utilização de matrizes esparsas [Hi00].

Durante o processo de resolução do problema proposto, as matrizes utilizadas são submetidas a diversas operações. Algumas destas operações podem ser significativamente otimizadas se a matriz original for *permutada*. Uma permutação eficiente será concebida neste trabalho. Outras possíveis permutações são obtidas diretamente a partir de comandos do MATLAB [Hi00].

A utilização de uma malha espacial irregular pode reduzir o número total de pontos necessários para discretizar o modelo contínuo quando uma ou mais propriedades do sistema variarem rapidamente em uma região, permanecendo constantes nas demais. Com um menor número de pontos, o código se torna mais eficiente.

O método inicial proposto será o mais simples possível, não empregando efetivamente nenhuma das técnicas mencionadas acima e utilizando inversão de matriz para a solução dos sistemas de equações lineares. Posteriormente, e de forma gradativa, serão apresentadas formas de otimizar a solução numérica. O primeiro passo será substituir a inversão por uma decomposição LU. Em seguida, cada uma das técnicas acima indicadas será aplicada.

2 Formulação do Problema Contínuo e Discreto

O problema físico estudado será o comportamento de ondas longas lineares em um canal com um fundo irregular. O sistema de equações diferenciais que rege este fenômeno é conhecido como as equações de águas rasas. Neste contexto, as terminologias 'águas rasas' e 'ondas longas' são equivalentes, já que este regime é controlado pela razão entre a profundidade média e o comprimento de onda. O regime de interesse ocorre quando esta razão é pequena [Na93].

As equações de águas rasas são

$$\begin{cases} u_t + g\eta_x = 0 \\ \eta_t + (hu)_x = 0 \end{cases},$$

em que $u = u(x,t)$ e $\eta = \eta(x,t)$ são funções no espaço e no tempo que representam a velocidade do fluido e a elevação da onda, respectivamente. A aceleração da gravidade é representada por g e $h = h(x)$ é uma função no espaço que representa a profundidade do canal a partir da superfície de repouso [Me83].

Pode-se obter uma simplificação no sistema acima fazendo a troca de variáveis $\psi = hu$. Com essa mudança, o sistema se torna:

$$\begin{cases} \psi_t + (gh)\,\eta_x = 0 \\ \eta_t + \psi_x = 0 \end{cases}.$$

Este sistema é mais simples pois não será necessária a aplicação da regra do produto para a derivação de hu.

O próximo passo será a representação desse sistema de uma forma numérica, ou seja, deve-se aproximar as derivadas por diferenças finitas.

Neste ponto, é importante a definição do tipo de canal que será estudado. Em outras palavras, o canal apresentará variações de profundidade muito bruscas ou será suave? Caso o canal apresente variações bruscas de profundidade, a malha espacial deverá ser mais refinada para que o perfil do fundo do canal seja bem representado em toda a sua extensão. Porém, uma malha muito refinada acarreta em um maior tempo de execução.

Uma saída para este problema no desempenho do programa é usar um malha irregular, que seja mais refinada nos trechos mais íngremes e mais grosseira nos trechos mais suaves. Dessa forma, o programa não perde em precisão e confiabilidade e também não perde muito em desempenho e velocidade.

O método escolhido para se aproximar as derivadas com uma malha espacial irregular foi o proposto por Cathers, Bates e O'Connor [Ca89]. Considerando-se uma malha espacial com J pontos e uma malha temporal com N pontos, o sistema de equações diferenciais pode ser aproximado por

$$\begin{aligned} &\frac{\Delta x_{j-1}}{6\Delta t}\left[(\psi^{n+1} - \psi^n)_{j-1} + 2\,(\psi^{n+1} - \psi^n)_j\right] + \\ &\frac{\Delta x_j}{6\Delta t}\left[2\,(\psi^{n+1} - \psi^n)_j + (\psi^{n+1} - \psi^n)_{j+1}\right] + \end{aligned} \tag{1}$$

$$+\frac{gh_j}{4}(\eta_{j+1}-\eta_{j-1})^{n+1}+\frac{gh_j}{4}(\eta_{j+1}-\eta_{j-1})^{n}=0$$

$$\begin{aligned}&\frac{\Delta x_{j-1}}{6\Delta t}\left[(\eta^{n+1}-\eta^{n})_{j-1}+2\ (\eta^{n+1}-\eta^{n})_{j}\right]+\\&+\frac{\Delta x_{j}}{6\Delta t}\left[2\ (\eta^{n+1}-\eta^{n})_{j}+(\eta^{n+1}-\eta^{n})_{j+1}\right]+\\&+\frac{1}{4}(\psi_{j+1}-\psi_{j-1})^{n+1}+\frac{1}{4}(\psi_{j+1}-\psi_{j-1})^{n}=0\end{aligned}\tag{2}$$

em que $\Delta t=t_{n+1}-t_n=$ constante, $\Delta x_j=x_{j+1}-x_j$, $\psi_j^n=\psi\left(x_j,t_n\right)$ e $\eta_j^n=\eta\left(x_j,t_n\right)$. Os índices j e n são tais que $j=1,2,...,J$ e $n=1,2,...,N$.

No caso particular de uma malha espacial uniforme, tem-se $\Delta x=$ constante, e o sistema fica reduzido às equações

$$\begin{aligned}&\frac{1}{6}\left[\frac{(\psi^{n+1}-\psi^{n})_{j-1}}{\Delta t}+4\ \frac{(\psi^{n+1}-\psi^{n})_{j}}{\Delta t}+\frac{(\psi^{n+1}-\psi^{n})_{j+1}}{\Delta t}\right]+\\&+\frac{gh_j}{2}\left[\frac{(\eta_{j+1}-\eta_{j-1})^{n+1}}{2\Delta x}+\frac{(\eta_{j+1}-\eta_{j-1})^{n}}{2\Delta x}\right]=0\end{aligned}\tag{3}$$

$$\begin{aligned}&\frac{1}{6}\left[\frac{(\eta^{n+1}-\eta^{n})_{j-1}}{\Delta t}+4\ \frac{(\eta^{n+1}-\eta^{n})_{j}}{\Delta t}+\frac{(\eta^{n+1}-\eta^{n})_{j+1}}{\Delta t}\right]+\\&+\frac{1}{2}\left[\frac{(\psi_{j+1}-\psi_{j-1})^{n+1}}{2\Delta x}+\frac{(\psi_{j+1}-\psi_{j-1})^{n}}{2\Delta x}\right]=0.\end{aligned}\tag{4}$$

Pode-se reescrever o sistema geral acima de uma forma mais simples, com os termos no tempo $n+1$ à esquerda do sinal de igual e os termos em n à direita:

$$\begin{aligned}&m_{j-1}\ \psi_{j-1}^{n+1}+(2m_{j-1}+2m_j)\ \psi_j^{n+1}+m_j\ \psi_{j+1}^{n+1}+(-p_j)\ \eta_{j-1}^{n+1}+0\ \eta_j^{n+1}+p_j\ \eta_{j+1}^{n+1}=\\&=m_{j-1}\ \psi_{j-1}^{n}+(2m_{j-1}+2m_j)\ \psi_j^{n}+m_j\ \psi_{j+1}^{n}+p_j\ \eta_{j-1}^{n}+0\ \eta_j^{n}+(-p_j)\ \eta_{j+1}^{n}\end{aligned}\tag{5}$$

$$\begin{aligned}&(-q)\ \psi_{j-1}^{n+1}+0\ \psi_j^{n+1}+q\ \psi_{j+1}^{n+1}+m_{j-1}\ \eta_{j-1}^{n+1}+(2m_{j-1}+2m_j)\ \eta_j^{n+1}+m_j\ \eta_{j+1}^{n+1}=\\&=q\ \psi_{j-1}^{n}+0\ \psi_j^{n}+(-q)\ \psi_{j+1}^{n}+m_{j-1}\ \eta_{j-1}^{n}+(2m_{j-1}+2m_j)\ \eta_j^{n}+m_j\ \eta_{j+1}^{n}\end{aligned}\tag{6}$$

em que $m_j = \Delta x_j/(6\Delta t)$, $p_j = gh_j/4$ e $q = 1/4$.

Analisando este último sistema, torna-se claro que o método empregado é progressivo no tempo, pois a solução no tempo $n+1$ é determinada a partir do tempo n anterior. Nota-se também que o método é implícito, pois não é possível obter *diretamente* cada valor da solução num estágio de tempo a partir do estágio anterior. Para um método implícito em geral, é necessário resolver um sistema de equações algébricas a cada avanço no tempo.

Existem algumas condições para que este problema possa ser resolvido. Primeiramente, é necessário conhecer-se as condições iniciais do problema, ou seja, $\psi(x, t_1)$ e $\eta(x, t_1)$ devem ser dados no instante inicial, aqui denotado por t_1. Além disso, deve-se conhecer o comportamento da onda nas extremidades da região em questão. Logo, as condições de contorno $\psi(x_1, t)$, $\psi(x_J, t)$, $\eta(x_1, t)$ e $\eta(x_J, t)$ devem ser dadas.

A onda proposta para este estudo será uma Gaussiana, com

$$u\left(x, t_1\right) = \eta\left(x, t_1\right) = \exp\left[-\left(x-a\right)^2 \Big/ \varepsilon\right]$$

e

$$\psi\left(x, t_1\right) = h\left(x\right) u\left(x, t_1\right),$$

em que a indica a posição inicial da onda e ε um fator para calibrar a largura da onda.

A condição de contorno será homogênea, ou seja, os valores de u e η nas extremidades serão zero. Obviamente ψ também será nula nas extremidades. A grande vantagem da condição de contorno homogênea é que ela torna o programa mais simples. Entretanto, esta condição de contorno fornece resultados imprecisos caso alguma perturbação (onda) atinja uma das extremidades do canal. Para se contornar este contratempo, pode-se 'aumentar' o tamanho do canal ou diminuir o tempo total de estudo, de forma que as ondas formadas não consigam atingir as extremidades do canal.

Obviamente, o programa aqui desenvolvido também será compatível com outras condições iniciais e/ou condições de contorno, desde que se façam as devidas alterações.

As malhas espacial e temporal determinam duas matrizes $J \times N$, que representam os valores das funções ψ e η em todos os pontos (x_j, t_n). Estas matrizes podem ser justapostas formando uma matriz $2J \times N$, de forma que cada coluna seja formada pelos valores de ψ e η, respectivamente, em cada estágio de tempo.

Entretanto, os valores de ψ e η nas extremidades já são conhecidos e não precisam ser calculados. Restam, então, $2(J-2)$ incógnitas para cada estágio no tempo. A matriz solução terá, então, a dimensão $2(J-2) \times N$, em que a primeira coluna representa o momento inicial conhecido.

O próximo passo será representar as $2(J-2)$ equações do sistema sob a forma matricial $A\vec{s}^{\,n+1} = B\vec{s}^{\,n}$, em que A e B são matrizes formadas pelos coeficientes do sistema e $\vec{s}^{\,n}$ representa a n-ésima coluna da matriz solução e pode ser escrito como:

$$\vec{s}^{\,n} = \begin{bmatrix} \psi_2^n & \psi_3^n & \cdots & \psi_{J-1}^n & \eta_2^n & \eta_3^n & \cdots & \eta_{J-1}^n \end{bmatrix}^T.$$

A matriz A, assim como a matriz B, possui uma característica interessante: ela é formada por quatro matrizes tridiagonais justapostas. Na verdade são apenas três matrizes, sendo que uma se repete. Essas três matrizes são $J-2 \times J-2$ e estão representadas a seguir:

$$M_1 = \begin{bmatrix} 2\,(m_1+m_2) & m_2 & 0 & \cdots & 0 \\ m_2 & 2\,(m_2+m_3) & m_3 & \cdots & 0 \\ 0 & m_3 & 2\,(m_3+m_4) & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & m_{J-2} \\ 0 & \cdots & 0 & \cdots & 2\,(m_{J-2}+m_{J-1}) \end{bmatrix},$$

$$M_2 = \begin{bmatrix} 0 & p_2 & 0 & 0 & \cdots & 0 \\ -p_3 & 0 & p_3 & 0 & \cdots & 0 \\ 0 & -p_4 & 0 & p_4 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & 0 & 0 & -p_{J-1} & 0 \end{bmatrix}, M_3 = \begin{bmatrix} 0 & q & 0 & 0 & \cdots & 0 \\ -q & 0 & q & 0 & \cdots & 0 \\ 0 & -q & 0 & q & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & 0 & 0 & -q & 0 \end{bmatrix}.$$

As matrizes A e B podem ser representadas como

$$A = \begin{bmatrix} M_1 & M_2 \\ M_3 & M_1 \end{bmatrix} \text{ e } B = \begin{bmatrix} M_1 & -M_2 \\ -M_3 & M_1 \end{bmatrix}.$$

Os elementos das matrizes A e B não dependem do valor de n e estas podem ser usadas para todos os estágios no tempo sem alterações.

3 Aspectos de Álgebra Linear

A próxima etapa será resolver os $N-1$ sistemas de equações lineares de $2(J-2)$ variáveis cada. Em um primeiro momento esses sistemas serão resolvidos com a inversão da matriz A. Posteriormente será utilizado outro método.

Em um sistema da forma $A\vec{s}^{\,n+1} = B\vec{s}^{\,n}$, o vetor $\vec{s}^{\,n+1}$ pode ser calculado através da equação $\vec{s}^{\,n+1} = A^{-1}B\vec{s}^{\,n}$. Uma vez que a matriz $C = A^{-1}B$ tenha sido calculada, a progressão no tempo se torna muito simples, pois cada estágio no tempo é obtido a partir de uma simples multiplicação entre a matriz C e o vetor do estágio no tempo anterior.

Como será visto a seguir, a inversão da matriz A não é o método mais eficiente para a resolução deste problema, mas com certeza é o mais simples.

Inicialmente a maior preocupação será a eficiência da estratégia para a solução dos sistemas lineares. Isto será facilitado se, a princípio, uma malha espacial regular for usada. Os detalhes para uma malha irregular serão deixados para um momento posterior quando a propagação de ondas for estudada. Pelas mesmas razões, a profundidade do canal será mantida constante. Outro importante detalhe é que o estudo será feito em um contexto adimensional.

O canal aqui estudado terá 10 unidades de comprimento (uc), sendo que a onda inicial estará centrada na posição 1. Da mesma forma, o tempo total de estudo será de 8 unidades de tempo (ut). Tanto a profundidade quanto a aceleração da gravidade serão unitárias; deste modo a velocidade de propagação da onda, dada por $c_0 = \sqrt{gh}$, também será unitária.

Um último e importante detalhe, é que as variáveis Δx e Δt são independentes entre si, ou seja, o valor de uma não restringe ou limita os valores que a outra pode assumir. Esta é uma característica do método implícito. Entretanto, por critérios de precisão, é interessante que as duas sejam da mesma ordem de grandeza. Inicialmente, essas variáveis terão valores iguais a 0,05, fazendo com que haja (J = 200) pontos na malha espacial e (N = 160) na temporal.

Todas essas grandezas, entretanto, podem assumir outros valores de acordo com o caso a ser analisado.

O código `Onda01.m` (c.f. Apêndice), escrito em linguagem MATLAB, apresenta esta estratégia inicial para o problema.

O esperado é que a onda inicial chegue ao final do canal sem mudar de forma, pois a profundidade é constante. O resultado obtido, mostrado na Figura 1, entretanto não condiz com esta expectativa.

O problema apresentado pela figura é uma sutil deformação da onda à medida em que esta vai se propagando no canal. A amplitude da onda vai diminuindo gradualmente e, além disso, há a formação de uma pequena ondulação atrás da onda. Como pode ser visto na Figura 2 esta deformação também ocorre em um canal duas vezes maior.

Este problema ocorre porque as malhas usadas não são capazes de representar satisfatoriamente o fenômeno. Deve-se, então, usar malhas mais finas.

Essas deformações na onda desaparecem quando as variáveis Δx e Δt assumem valores menores ou iguais a 0,025. Assim sendo, a partir deste instante, o valor 0,02 será usado no código. Desta forma, a malha espacial terá (J = 500) pontos e a temporal (N = 400).

Nota: Em um momento posterior, será interessante trabalhar com canais maiores. Nestas circunstâncias, o valor 0,02 para Δx e Δt pode não ser apropriado, con-

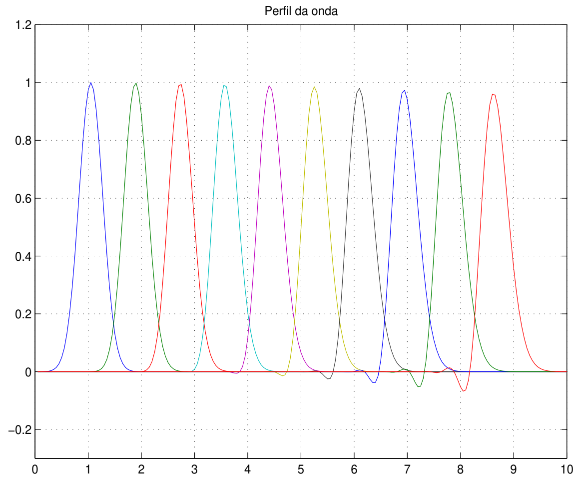

Figura 1: Simulação da evolução da onda em um canal de fundo plano.

duzindo a resultados imprecisos. De fato, este valor só deve ser usado para canais com 20 ou menos uc. O valor 0,01, por sua vez, pode ser utilizado para canais com até 80 uc. Esta imprecisão nos resultados se deve ao acúmulo de erros durante o extenso processo de cálculo.

Com essas mudanças, o código se torna mais preciso, e o resultado obtido condiz com o esperado. A Figura 3 apresenta o perfil de onda esperado.

Surge, entretanto, um novo problema: a execução do código se torna muito lenta. Enquanto o código com $\Delta x = \Delta t = 0,05$ ($J = 500, N = 400$) foi executado em pouco mais de 3 segundos, o código com $\Delta x = \Delta t = 0,02$ ($J = 200, N = 160$) levou quase 1 minuto e meio[1].

O primeiro passo para a otimização deste programa é determinar quanto tempo o computador leva para executar cada parte ou bloco do código. Isto será feito através dos comandos `tic` e `toc`, da maneira demonstrada no código `Onda02.m`. Este programa cria um vetor `t` com os tempos de execução de cada bloco e o tempo total. A Tabela 1 mostra estes tempos e as respectivas porcentagens (tempo em segundos).

Bloco	1	2	3	4	5	6	7	8	Total
Tempo	0,00	0,00	0,05	0,61	41,47	48,05	0,00	0,11	90,29
%	0,00	0,00	0,06	0,68	45,93	53,22	0,00	0,12	100,00

Tabela 1: Inversão

[1] O computador utilizado foi um PC K7 700 MHz com 128 MB, rodando Windows 98 e MATLAB 5.2.

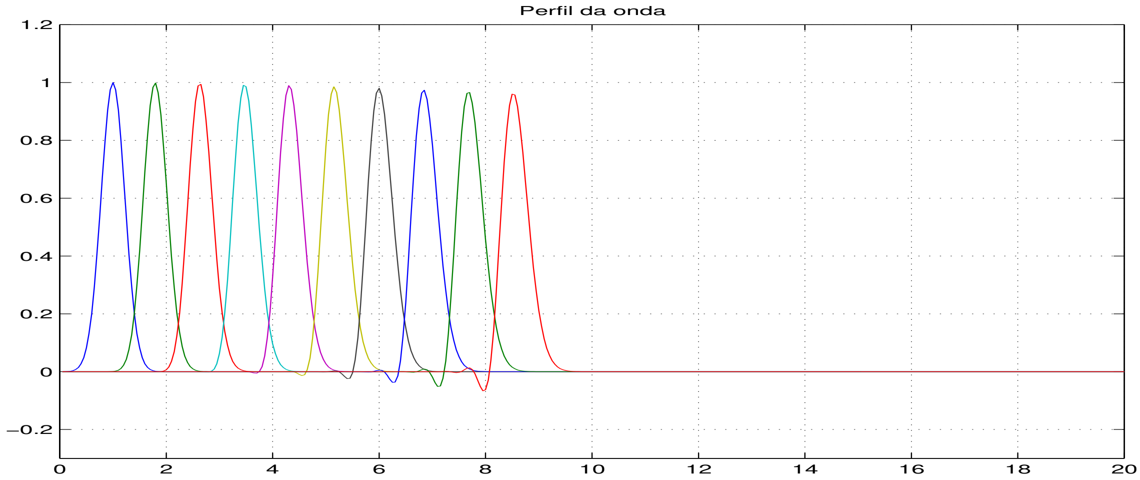

Figura 2: Simulação da evolução da onda em um canal de fundo plano. O canal é duas vezes mais longo que o anterior.

A presença de 'zeros' na Tabela 1 não significa que os blocos correspondentes foram executados instantaneamente, mas sim que estes blocos foram executados em um tempo menor que a precisão dos comandos `tic` e `toc` (0,01s).

O bloco 5 corresponde ao processo de inversão da matriz A e o bloco 6 à progressão no tempo e juntos representam mais de 98% do tempo total de execução. Assim sendo, deve-se buscar novas alternativas que substituam ou aumentem a eficiência desses blocos.

Os métodos mais comuns para a resolução de sistemas de equações lineares em computadores são aqueles que fazem uso da eliminação Gaussiana. Um dos métodos diretos mais eficientes é o da decomposição ou fatoração LU, que pode ser utilizado na resolução de praticamente qualquer sistema de equações lineares.

Este método consiste na decomposição da matriz dos coeficientes em duas matrizes triangulares, sendo uma inferior (L) e outra superior (U). Estas matrizes são tais que o produto LU é igual à matriz dos coeficientes original. Após a decomposição, o sistema é resolvido através da aplicação do simples algoritmo de substituição em dois sistemas triangulares.

A linguagem MATLAB dispõe do comando `lu` para decompor a matriz A nas duas matrizes triangulares desejadas. Este comando também fornece uma matriz permutação P, de forma que o sistema, inicialmente escrito como $A\vec{s}^{\,n+1} = B\vec{s}^{\,n}$, pode ser representado por $LU\vec{s}^{\,n+1} = PB\vec{s}^{\,n}$. Fazendo $\vec{r} = U\vec{s}^{\,n+1}$, o sistema fica reduzido aos dois sistemas triangulares

$$\left\{ \begin{array}{l} L\vec{r} = PB\vec{s}^{\,n} \\ U\vec{s}^{\,n+1} = \vec{r} \end{array} \right. .$$

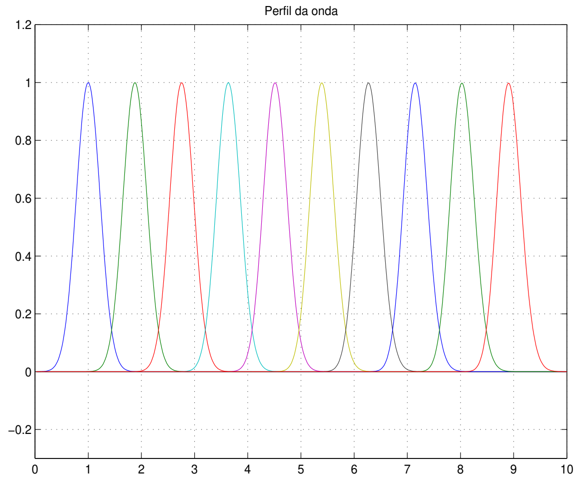

Figura 3: Simulação da evolução da onda em um canal de fundo plano. Simulação feita com uma melhor resolução numérica. A depressão na parte posterior da onda desaparece.

O primeiro sistema é resolvido através de substituição direta e o segundo através de uma retro-substituição.

A vantagem deste método em relação ao método da inversão é que a decomposição LU geralmente é executada mais rapidamente do que a inversão. Este método, entretanto, apresenta uma desvantagem. Embora a matriz A se mantenha constante em relação ao tempo, o vetor com os termos independentes deve ser atualizado a cada avanço no tempo. Logo, antes de se solucionar efetivamente cada sistema com as substituições, será necessário efetuar o produto da matriz B pelo vetor solução do estágio anterior.

O código `Onda03.m` representa este método e a Tabela 2 mostra o tempo de execução dos blocos 5 e 6 e as respectivas porcentagens. Para fins de comparação, a Tabela 2 traz também os dados da Tabela 1.

Método		Bloco 5	Bloco 6	Total
I	Tempo	28,40	$3,94 \cdot 10^3$	$3,96 \cdot 10^3$
	%	0,72	99,25	100,00
II	Tempo	41,47	48,05	90,29
	%	45,93	53,22	100,00

Tabela 2: Decomposição LU (I) e Inversão (II)

Como era esperado, a decomposição LU foi executada mais rapidamente do que a inversão, mas o tempo gasto para a progressão no tempo foi espantosamente alto.

Uma boa maneira para compreender porque a progressão no tempo demorou tanto é executar a listagem abaixo e analisar os resultados:

```
tic
i = 0;
y = zeros(1,10/.0001+1);
for t = 0:.0001:10
   i = i+1;
   y(i) = sin(t);
end
t1 = toc

tic
t = 0:.0001:10;
y = sin(t);
t2 = toc
```

Este código calcula o seno de 100001 valores no intervalo [0,10]. Este procedimento é feito de dois modos distintos. Primeiramente, utiliza-se um laço `for`, e, em seguida, a versão vetorizada do mesmo código. O tempo de execução de cada bloco acima demonstra que o código vetorizado é bem mais eficiente. Neste caso, `t1`=1,09 e `t2`=0,06.

De modo geral, é sempre recomendado evitar o uso de construções em que cada elemento de um vetor ou matriz seja tratado por vez, pois a linguagem MATLAB é de altíssimo nível, sendo capaz de executar rapidamente construções vetorizadas.

É possível alterar o código referente à progressão no tempo no método LU de forma a torná-lo vetorizado. A linguagem MATLAB possui um operador direto para a resolução de sistemas que pode ser utilizado neste caso. Em um sistema da forma $K\vec{x} = \vec{v}$, o vetor com as incógnitas é diretamente determinado através do operador '\', com o comando `x=K\v`. Este comando faz a eliminação Gaussiana e as substituições correspondentes. Segundo a documentação fornecida com o software, o algoritmo utilizado na resolução de sistemas triangulares é essencialmente o mesmo que foi demonstrado no código `Onda03.m`. A diferença é que com o operador '\' o algoritmo é executado de forma vetorizada.

O código `Onda04.m` representa este último método, e a Tabela 3 apresenta os tempos de execução dos blocos em questão. Novamente os dados da Tabela 1 são repetidos.

Com o uso do operador '\' para a resolução dos sistemas, a progressão no tempo foi executada bem mais rapidamente, em um tempo compatível com o

Método		Bloco 5	Bloco 6	Total
I	Tempo	26,48	133,63	161,26
	%	16,42	82,87	100,00
II	Tempo	41,47	48,05	90,29
	%	45,93	53,22	100,00

Tabela 3: Decomposição LU com o operador '\' (I) e Inversão (II)

que se esperava. Entretanto, apesar deste ganho significativo de desempenho, a progressão no tempo fazendo uso das matrizes L e U foi mais lenta do que no caso da inversão da matriz e, no geral, este método foi menos eficiente que o método da inversão.

Esta relativa lentidão da progressão no tempo no caso da decomposição LU se deve ao fato de que, em cada etapa, este método executa uma multiplicação matricial e a resolução de dois sistemas triangulares. No caso da inversão da matriz, a progressão no tempo é feita apenas com uma multiplicação matricial.

Para realmente tornar este método mais eficiente do que o método da inversão, deve-se levar em conta uma característica fundamental das matrizes A e B: a maior parte dos elementos dessas matrizes são nulos. A linguagem MATLAB pode manipular esse tipo de matrizes, chamadas esparsas, de uma forma mais eficiente, trabalhando somente com os elementos não nulos. Dessa forma, há uma economia de memória e um melhor desempenho.

Para que o MATLAB reconheça essas matrizes como esparsas, basta adicionar os comandos `A=sparse(A)` e `B=sparse(B)` logo após a inicialização das matrizes A e B.

A Tabela 4 apresenta os tempos de execução para o código `Onda04.m` com essa modificação e repete os dados da Tabela 3.

Método		Bloco 5	Bloco 6	Total
I	Tempo	3,89	13,90	18,73
	%	20,77	74,21	100,00
II	Tempo	26,48	133,63	161,26
	%	16,42	82,87	100,00
III	Tempo	41,47	48,05	90,29
	%	45,93	53,22	100,00

Tabela 4: Decomposição LU com A e B esparsas (I), Decomposição LU com A e B completas (II) e Inversão (III)

Como mostram os dados da Tabela 4, este método se mostrou bem mais eficiente que os outros dois. Este ganho tão alto de velocidade é obtido devido à pequena quantidade de elementos não nulos que as matrizes A e B possuem.

A densidade de uma matriz pode ser definida como a razão entre o número de elementos não nulos e o número total de elementos da matriz. A densidade de uma matriz qualquer S pode ser calculada através do comando `nnz(S)/prod(size(S))`. Neste exemplo, a densidade das matrizes A e B é de apenas 0,5% .

O uso do comando `sparse` também torna o processo de inversão mais rápido, como mostra a Tabela 5. Entretanto, para uma maior eficiência deste método, a matriz solução também deve ser tratada como esparsa (esta matriz é realmente esparsa nos estágios iniciais do programa).

Método		Bloco 5	Bloco 6	Total
I	Tempo	4,89	20,43	26,31
	%	18,59	77,65	100,00
II	Tempo	41,47	48,05	90,29
	%	45,93	53,22	100,00
III	Tempo	3,89	13,90	18,73
	%	20,77	74,21	100,00

Tabela 5: Inversão com A, B e sol esparsas (I), Inversão com A, B e sol completas (II) e Decomposição LU (III)

Este método é mais rápido que a inversão sem o comando `sparse`, mas é mais lento que o método com a decomposição LU com as declarações de esparsidade.

Uma regra geral para matrizes esparsas é que quanto menor o número de elementos não nulos de uma matriz, mais veloz será o programa. Entretanto, as matrizes L e U possuem uma densidade relativamente alta (25%). Caso este valor fosse menor, certamente haveria algum ganho de velocidade.

Uma forma de se obter matrizes L e U menos densas é alterar a disposição das colunas da matriz A antes de se efetuar a decomposição LU. Determinadas permutações, quando aplicadas à matriz A, podem reduzir drasticamente o número de elementos não nulos das matrizes L e U. O maior problema nesta situação é determinar uma permutação que seja adequada.

Intuitivamente, uma permutação que poderia conduzir a melhores resultados seria uma que levasse os elementos não-nulos da matriz A para perto da diagonal principal dando assim lugar a uma banda (faixa com elementos não-nulos) mais estreita. Esta matriz A permutada possuirá um 'perfil diagonal' (uma matriz com uma banda bem estreita) e provavelmente irá gerar matrizes L e U menos densas.

Na formulação deste problema, foi proposto que os vetores $\vec{s}^{\,n}$ fossem da forma

$$\vec{s}^{\,n} = \begin{bmatrix} \psi_2^n & \psi_3^n & \cdots & \psi_{J-1}^n & \eta_2^n & \eta_3^n & \cdots & \eta_{J-1}^n \end{bmatrix}^T.$$

Deste modo, as $J-2$ primeiras colunas da matriz A representam os coeficientes das incógnitas ψ_i^{n+1} e as $J-2$ colunas restantes os coeficientes das incógnitas η_i^{n+1}. Pode-se alterar esta disposição intercalando-se as colunas da matriz A de forma que as colunas ímpares representem os coeficientes das incógnitas ψ_i^{n+1} e as pares os coeficientes das incógnitas η_i^{n+1}. Assim sendo, os vetores $\vec{s}^{\,n}$ serão da forma

$$\vec{s}^{\,n} = \begin{bmatrix} \psi_2^n & \eta_2^n & \psi_3^n & \eta_3^n & \cdots & \psi_{J-1}^n & \eta_{J-1}^n \end{bmatrix}^T.$$

A matriz permutada obtida ainda não possui o perfil de uma matriz diagonal. Para que seus elementos não-nulos estejam concentrados próximos à diagonal principal, deve-se ainda alterar a disposição das linhas seguindo o mesmo princípio utilizado para a permutação das colunas. Em outras palavras, as $J-2$ primeiras linhas devem ser transferidas para as linhas ímpares e as demais linhas transferidas para as linhas pares.

Como pode ser visto na Figura 4, esta permutação realmente atinge o objetivo proposto de levar todos os elementos não-nulos para perto da diagonal principal. A Figura 4 também mostra as matrizes L geradas a partir das matrizes A original e permutada. Este gráfico foi obtido com o comando `spy`, de forma que cada ponto representa um elemento não-nulo da matriz.

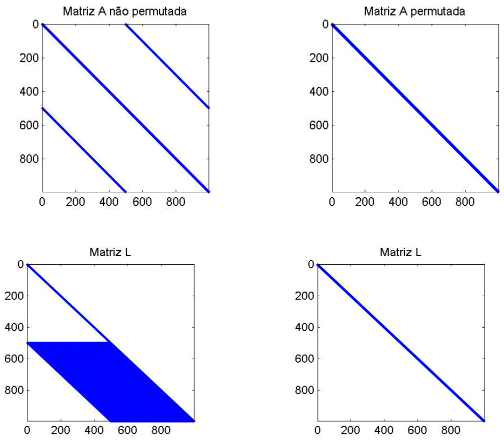

Figura 4: Perfis da matriz L **sem** e **com** o efeito da permutação. As partes escuras representam elementos não-nulos. Na parte inferior de L os elementos não-nulos foram criados durante o processo da decomposição LU.

A permutação efetuada alterou a disposição das linhas e das colunas da matriz A. Uma análise mais cuidadosa revela que a permutação das linhas é desnecessária. O tempo de execução e o perfil das matrizes L e U geradas não sofre alterações significativas se a disposição original das linhas for mantida. Na verdade, qualquer permutação de linhas é perdida ou mudada quando da execução do comando `lu`, já que existe um pivoteamento. O código `Onda05.m` apresenta o método em que apenas as colunas são permutadas. A Figura 5 mostra o perfil da matriz A quando somente suas colunas são permutadas e também seu perfil quando somente as linhas são permutadas. As matrizes L correspondentes também são exibidas.

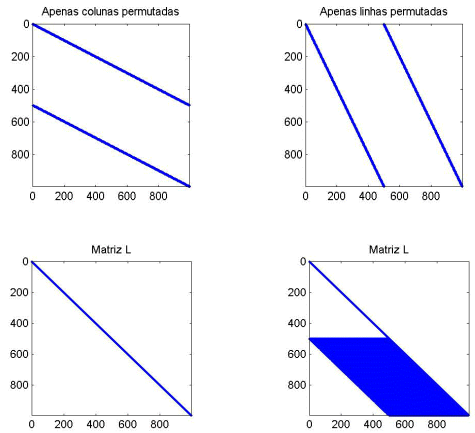

Figura 5: Perfis da matriz L **com** permutações de linha ou de coluna.

Neste exemplo, a permutação concebida intuitivamente foi bastante eficiente, gerando matrizes L e U bastante esparsas. Em outras situações, entretanto, encontrar uma permutação eficiente pode se mostrar uma tarefa bastante difícil. Existem algoritmos já implementados no MATLAB que retornam possíveis permutações. São elas:

- Column minimum degree permutation (colmmd);
- Symmetric minimum degree permutation (symmmd);
- Symmetric reverse Cuthill-McKee permutation (symrcm).

Os comandos `colmmd`, `symmmd` e `symrcm` retornam vetores que representam essas permutações.

A Figura 6 mostra o perfil da matriz A original e o perfil (nada intuitivo) de cada uma das três matrizes permutadas segundo os vetores fornecidos pelos comandos acima.

Essas matrizes, quando decompostas, fornecem as matrizes triangulares inferiores L apresentadas na Figura 7.

Através da Figura 7 fica bem claro que essas permutações também tornaram a matriz L bem menos densa. O mesmo vale para a matriz U. A Tabela 6

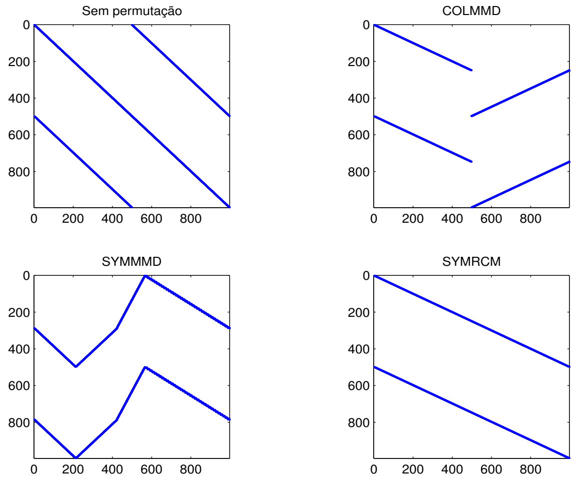

Figura 6: Perfis da matriz do sistema linear **sem** e **com** o efeito das diversas permutações. As partes escuras representam elementos não-nulos.

apresenta o número de elementos não nulos e a densidade de cada uma dessas matrizes. A Tabela 6 também inclui os dados da primeira permutação concebida intuitivamente, que será chamada de permutação diagonal.

	Sem permut.	Diagonal	COLMMD	SYMMMD	SYMRCM
Não nulos	249993	2991	2994	4208	2992
Densidade (%)	25,20	0,30	0,30	0,42	0,30

Tabela 6: Número de elementos não nulos e densidade das matrizes L.

Com o uso de matrizes esparsas e o uso dessas permutações o processo de decomposição LU e a progressão no tempo se tornam ainda mais rápidos. O tempo de execução de alguns desses processos se tornaram mais rápidos que a própria precisão de 0,01s dos comandos `tic` e `toc`. A Tabela 7 mostra o tempo total de execução do código usando cada uma dessas permutações.

	Sem permutação	Diagonal	COLMMD	SYMMMD	SYMRCM
Tempo	19,33	2,15	2,30	2,30	2,09

Tabela 7: Tempo total de execução com cada permutação.

A partir deste instante, o tamanho do canal será aumentado de 10 para 20 para tornar a execução mais lenta e, desta forma, mensurável. Pelas mesmas razões o valor do tempo total de estudo passará de 8 para 18. Estas modificações implicam em $J = 1000$ e $N = 800$, gerando matrizes 1996×1996.

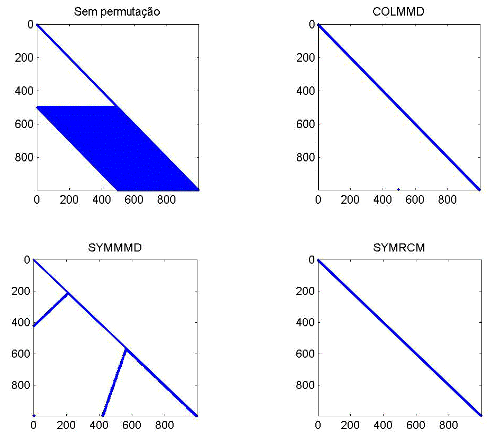

Figura 7: Perfis da matriz L **sem** e **com** o efeito das diversas permutações. As partes escuras representam elementos não-nulos.

A Tabela 8 apresenta os tempos de execução referentes a cada um dos quatro tipos de permutação. O novo tempo de execução do método sem permutação também está presente. Os métodos com as permutações fornecidas pelo MATLAB estão apresentados no código `Onda06.m`.

Permutação		Bloco 5	Bloco 6	Total
Diagonal	Tempo	0,22	9,06	24,60
	%	0,89	36,83	100,00
COLMMD	Tempo	0,77	8,46	24,66
	%	3,12	34,31	100,00
SYMMMD	Tempo	0,28	8,45	23,61
	%	1,17	35,79	100,00
SYMRCM	Tempo	0,33	8,46	22,51
	%	1,47	37,58	100,00
Sem permutação	Tempo	35,60	94,41	142,20
	%	25,04	66,39	100,00

Tabela 8: Tempos de execução com cada permutação (J =1000).

Além do significativo ganho de desempenho, há outros dados interessantes nesta tabela: os processos de decomposição LU e progressão no tempo passaram a corresponder a menos da metade do tempo total de execução do programa. Outro dado interessante, observado nas Tabelas 7 e 8, é que a permutação diagonal intuitiva acabou sendo menos eficiente que a permutação SYMRCM.

A partir deste momento é necessário otimizar os outros blocos do código. A Tabela 9 apresenta os tempos de execução de cada um dos blocos do código `Onda06.m` com a permutação SYMRCM.

Bloco	1	2	3	4	5	6	7	8	Total
Tempo	0,00	0,05	0,17	9,66	0,33	8,46	0,60	3,24	22,51
%	0,00	0,22	0,76	42,91	1,47	37,58	2,67	14,39	100,00

Tabela 9: Tempos de execução de cada bloco do código `Onda06.m` (SYMRCM).

Os blocos 1, 2 e 8 já estão otimizados e não há muito o que se fazer a respeito deles. Em relação ao bloco 3, onde são inicializados alguns vetores, haverá algum ganho de desempenho se os laços `for` forem substituídos pelo código vetorizado equivalente.

No bloco 4, as matrizes A e B são construídas e depois convertidas pelo comando `sparse`. Este processo requer uma grande quantidade de memória para alocar as matrizes completas e é muito lento. Uma saída mais inteligente seria construir as matrizes A e B diretamente como esparsas. O comando `sparse` também pode ser utilizado para criar essas matrizes diretamente.

Um último processo que pode ser otimizado diz respeito aos blocos 6 e 7. No bloco 6, a matriz solução é finalmente construída. Entretanto, não é necessário armazenar a solução para todos os estágios no tempo. Os únicos estágios que precisam ser armazenados são aqueles necessários para a construção do gráfico da evolução da onda no canal. Dessa forma, esses estágios serão armazenados em uma matriz auxiliar `aux1` e a matriz `sol` terá dimensão $2J - 4 \times 2$. Além disso, o laço no bloco 7 deverá ser incorporado ao laço no bloco 6. Essa mudança também afeta o bloco 2, onde é feita a declaração da matriz `sol`.

O código com todas essas mudanças está apresentado no código `Onda07.m` e a Tabela 10 contém os novos tempos de execução de cada bloco.

	Bloco	1	2	3	4	5	6	7	8	Total
Otimizado	Tempo	0,00	0,00	0,06	0,11	0,05	5,17	0,00	0,05	5,44
	%	0,00	0,00	1,10	2,02	0,91	95,04	0,00	0,91	100,00
Não otimizado	Tempo	0,00	0,05	0,17	9,66	0,33	8,46	0,60	3,24	22,51
	%	0,00	0,22	0,76	42,91	1,47	37,58	2,67	14,39	100,00

Tabela 10: Tempos de execução de cada bloco do código otimizado.

Este código final é bastante rápido, e de execução praticamente instantânea no computador utilizado.

Neste momento em que o código já foi quase que completamente otimizado, é interessante determinar a ordem de complexidade do algoritmo. Este resultado será determinado empiricamente, através dos tempos de execução do programa para alguns valores de J e N. Como serão utilizados canais de grandes dimensões, o valor adotado para Δx e Δt será de 0,01.

A Tabela 11 apresenta os tempos de execução para alguns valores de J e N, de forma que a razão N/J se mantenha igual a 0,8.

J	N	Bloco	4	5	6	Total
1000	800	Tempo	0,05	0,06	5,16	5,27
		%	0,95	1,14	97,91	100,00
2000	1600	Tempo	0,22	0,11	21,48	21,81
		%	1,01	0,50	98,49	100,00
4000	3200	Tempo	0,77	0,11	94,36	95,30
		%	0,81	0,12	99,01	100,00
8000	6400	Tempo	2,96	0,28	409,74	413,09
		%	0,72	0,07	99,19	100,00

Tabela 11: Tempos de execução do código para alguns valores de J e N.

Os resultados da Tabela 11 sugerem que o algoritmo possui uma complexidade bem próxima a quadrática ($O(n^2)$). A Figura 8 apresenta uma comparação entre os tempos de execução do algoritmo (linha cheia) e os tempos para um caso quadrático (linha pontilhada).

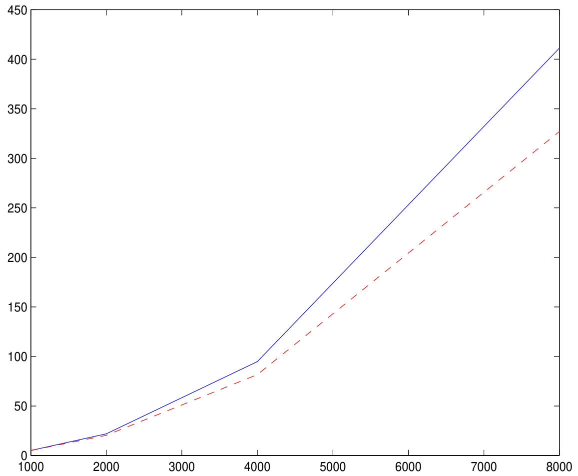

Figura 8: Tempo de execução em função do número de pontos na malha espacial.

4 Implementação da Malha Espacial Irregular

O código desenvolvido para o caso de uma profundidade constante, usando uma malha espacial regular, já apresenta um nível de eficiência satisfatório. É momento, então, de iniciar-se a discussão sobre os detalhes referentes à malha espacial irregular.

Como mencionado, a principal motivação para o uso de uma malha irregular é a possibilidade de adaptar-se a malha espacial às características do canal. Em outras palavras, a malha poderá ser mais refinada onde o canal requer uma representação mais detalhada e mais grosseira nos demais trechos.

Para fins de simplicidade, o perfil do canal apresentará ondulações senoidais. A escolha pelas ondulações senoidais se deve ao fato de que os resultados teóricos neste caso já são conhecidos [Gu92 e Na93]. Estas ondulações estarão presentes somente em um determinado trecho do canal, sendo este plano nos demais trechos. Nos trechos planos se usará um espaçamento Δx_1, e nos trechos sinuosos Δx_2.

O código `Onda08.m` representa este método com solo sinuoso e malha espacial irregular.

Neste momento da discussão, seria interessante realizar alguns testes para ratificar a precisão do método com a malha irregular.

O primeiro teste irá verificar se as ondas obtidas pelos métodos regular e irregular, com um fundo plano, são iguais (c.f. Figura 11). Obviamente, as condições iniciais devem ser as mesmas para ambos os casos.

Neste teste, deve-se comparar o perfil e a posição das ondas formadas no gráfico. A Figura 9 mostra os resultados obtidos para ambos os casos. Para o caso regular será usado $\Delta x_1 = \Delta x_2 = 0,01$. No caso irregular, $\Delta x_1 = 0,02$ e $\Delta x_2 = 0,01$.

Como era esperado, os dois casos forneceram resultados praticamente iguais. Para uma comparação mais precisa, na Figura 10 serão mostrados o perfil final da onda e o perfil inicial da onda, deslocado para a posição final. Este teste verifica que a onda permanece inalterada após atravessar o canal.

Na Figura 10, o perfil de onda em linha cheia é o obtido através do programa e o perfil de onda em linha tracejada é o perfil inicial da onda deslocado para a posição final. Como esperado a onda não mudou de forma, e também foi verificado que ela propagou com a velocidade unitária esperada.

O próximo teste irá verificar se as ondas obtidas pelos métodos regular e irregular, com um fundo irregular, são iguais. O modelo aqui testado é análogo ao anterior, mas agora o fundo apresenta três ondulações de amplitude igual a 0,4 no trecho entre 10 e 15.

Como era esperado, os dois casos forneceram resultados praticamente iguais. Para uma comparação mais precisa, na Figura 12 são mostrados o perfil final da onda e o perfil inicial da onda, deslocado para a posição final do caso do fundo

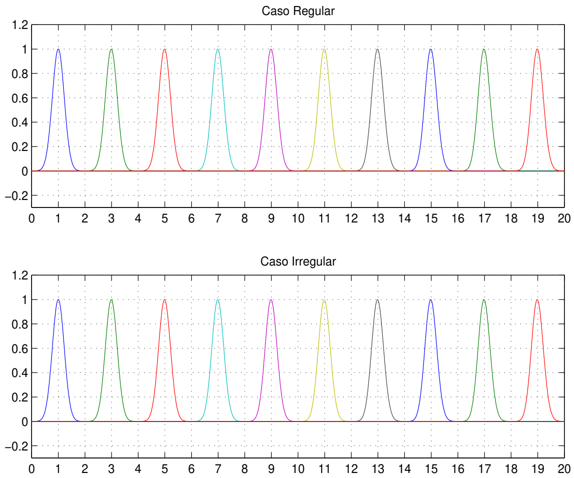

Figura 9: Comparação entre uma malha uniforme (regular) e uma malha irregular. Nos dois casos o fundo do canal é plano.

plano. Dessa forma, será possível verificar as alterações no perfil da onda e na velocidade de propagação da onda.

Pode-se notar que o perfil da onda foi alterado durante sua propagação no canal, pois houve a formação de ondas refletidas e transmitidas (Figura 11) e a amplitude do pulso principal foi levemente reduzida (Figura 12). Também, de acordo com as Figuras 9 e 11, pode-se notar que a velocidade de propagação foi ligeiramente reduzida no trecho sinuoso, fazendo com que o pulso principal se encontre um pouco recuado em relação ao caso plano (Figura 12). O estudo de ondas sobre topografias periódics pode ser encontrado em Guazzeli, Rey e Belzons [Gu92] e suas referências.

Em relação à eficiência do método, o tempo de execução do caso com malha espacial regular, com $J = 2000$, foi de 25 segundos. Para o caso irregular, com $J = 1250$, foi de apenas 15 segundos. Esta economia de tempo é bastante significativa.

Um último teste irá verificar o perfil da onda ao atravessar um canal com uma série de ondulações com um pequeno comprimento de onda. Caso a ordem da largura do pulso da onda seja superior à ordem do comprimento de onda das ondulações no solo, a onda parece não "sentir" a variação da profundidade do canal. Sabe-se que, nesta situação, os resultados obtidos são semelhantes ao caso em que uma onda atravessa um canal com um fundo efetivamente plano [Na93]. Segundo Nachbin, a onda não emite reflexões no trecho sinuoso e viaja com uma velocidade menor do que a do caso homogêneo (fundo plano).

A Figura 13 mostra o perfil da onda e o perfil do solo ondulado. Esta figura foi obtida com $\Delta x_1 = 0,02$, $\Delta x_2 = 0,01$, $\Delta t = 0,01$. O solo possui 80 oscilações

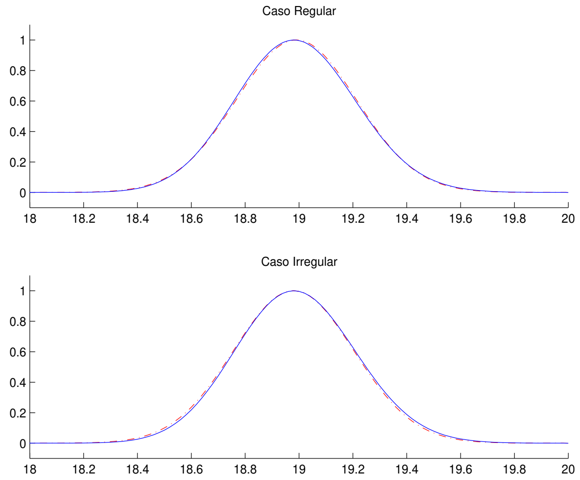

Figura 10: Detalhe da Figura 9.

de amplitude 0,5 entre 10 e 18. Tem-se ainda $J = 1400$ e $N = 1800$.

Uma análise mais cuidadosa da Figura 13 revela que a velocidade de propagação da onda no trecho ondulado é menor do que 1. O que ocorre é que a onda é afetada pelas rápidas oscilações no fundo do canal de forma sutil. Este detalhe é evidenciado pelo fato de que não há a formação de ondas refratadas durante o trecho ondulado. A onda sente apenas uma mudança na profundidade do canal ao entrar e depois ao sair do canal.

5 Conclusão

De fato, as técnicas empregadas reduziram drasticamente o tempo de execução do código. Enquanto a versão proposta inicialmente, com a inversão da matriz A, foi executada em aproximadamente 90 segundos (Tabela 1), a versão com a decomposição LU, código vetorizado, e matrizes esparsas permutadas foi executada em aproximadamente 2 segundos (Tabela 7).

Outra grande vantagem dessas técnicas é que elas são bastante gerais. A vetorização pode ser utilizada em qualquer problema em que seja necessário o uso de vetores ou matrizes. As técnicas de matrizes esparsas e aplicação de permutações podem ser empregadas em muitos problemas com este tipo de matrizes (muito comuns em problemas com equações diferenciais). A aplicação de uma malha irregular talvez não possa ser efetuada em tantos problemas, mas mesmo assim esta técnica é bastante geral.

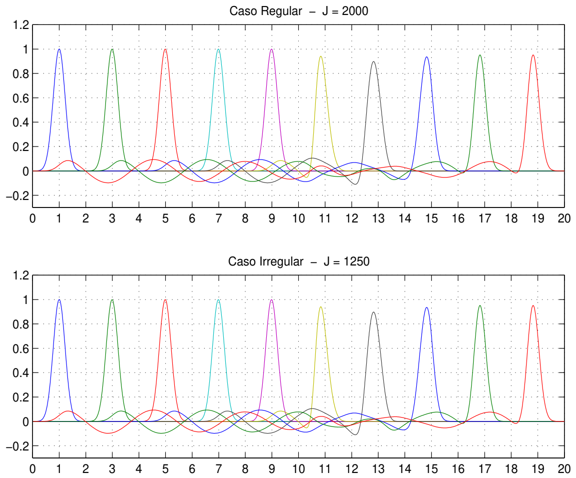

Figura 11: Experimentos computacionais na presença de uma topografia senoidal.

6 Referências Bibliográficas

[Ca89] B. Cathers, S. Bates, B. A. O'Connor, Internal Wave Reflections and Transmissions Arising from a Non-Uniform Mesh – Part I, *International Journal for Numerical Methods in Fluids*, vol. 9, pp. 783-810, 1989.

[Gu92] E. Guazzeli, V. Rey, M. Belzons, Higher Order Bragg Reflection of Gravity Surface Waves by Periodic Beds, *J. Fluid Mech.*, 245, pp. 301-317, 1992.

[Hi00] D. J. Higham, N. J. Higham, *MATLAB Guide*, SIAM, 2000.

[Me83] C. C. Mei, *The Applied Dynamics of Ocean Surface Waves*, John Wiley, 1983.

[Na93] A. Nachbin, Modelling of Water Waves in Shallow Channels, *Computational Mechanics Publications*, Sonthampton, UK, 1993.

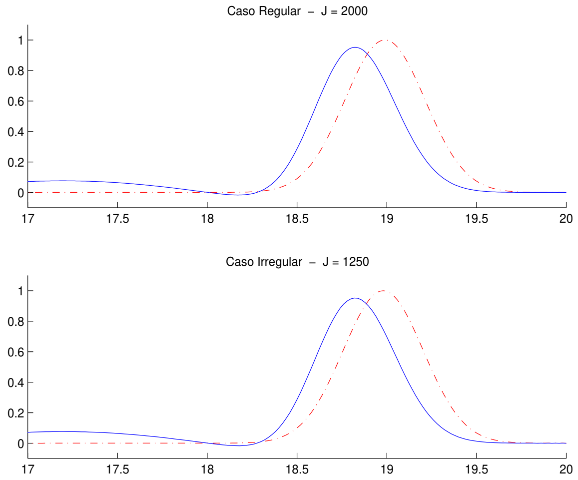

Figura 12: Detalhe da Figura 11.

Apêndice

Os códigos citados no texto se encontram neste apêndice. Os códigos `Onda01.m` e `Onda08.m` se encontram completos. No caso dos demais, somente os trechos (blocos) de interesse são apresentados.

Onda01.m

```
% Onda01.m
clear

% 1. Variaveis independentes
g = 1;          % aceleracao da gravidade
dx = 0.05;      % intervalo no espaco
dt = 0.05;      % intervalo no tempo
xf = 10;        % comprimento do canal - x final
tf = 8;         % tempo total de estudo - t final
a = 1;          % posicao inicial da onda
e = 0.1;        % calibra largura da onda
est = 10;       % quantos estagios no tempo serao impressos

% 2. Variaveis dependentes
J = round(xf/dx);               % numero de pontos no espaco
N = round(tf/dt);               % numero de estagios no tempo
passo = floor(N/(est-1));       % de quantos em quantos dt's imprimir
sol = zeros(2*J-4,N);           % matriz solucao
aux = zeros(J-2,est);           % matriz auxiliar

%3. Construcao dos vetores iniciais
```

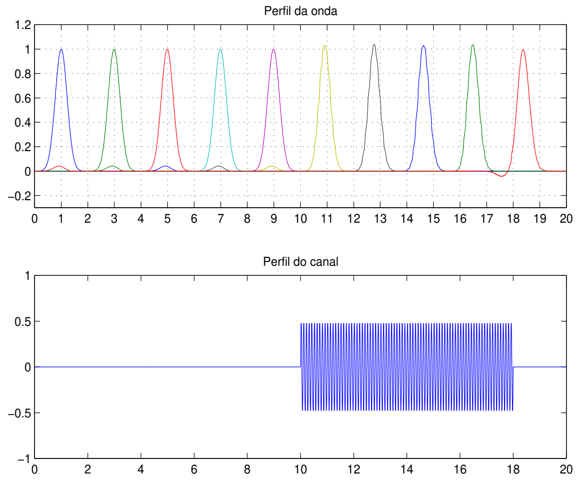

Figura 13: Experimento numérico no caso em que o perfil senoidal do solo varia rapidamente. Neste caso a malha deve ser fina na região da topografia para que esta seja representada adequadamente.

```
h = ones(1,J);          % profundidade do canal
for i = 1:J
   x(i) = dx*i;         % vetor espacial
end
for i = 1:J-2
   sol(i,1) = h(i+1)*exp(-(x(i+1)-a)\^{}2/e);    % profundidade x velocidade do fluido
   sol(i+J-2,1) = exp(-(x(i+1)-a)\^{}2/e);       % elevacao da onda
end
for i = 1:J-1
   m(i) = (x(i+1)-x(i))/(6*dt);
end

%4. Construcao das matrizes A e B
M1 = diag(m(2:J-2),-1) + diag(2*m(1:J-2)+2*m(2:J-1),0) + diag(m(2:J-2),1);
M2 = diag(-g/4*h(3:J-1),-1) + diag(g/4*h(2:J-2),1);
M3 = diag(-1/4*ones(1,J-3),-1) + diag(1/4*ones(1,J-3),1);
A = [M1, M2; M3,M1];
B = [M1,-M2;-M3,M1];

% 5. Inversao da matriz A
C = inv(A)*B;

% 6. Progressao no tempo
for i = 2:N
   sol(:,i) = C*sol(:,i-1);
end
```

```
% 7. Preparacao dos resultados
j = 1;
k = 1;
aux(:,k) = sol(J-1:2*J-4,1);
for i = 2:N
   j = j + 1;
   if j == passo
      k = k + 1;
      aux(:,k) = sol(J-1:2*J-4,i);
      j = 0;
      if k >= est
         break
      end
   end
end

% 8. Impressao dos resultados
figure('name',['J = ',int2str(J),'  N = ',int2str(N),'  dx = ',num2str(dx),'
        dt = ',num2str(dt)],'numbertitle','off')  %deve ser digitado como
                                                  %CONTINUACAO da linha acima
plot(x,[zeros(1,est);aux;zeros(1,est)])
title('Perfil da onda')
ylim([-0.3,1.2])
```

Onda02.m

```
% Onda02.m
clear
t = zeros(1,9);     % tempo de execucao de cada bloco do codigo

% 1. Variaveis independentes
tic
g = 1;            % aceleracao da gravidade
dx = 0.02;        % intervalo no espaco
dt = 0.02;        % intervalo no tempo
xf = 10;          % comprimento do canal - x final
tf = 8;           % tempo total de estudo - t final
a = 1;            % posicao inicial da onda
e = 0.1;          % calibra largura da onda
est = 10;         % quantos estagios no tempo serao impressos
t(1) = toc;

% 8. Impressao dos resultados
tic
figure('name',['J = ',int2str(J),'  N = ',int2str(N),'  dx = ',num2str(dx),'
      dt = ',num2str(dt)],'numbertitle','off')   %deve ser digitado como
                                                  %CONTINUACAO da linha acima
```

```
plot(x,[zeros(1,est);aux;zeros(1,est)])
title('Perfil da onda')
ylim([-0.3,1.2])
t(8) = toc;
t(9) = sum(t)
```

Os comandos `tic` e `toc` devem ser acrescentados nos demais blocos de maneira semelhante.

Onda03.m

```
% 5. Decomposicao da matriz A
tic
[L,U,P] = lu(A);
PB = P*B;
t(5) = toc;

% 6. Progressao no tempo
tic
r = zeros(1,2*J-4);
for i = 2:N
   D = PB*sol(:,i-1);
   r(1) = D(1)/L(1,1);
   for j = 2:1:2*J-4                     % percorre as linhas de cima para baixo
      soma = 0;
      for k = 1:1:j-1                    % percorre as colunas
         soma = soma + L(j,k)*r(k);
      end
      r(j) = (D(j)-soma)/L(j,j);
   end
   sol(2*J-4,i) = r(2*J-4)/U(2*J-4,2*J-4);
   for j = 2*J-4-1:-1:1                  % percorre as linhas de baixo para cima
      soma = 0;
      for k = j+1:1:2*J-4                % percorre as colunas
         soma = soma + U(j,k)*sol(k,i);
      end
      sol(j,i) = (r(j)-soma)/U(j,j);
   end
end
t(6) = toc;
```

Onda04.m

```
% 4. Construcao das matrizes A e B
tic
M1 = diag(m(2:J-2),-1) + diag(2*m(1:J-2)+2*m(2:J-1),0) + diag(m(2:J-2),1);
M2 = diag(-g/4*h(3:J-1),-1) + diag(g/4*h(2:J-2),1);
```

```
M3 = diag(-1/4*ones(1,J-3),-1) + diag(1/4*ones(1,J-3),1);
A = [M1, M2; M3,M1];
B = [M1,-M2;-M3,M1];
%A = sparse(A);
%B = sparse(B);
t(4) = toc;

% 6. Progressao no tempo
tic
for i = 2:N
   D = PB*sol(:,i-1);
   sol(:,i) = U\(L\D);
end
t(6) = toc;
```

Onda05.m

```
% 5. Decomposicao da matriz A
tic
pp = zeros(1,2*J-4);            % vetor permutacao
pp(1:2:2*J-5) = 1:J-2;
pp(2:2:2*J-4) = J-1:2*J-4;
[L,U,P] = lu(A(:,pp));
PB = P*B(:,pp);
sol(:,1) = sol(pp,1);
t(5) = toc;

% 7. Preparacao dos resultados
tic
pp_inv(pp) = 1:2*J-4;
sol = sol(pp_inv,:);
j = 1;
k = 1;
aux(:,k) = sol(J-1:2*J-4,1);
for i = 2:N
   j = j + 1;
   if j == passo
      k = k + 1;
      aux(:,k) = sol(J-1:2*J-4,i);
      j = 0;
      if k >= est
         break
      end
   end
end
t(7) = toc;
```

Onda06.m

```
% 5. Decomposicao da matriz A
tic
%pp = colmmd(A);
%pp = symmmd(A);
pp = symrcm(A);
[L,U,P] = lu(A(:,pp));
PB = P*B(:,pp);
sol(:,1) = sol(pp,1);
t(5) = toc;
```

Onda07.m

```
% 3. Construcao dos vetores iniciais
tic
h = ones(1,J);          % profundidade do canal
x(1:J) = dx*(1:J);      % vetor espacial
sol(J-1:2*J-4,1) = exp(-((x(2:J-1))'-a).^2/e);     % elevacao da onda
sol(1:J-2,1) = sol(J-1:2*J-4,1).*(h(2:J-1))';      % velocidade do fluido x profundidade
m(1:J-1) = (x(2:J)-x(1:J-1))/(6*dt);
t(3) = toc;

% 4. Construcao das matrizes A e B
tic
M1 = sparse([2:J-2,1:J-2,1:J-3],[1:J-3,1:J-2,2:J-2],[m(2:J-2),
     2*(m(1:J-2)+m(2:J-1)),m(2:J-2)],J-2,J-2);   %deve ser digitado como
                                                 %CONTINUACAO da linha acima
M2 = sparse([2:J-2,1:J-3],[1:J-3,2:J-2],[-g/4*h(3:J-1),g/4*h(2:J-2)],J-2,J-2);
M3 = sparse([2:J-2,1:J-3],[1:J-3,2:J-2],[repmat(-1/4,[1,J-3]),
     repmat(1/4,[1,J-3])],J-2,J-2);   %deve ser digitado como
                                      %CONTINUACAO da linha acima
A = [M1, M2; M3,M1];
B = [M1,-M2;-M3,M1];
t(4) = toc;

% 6. Progressao no tempo
tic
j = 1;
k = 1;
aux1(:,k) = sol(:,1);
for i = 2:N
   D = PB*sol(:,1);
   sol(:,2) = U\(L\D);
   j = j + 1;
   if j == passo
      k = k + 1;
      aux1(:,k) = sol(:,2);
      j = 0;
      if k >= est
```

```
         break
      end
   end
   sol(:,1) = sol(:,2);
end
t(6) = toc;

% 7. Preparacao dos resultados
tic
pp_inv(pp) = 1:2*J-4;
aux1 = aux1(pp_inv,:);
aux2(2:J-1,:) = aux1(J-1:2*J-4,:);
t(7) = toc;
```

Onda08.m

```
% Onda08.m
clear
t = zeros(1,9);     % tempo de execucao de cada bloco do codigo

% 1. Variaveis independentes
tic
g = 1;              % aceleracao da gravidade
dx1 = 0.02;         % intervalo no espaco para malha grossa
dx2 = 0.01;         % intervalo no espaco para malha fina
dt = 0.01;          % intervalo no tempo
xf = 20;            % comprimento do canal - x final
tf = 18;            % tempo total de estudo - t final
a = 1;              % posicao inicial da onda
b = 10;             % posicao inicio das oscilacoes no solo
c = 18;             % posicao final das oscilacoes no solo
e = 0.1;            % calibra largura da onda
amp = 0.5;          % amplitude das oscilacoes no solo
no = 80;            % numero de oscilacoes
est = 10;           % quantos estagios no tempo serao impressos
t(1) = toc;

% 2. Variaveis dependentes
tic
J = round((xf-(c-b))/dx1+(c-b)/dx2);       % numero de pontos no espaco
N = round(tf/dt);               % numero de estagios no tempo
passo = floor(N/(est-1));       % de quantos em quantos dt's imprimir
sol = zeros(2*J-4,2);           % matriz solucao
aux1 = zeros(2*J-4,est);
aux2 = zeros(J,est);            % matrizes auxiliares
j1 = round(b/dx1);              % numero de pontos no primeiro trecho
j2 = round((c-b)/dx2);          % numero de pontos no segundo trecho
t(2) = toc;
```

```
% 3. Construcao dos vetores iniciais
tic
x(1) = dx1;             % vetor espacial
for i = 2:j1,
   x(i) = x(i-1) + dx1;
end
for i = j1+1:j1+j2,
   x(i) = x(i-1) + dx2;
end
for i = j1+j2+1:J,
   x(i) = x(i-1) + dx1;
end
h = ones(1,J);          % profundidade do canal
h(j1+1:j1+j2) = h(j1+1:j1+j2) - amp*sin(2*pi*no/(c-b)*(x(j1+1:j1+j2)+j1));
sol(J-1:2*J-4,1) = exp(-((x(2:J-1))'-a).^2/e);     % elevacao da onda
sol(1:J-2,1) = sol(J-1:2*J-4,1).*(h(2:J-1))';      % velocidade do fluido x profundidade
m(1:J-1) = (x(2:J)-x(1:J-1))/(6*dt);
t(3) = toc;

% 4. Construcao das matrizes A e B
tic
M1 = sparse([2:J-2,1:J-2,1:J-3],[1:J-3,1:J-2,2:J-2],[m(2:J-2),
     2*(m(1:J-2)+m(2:J-1)),m(2:J-2)],J-2,J-2);  %deve ser digitado como
                                                %CONTINUACAO da linha acima
M2 = sparse([2:J-2,1:J-3],[1:J-3,2:J-2],[-g/4*h(3:J-1),g/4*h(2:J-2)],J-2,J-2);
M3 = sparse([2:J-2,1:J-3],[1:J-3,2:J-2],[repmat(-1/4,[1,J-3]),
     repmat(1/4,[1,J-3])],J-2,J-2);  %deve ser digitado como
                                     %CONTINUACAO da linha acima
A = [M1, M2; M3,M1];
B = [M1,-M2;-M3,M1];
t(4) = toc;

% 5. Decomposicao da matriz A
tic
pp = symrcm(A);
[L,U,P] = lu(A(:,pp));
PB = P*B(:,pp);
sol(:,1) = sol(pp,1);
t(5) = toc;

% 6. Progressao no tempo
tic
j = 1;
k = 1;
aux1(:,k) = sol(:,1);
for i = 2:N
   D = PB*sol(:,1);
```

```
   sol(:,2) = U\(L\D);
   j = j + 1;
   if j == passo
      k = k + 1;
      aux1(:,k) = sol(:,2);
      j = 0;
      if k >= est
         break
      end
   end
   sol(:,1) = sol(:,2);
end
t(6) = toc;

% 7. Preparacao dos resultados
tic
pp_inv(pp) = 1:2*J-4;
aux1 = aux1(pp_inv,:);
aux2(2:J-1,:) = aux1(J-1:2*J-4,:);
t(7) = toc;

% 8. Impressao dos resultados
tic
figure('name',['J = ',int2str(J),'  N = ',int2str(N),'  dx1 = ',num2str(dx1),'
      dx2 = ',num2str(dx2),'  dt = ',num2str(dt)],'numbertitle','off')
                                                    %deve ser digitado como
                                                    %CONTINUACAO da linha acima

subplot(2,1,1)
plot(x,aux2)
title('Perfil da onda')
xlim([0,xf])
ylim([-0.3,1.2])
set(gca,'XTick',0:1:xf)
grid on
subplot(2,1,2)
plot(x,1-h)
title('Perfil do canal')
xlim([0,xf])
ylim([-1,1])
t(8) = toc;
t(9) = sum(t)
```