Um método adaptativo de diferenças finitas utilizando wavelets e sua aplicação na resolução numérica de um modelo transiente de micropropulsores a hidrazina

Autor: Rodrigo Morante Blanco
Orientadora: Profa. Dra. Sônia Maria Gomes

Departamento de Matemática Aplicada
IMECC–UNICAMP

Trabalho financiado pela FAPESP, processo 00/00317-4

2002

Para Ailín

Agradecimentos

À minha esposa, Ailín. Sem seu amor, sua ajuda, sua dedicação, certamente não estaríeis lendo este trabalho.

À minha família. A distância não existe em certos casos. Afortunadamente.

Desejo agradecer de forma muito especial à Dra. Margarita Suárez, Professora Emérita da Faculdade de Matemática da Universidade de Havana. Foi ela quem guiou meus primeiros passos no mundo fascinante da matemática numérica.

À Dra. Sônia M. Gomes por me convidar a trabalhar neste tema, e pelo apoio mostrado.

À Dra. Cristina Cunha, à Dra. Véra Lopes e à Dra. Márcia Ruggiero, por seus valiosos comentários e aulas.

À Cidinha, à Fátima, à Tânia e ao Ednaldo.

Às minhas amigas e aos meus amigos, em especial Lucelina, Luziane, Lilliam & Ricardo, Irene, Valéria.

À Fundação de Amparo à Pesquisa do Estado de São Paulo, pelo auxílio financeiro (processo 00/00317-4).

À Banca Examinadora, formada pela Dra. Sônia M. Gomes, a Dra. Magda Kimico Kaibara, a Dra. Véra Lopes e o Dr. Petrônio Pulino.

É claro que deixo de mencionar muitas pessoas, o que não significa que as tenha esquecido. Muito obrigado a todos vocês.

Resumo

É utilizado um esquema adaptativo para a resolução numérica de um sistema de equações em derivadas parciais, em particular, as equações que modelam a decomposição catalítica da hidrazina num micropropulsor. É feita uma revisão da análise multirresolução e da decomposição wavelet, ferramentas eficazes na implementação de um esquema adaptativo que combinado com um esquema de diferenças finitas resolve o problema com um custo computacional reduzido em termos de armazenagem de dados e rapidez. Os resultados deste método adaptativo são comparados com os de um método clássico, também implementado. São apresentados todos os programas utilizados, desenvolvidos pelo autor, devidamente comentados.

Abstract

An adaptive scheme is used in the numerical resolution of a system of partial differential equations modelling the catalytic decomposition of hydrazine in a thruster. Multirresolution analysis and wavelet decomposition are examined, which prove effective tools when implementing an adaptive scheme that, combined with a finite differences scheme solves the problem with reduced computational cost in terms of data storage and speed. The results of this adaptive method are compared with those of a classical method, which is also implemented. All the programs, which were written by the author, are presented and due commented.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Na resolução numérica de equações diferenciais parciais é comum o uso de esquemas de diferenças finitas em malhas regulares. Estes métodos têm sido amplamente utilizados e continuarão a sê-lo, devido aos bons resultados, em geral, e sua facilidade de programação. Porém, em casos freqüentes, em que as soluções apresentam mudanças bruscas de comportamento, o uso de uma malha regular não é o mais conveniente. Se for usada uma malha refinada capaz de descrever com precisão as funções em zonas de mudanças bruscas, também nas zonas de comportamento suave serão utilizados muitos pontos, o que, além de não ser necessário, vai fazer com que o custo computacional seja excessivamente alto. Se for usada uma malha pouco refinada, o custo computacional diminuirá, assim como a qualidade dos resultados, devido à pobre representação da função próximo das irregularidades. Por esta razão, vários esquemas adaptativos têm sido propostos.

Neste trabalho considera-se o esquema adaptativo proposto em [9] para a solução numérica de equações que envolvem variação temporal. Em cada passo de tempo, a malha espacial é irregular e escolhida dinamicamente: terá mais pontos naquelas zonas onde o comportamento da solução seja brusco, e menos onde seja suave. Para a detecção de tais regiões, utilizam-se coeficientes wavelet como indicadores de regularidade local. O esquema é aplicado na simulação numérica de um sistema de equações que modela o comportamento de um propulsor para controle de atitude de satélites.

Esta dissertação está organizada da seguinte maneira:

No Capítulo 2 são descritos os principais resultados da teoria wavelet estudados durante a realização deste trabalho.

No Capítulo 3 é formulado o problema que propomos resolver: a distri-

buição de temperatura e das frações de massa de gases num micropropulsor a hidrazina. Para resolver numericamente este problema, são utilizados dois esquemas clássicos, um implícito e outro explícito, para gerarem soluções de referência. O esquema adaptativo é uma combinação do esquema explícito de diferenças finitas com as ferramentas desenvolvidas no Capítulo 2, visando a redução do custo computacional. A qualidade das soluções obtidas mediante o método adaptativo é estabelecida ao compará-las com as soluções de referência.

No Apêndice A apresentam-se duas ferramentas (esquemas ENO e *lifting*) que foram estudadas e implementadas, mas que ainda não foram utilizadas no método adaptativo. Porém, devido às suas potencialidades, acreditamos que possam ser úteis em trabalhos futuros.

No Apêndice B estão todos os códigos utilizados no trabalho, criados pelo autor em MATLAB e devidamente descritos.

No Apêndice C apresenta-se o valor de cada parâmetro e de cada magnitude física envolvidos no modelo que descreve a decomposição catalítica da hidrazina num micropropulsor.

Capítulo 2

Wavelets e esquemas de multirresolução

2.1 Análise multirresolução

Considera-se o espaço de Hilbert

$$\mathcal{L}^2(\mathbb{R}) = \left\{ f : \mathbb{R} \to \mathbb{C} \text{ tais que } \int_{\mathbb{R}} |f(x)|^2 \ dx < \infty \right\},$$

munido do produto interno

$$\langle f(x), g(x) \rangle = \int_{\mathbb{R}} f(x)\, \overline{g(x)}\ dx.$$

A definição de uma análise multirresolução introduzida por Mallat em 1989 e citada por Meyer [10] é a seguinte:

Definição 2.1.1. *A seqüência* $\{V_j\}_{j\in\mathbb{Z}}$ *de subespaços de* $\mathcal{L}^2(\mathbb{R})$ *forma uma* análise multirresolução *se:*

1. $\cdots \subset V_{-1} \subset V_0 \subset V_1 \subset \cdots$.
2. $\overline{\bigcup_{j\in\mathbb{Z}} V_j} = \mathcal{L}^2(\mathbb{R})$.
3. $\bigcap_{j\in\mathbb{Z}} V_j = \{0\}$.

4. *Existe uma relação de escala diádica entre os subespaços,*

$$f(x) \in V_0 \iff f(2x) \in V_1 \iff f(2^j x) \in V_j.$$

5. *Existe uma função $\varphi \in V_0$, chamada* função de escala*, tal que o conjunto*

$$\{\varphi_{0,k} : \varphi_{0,k}(x) = \varphi(x-k),\, k \in \mathbb{Z}\}$$

forma uma base de Riesz de V_0. Ou seja, é um conjunto de funções linearmente independentes que gera o espaço V_0 e existem constantes C_1 e C_2 tais que para toda seqüência finita $\{c_k\}$:

$$C_1 \sum_k |c_k|^2 \leq \left\| \sum_k c_k \varphi(x-k) \right\|_{\mathcal{L}^2}^2 \leq C_2 \sum_k |c_k|^2 . \tag{2.1}$$

Isto implica que se a série dos coeficientes é de quadrado somável, então a série de funções converge em $\mathcal{L}^2(\mathbb{R})$ e vice-versa.

Como conseqüência da definição de uma análise multirresolução, resulta que o conjunto $\{\varphi_{j,k} : \varphi_{j,k}(x) = \varphi(2^j x - k),\, k \in \mathbb{Z}\}$ forma uma base de Riesz de V_j. Desta forma, podemos escrever qualquer função $f \in V_j$ como

$$f(x) = \sum_{k \in \mathbb{Z}} c_k \varphi_{j,k}(x). \tag{2.2}$$

Por último, tendo em vista a inclusão $V_0 \subset V_1$, é válida uma relação do tipo

$$\varphi(x) = \sum_{l \in \mathbb{Z}} h_l \, \varphi(2x - l). \tag{2.3}$$

Os escalares h_l são chamados de *coeficientes do filtro de escala*.

Definição 2.1.2. *Uma função de escala contínua φ é denominada* interpoladora *se*

$$\varphi(k) = \delta_{0,k} = \begin{cases} 1, & k = 0, \\ 0, & k \neq 0, \end{cases}$$

para todo $k \in \mathbb{Z}$.

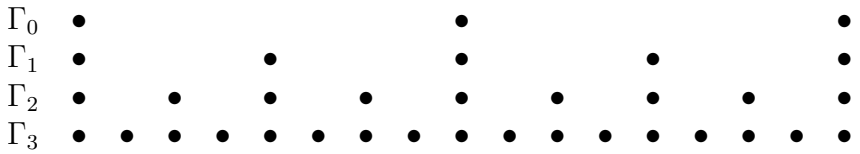


Figura 2.1: As malhas diádicas são construídas a partir de $\Gamma_0 = \mathbb{Z}$ acrescentando, a cada novo nível, os pontos intermediários entre dois pontos consecutivos do nível anterior.

Neste caso, toda $f \in V_j$ satisfaz

$$f(x) = \sum_{k\in\mathbb{Z}} f(2^{-j}k)\varphi_{j,k}(x). \tag{2.4}$$

De fato, se consideramos $x = 2^{-j}k$ em (2.2) temos

$$f(2^{-j}k) = \sum_{l\in\mathbb{Z}} c_l\varphi_{j,l}(2^{-j}k) = \sum_{l\in\mathbb{Z}} c_l\varphi(2^j2^{-j}k - l) = \sum_{l\in\mathbb{Z}} c_l\delta_{0,k-l} = c_k.$$

Ou seja, os coeficientes são simples avaliações da função nos pontos da forma $x = 2^{-j}k$, portanto para calculá-los não é preciso realizar nenhuma operação complexa (como integração), o que representa uma grande vantagem com respeito a outras bases.

Definimos a malha diádica Γ_j como sendo o conjunto

$$\Gamma_j = \{x_{j,k} \in \mathbb{R} : x_{j,k} = 2^{-j}k,\ k \in \mathbb{Z}\},\ j \in \mathbb{Z}; \tag{2.5}$$

veja a Figura 2.1. Mediante (2.4), estabelecemos uma identificação entre os espaços físico Γ_j e funcional V_j, dado que a coordenada de f que corresponde a $\varphi_{j,k} \in V_j$ é exatamente a avaliação de f no ponto $x_{j,k} \in \Gamma_j$. Esta identificação será útil em nossos algoritmos.

Podemos estender (2.4) para qualquer função contínua de $\mathcal{L}^2(\mathbb{R})$ considerando o operador

$$P_jf = \sum_{k\in\mathbb{Z}} f(2^{-j}k)\varphi_{j,k}, \tag{2.6}$$

em que $P_jf = f$ se $f \in V_j$ e portanto P_j é um operador de projeção. Os valores $f_{j,k} = f(2^{-j}k)$ estão associados aos pontos $x_{j,k}$ da malha diádica.

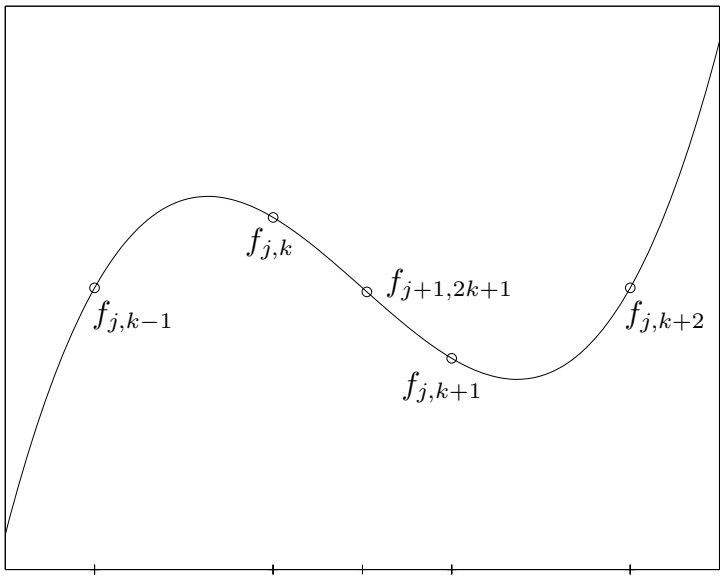


Figura 2.2: Interpolação de grau 3.

2.2 Esquema de subdivisão interpolador

A definição da função de escala interpoladora φ é baseada num esquema de subdivisão interpolador [9]. A regra básica é a interpolação polinomial de grau $p-1$.

Começando com $f_{0,k}$, $k \in \mathbb{Z}$, o algoritmo vai gerar uma função f_{j+1} definida em Γ_{j+1}, fazendo uma interpolação de grau $p-1$ em cada novo ponto $x_{j+1,2k+1}$, a partir dos valores f_j do nível anterior. Empregamos os p pontos vizinhos mais próximos em Γ_j. Se p é um número par, a interpolação é simétrica (Figura 2.2).

Formalmente,

$$f_{j+1} = \begin{cases} f_{j+1,2k} & = f_{j,k}, \\ f_{j+1,2k+1} & = \boldsymbol{p}_{j+1,2k+1}(x_{j+1,2k+1}), \end{cases}$$

em que o polinômio de interpolação $\boldsymbol{p}_{j+1,2k+1}(x)$ é

$$\sum_{l=-\frac{p}{2}+1}^{\frac{p}{2}} f_{j,k+l} \frac{\prod_{h=-\frac{p}{2}+1}^{l-1}(x - x_{j,k+h}) \prod_{h=l+1}^{\frac{p}{2}}(x - x_{j,k+h})}{\prod_{h=-\frac{p}{2}+1}^{l-1}(x_{j,k+l} - x_{j,k+h}) \prod_{h=l+1}^{\frac{p}{2}}(x_{j,k+l} - x_{j,k+h})}. \qquad (2.7)$$

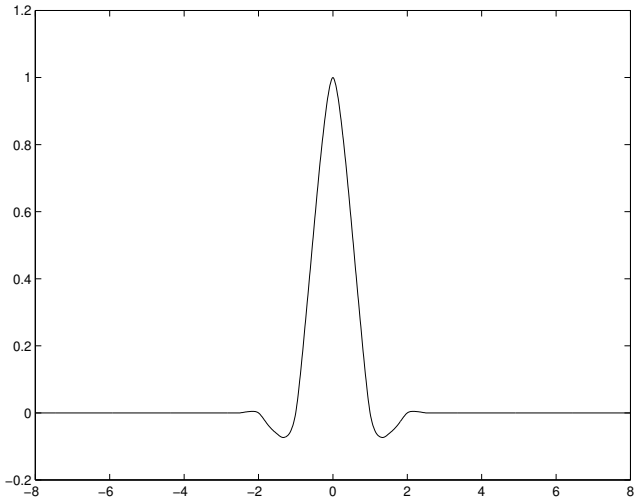


Figura 2.3: Função de escala de Deslauriers e Dubuc, $p = 4$.

No caso $p = 4$ o resultado do esquema no ponto $x_{j+1,2k+1} = 2^{-(j+1)}(2k+1)$ é

$$\boldsymbol{p}_{j+1,2k+1}(x_{j+1,2k+1}) = -\frac{1}{16}\left(f_{j,k-1} + f_{j,k+2}\right) + \frac{9}{16}\left(f_{j,k} + f_{j,k+1}\right). \qquad (2.8)$$

2.3 Função de escala de Deslauriers e Dubuc

A aplicação do esquema de subdivisão interpolador simétrico (p par) a partir da seqüência $\delta_{0,k}$, $k \in \mathbb{Z}$ dá origem à função de escala de Deslauriers e Dubuc (Figura 2.3). Suas principais características são:

1. φ é interpoladora, ou seja, $\varphi(k) = \delta_{0,k}$ para todo $k \in \mathbb{Z}$.
2. φ é contínua e possui suporte compacto no intervalo $[-p+1, p-1]$.
3. φ satisfaz a equação de escala:

$$\varphi(x) = \sum_l \varphi(l/2)\varphi(2x - l). \qquad (2.9)$$

No caso da interpolação cúbica ($p = 4$) os coeficientes $h_l = \varphi(l/2)$ não-nulos são dados na tabela a seguir:

$l/2$	$-3/2$	$-1/2$	0	$1/2$	$3/2$
$\varphi(l/2)$	$-1/16$	$9/16$	1	$9/16$	$-1/16$

4. φ é par, ou seja $\varphi(x) = \varphi(-x)$, pois o esquema interpolador é simétrico.

5. Todo polinômio $q(x)$ de grau menor ou igual a $p-1$ pode ser escrito como combinação linear de $\varphi_{j,k}$, $j, k \in \mathbb{Z}$:

$$q(x) = \sum_k q(2^{-j}k)\varphi_{j,k}(x).$$

2.4 Wavelets interpoladoras—aspecto funcional

Definimos o operador de interpolação $P_j f \in V_j$ de uma função contínua f de $\mathcal{L}^2(\mathbb{R})$ como sua projeção em V_j (2.6):

$$P_j f = \sum_{k\in\mathbb{Z}} f(2^{-j}k)\varphi_{j,k} = \sum_{k\in\mathbb{Z}} f_{j,k}\varphi_{j,k}.$$

Também temos

$$P_{j+1} f = \sum_{k\in\mathbb{Z}} f(2^{-j-1}k)\varphi_{j+1,k} = \sum_{k\in\mathbb{Z}} f_{j+1,k}\varphi_{j+1,k}.$$

Como $P_{j+1}f - P_j f \in V_{j+1}$ existem coeficientes $c_{j+1,k}$ tais que

$$P_{j+1} f - P_j f = \sum_{k\in\mathbb{Z}} c_{j+1,k}\varphi_{j+1,k}. \tag{2.10}$$

Acontece que $c_{j+1,2k} = 0$ porque

$$\begin{aligned} c_{j+1,2k} &= \sum_{l\in\mathbb{Z}} c_{j+1,l}\varphi_{j+1,l}(x_{j+1,2k}), \\ &= P_{j+1}f(x_{j+1,2k}) - P_j f(x_{j+1,2k}), \\ &= P_{j+1}f(x_{j+1,2k}) - P_j f(x_{j,k}), \\ &= f(2^{-j}k) - f(2^{-j}k), \\ &= 0. \end{aligned}$$

Portanto, podemos escrever (2.10) como

$$P_{j+1}f - P_jf = \sum_{k\in\mathbb{Z}} c_{j+1,2k+1}\varphi_{j+1,2k+1}, \tag{2.11}$$

onde só aparecem as funções correspondentes aos pontos de $\Gamma_{j+1} - \Gamma_j$. Definamos $\psi(x) = \varphi(2x-1)$ como nossa *função wavelet*. Então $\varphi_{j+1,2k+1} = \psi_{j,k}$ e os coeficientes $d_{j,k} = c_{j+1,2k+1}$ são chamados de *detalhes*. Temos

$$P_{j+1}f(x) - P_jf(x) = \sum_{k\in\mathbb{Z}} d_{j,k}\psi_{j,k}(x),$$

ou seja,

$$\begin{aligned} P_{j+1}f(x) &= P_jf(x) + \sum_{k\in\mathbb{Z}} d_{j,k}\psi_{j,k}(x), \\ \sum_{k\in\mathbb{Z}} f_{j+1,k}\varphi_{j+1,k}(x) &= \sum_{k\in\mathbb{Z}} f_{j,k}\varphi_{j,k}(x) + \sum_{k\in\mathbb{Z}} d_{j,k}\psi_{j,k}(x). \end{aligned} \tag{2.12}$$

A relação (2.12) é chamada de *decomposição wavelet*, onde a projeção de f na malha mais fina Γ_{j+1} é decomposta como a projeção de f na malha mais grossa Γ_j mais um termo de correção que depende dos detalhes. Seja $W_j \subset V_{j+1}$ o subespaço gerado pelas funções $\psi_{j,k}$; então (2.12) significa que $V_{j+1} = V_j \oplus W_j$, veja a Figura 2.4.

Avaliando (2.11) em $x_{j+1,2k+1}$,

$$d_{j,k} = f_{j+1,2k+1} - \sum_{l\in\mathbb{Z}} f_{j,l}\varphi_{j,l}(x_{j+1,2k+1}).$$

Finalmente, tendo em conta que $h_l = \varphi(l/2)$,

$$d_{j,k} = f_{j+1,2k+1} - \sum_{l\in\mathbb{Z}} f_{j,l}h_{2(k-l)+1}. \tag{2.13}$$

Isto significa que os coeficientes $d_{j,k}$ são a diferença entre o valor da função no ponto $x_{j+1,2k+1}$ e o valor interpolado. Ou seja, $d_{j,k}$ é o erro na interpolação no ponto $x_{j+1,2k+1}$ a partir dos valores nos p pontos vizinhos em Γ_j (Figura 2.5). Note-se que os pesos nessa interpolação são os coeficientes do filtro de escala de índices ímpares que por sua vez são os pesos da interpolação do esquema de subdivisão interpolador. Portanto, fixado p par, sempre fazemos uso do mesmo esquema de interpolação.

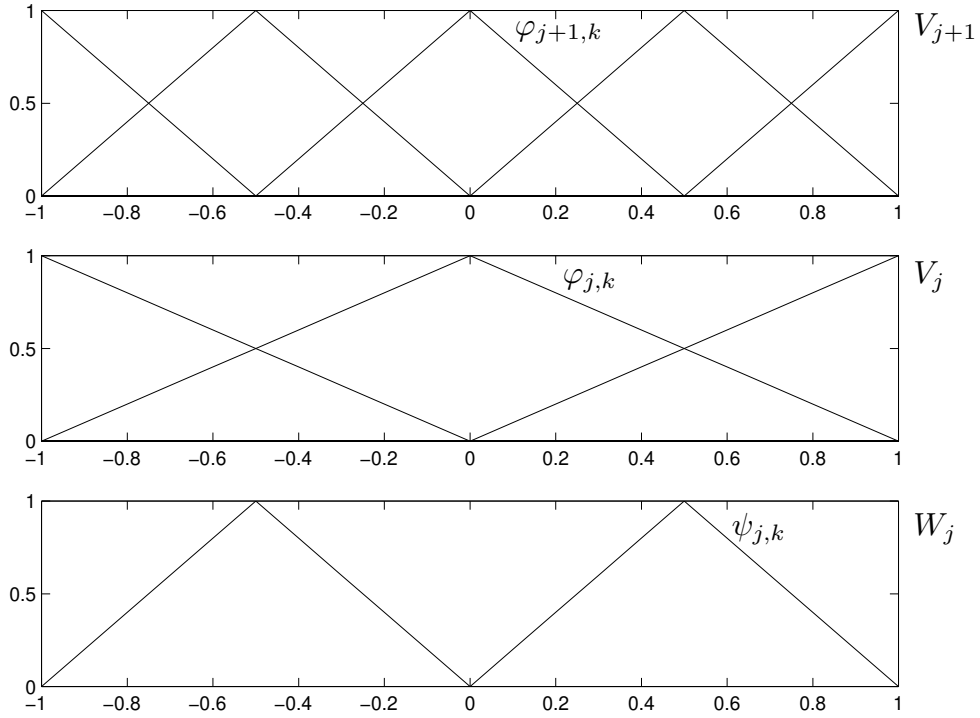


Figura 2.4: Funções da base de V_{j+1}, V_j e W_j, respectivamente, com $j = 0$ e $p = 2$ (caso linear).

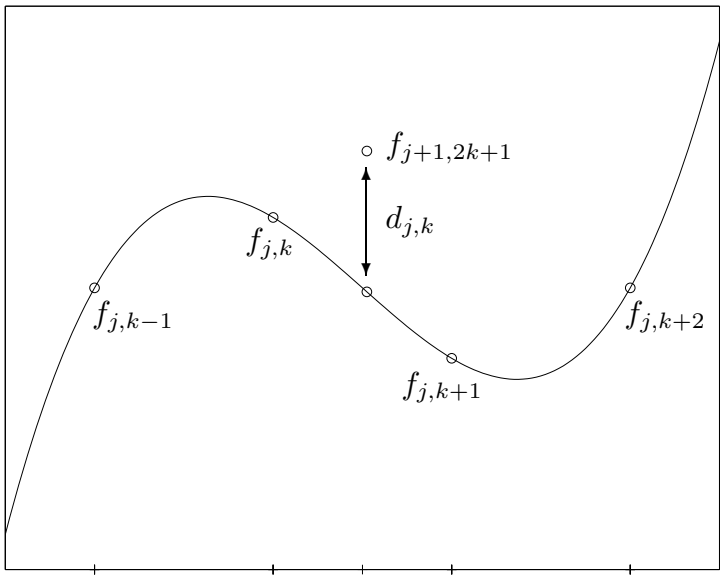


Figura 2.5: O coeficiente $d_{j,k}$ é o erro da interpolação da função em $x_{j+1,2k+1}$ utilizando seu valor nos pontos vizinhos de V_j; $p = 4$.

A *representação wavelet* é obtida aplicando repetidamente a fórmula (2.12):

$$\sum_{k\in\mathbb{Z}} f_{j+1,k}\varphi_{j+1,k} = \sum_{k\in\mathbb{Z}} f_{j_0,k}\varphi_{j_0,k} + \sum_{l=j_0}^{j}\sum_{k\in\mathbb{Z}} d_{l,k}\psi_{l,k}. \tag{2.14}$$

A projeção de f na malha mais fina Γ_{j+1} é decomposta como a projeção de f na malha mais grossa Γ_{j_0} mais os detalhes nos níveis intermediários. Novamente existe uma correspondência biunívoca entre os pontos $x_{j+1,2k+1} \in \Gamma_{j+1} - \Gamma_j$ e os coeficientes $d_{j,k}$, Figura 2.6.

2.5 Aspecto discreto

Na seção 2.4 vimos o aspecto funcional das wavelets interpoladoras definindo o operador de interpolação P_j e obtivemos a representação (2.12) em V_{j+1} de uma função f. A partir daqui obtivemos a relação (2.13) entre os detalhes, os filtros e os valores funcionais. Vamos dar uma interpretação desta fórmula em

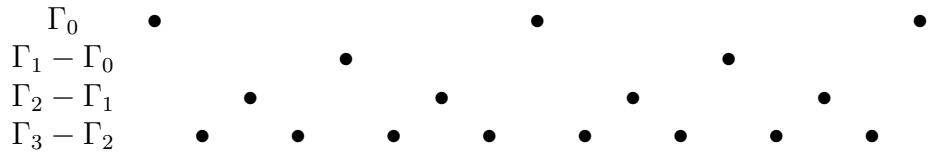


Figura 2.6: Cada ponto corresponde à localização de uma função wavelet; $j = 3$, $j_0 = 0$.

termos de uma transformada wavelet interpoladora, que é de fato o esquema que foi implementado computacionalmente em MATLAB. Daqui em diante vamos denotar por c^{j+1} o vetor que contém os valores da função f avaliados na malha diádica V_{j+1}, ou seja, $c_k^{j+1} = f_{j+1,k}$. O algoritmo para implementar esta transformada é o seguinte:

1. Dado um vetor c^{j+1}, ele é dividido em dois vetores, um correspondente aos índices pares e outro aos ímpares.

2. Os elementos pares são utilizados para prever o valor dos elementos ímpares.[1]

3. A previsão é corrigida com o valor verdadeiro dos elementos ímpares.

É conveniente considerar cada um destes passos como o resultado de aplicar um filtro nos vetores. Na Figura 2.7, o passo onde são calculados os valores previstos nos pontos correspondentes a índices ímpares é representado pelo filtro "P". O resultado desta previsão é comparado com o valor verdadeiro, mediante uma operação de subtração: quanto menor o resto, melhor a previsão. Finalmente, obtemos os dois vetores c^j (que contém os elementos pares de c^{j+1}) e d^j (que contém a comparação entre o valor dos elementos ímpares de c^{j+1} e o valor calculado a partir dos c^j). Uma das melhores propriedades desta transformada é que a inversa é imediata. Se quisermos recuperar c^{j+1} a partir de c^j e d^j, repetimos o algoritmo anterior com uma pequena modificação. Novamente interpolamos o valor dos elementos ímpares de c^{j+1} utilizando os elementos pares, e acrescentamos a diferença entre o valor previsto e o real, contido nos d^j. Finalmente, unimos os dois

[1] A previsão é feita mediante interpolação.

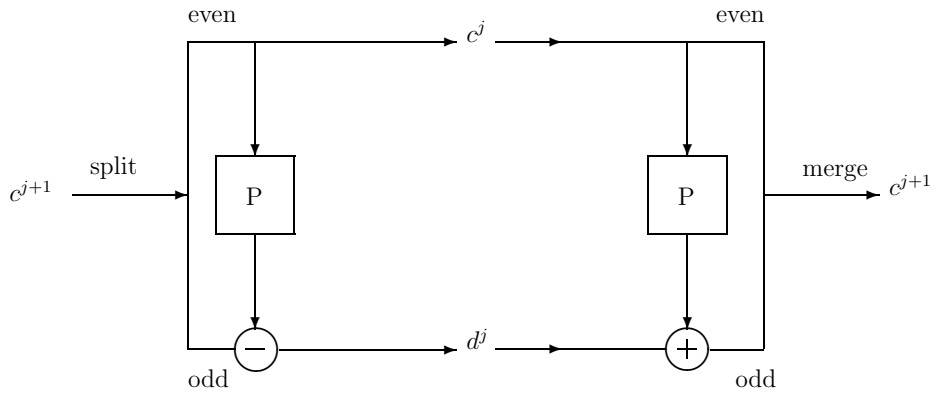


Figura 2.7: Ciclo de análise e síntese da transformada wavelet interpoladora.

vetores, intercalando seus valores.[2] O ciclo completo de análise e síntese está representado na Figura 2.7.

2.6 Wavelets biortogonais

Uma análise multirresolução ortogonal é aquela em que os elementos da base de Riesz de V_0 constituem um sistema ortonormal em $\mathcal{L}^2(\mathbb{R})$. Nesse caso, para $f \in V_0$, podemos escrever

$$f(x) = \sum_{k\in\mathbb{Z}} \langle f, \varphi_{0,k}\rangle \, \varphi_{0,k}(x), \tag{2.15}$$

porque $\langle \varphi_{0,k}, \varphi_{0,l}\rangle = \delta_{k,l}$. A relação (2.15) é válida no subespaço V_j, substituindo $\varphi_{0,k}$ por $\varphi_{j,k}$, e agora $f \in V_j$.

No nosso caso, a base de Riesz gerada pela função de escala de Deslauriers e Dubuc não é uma base ortonormal, mas introduzindo uma função dual podemos obter uma relação similar a (2.15).

Definição 2.6.1. *Uma função $\tilde{\varphi} \in \mathcal{L}^2$ tal que*

$$\langle \varphi_{j,k}, \tilde{\varphi}_{j,l}\rangle = \delta_{k,l},$$

em que $\tilde{\varphi}_{j,k}(x) = \tilde{\varphi}(2^j x - k)$, $j, k \in \mathbb{Z}$ é denominada função dual de φ.

[2]Na programação deste algoritmo a diferença entre a análise e a síntese é de apenas um sinal!

Para qualquer $f \in V_j$ temos

$$f(x) = \sum_{k\in\mathbb{Z}} \langle f, \tilde{\varphi}_{j,k}\rangle \, \varphi_{j,k}(x). \tag{2.16}$$

A função dual de φ também gera uma análise multirresolução $\tilde{V}_j$, $j \in \mathbb{Z}$. Ela satisfaz uma relação do tipo

$$\tilde{\varphi}(x) = \sum_{k\in\mathbb{Z}} \tilde{h}_k \tilde{\varphi}(2x-k).$$

A função wavelet ψ também tem sua função dual ($\tilde{\psi}$), que satisfaz a Definição 2.6.1 e que gera os espaços $\tilde{W}_j$ tais que $\tilde{V}_{j+1} = \tilde{V}_j \oplus \tilde{W}_j$. Estas wavelets, chamadas de *biortogonais*, generalizam o caso ortogonal em que $\tilde{\varphi} = \varphi$.

2.7 Representação esparsa

O nosso objetivo é representar uma função com a menor quantidade possível de pontos. A partir de uma representação wavelet obtemos uma representação wavelet esparsa quando todos os detalhes menores que ε são eliminados:

$$P^{\varepsilon}_{j+1} f = \sum_{k\in\mathbb{Z}} f_{j_0,k}\varphi_{j_0,k} + \sum_{(l,k)\in I(\varepsilon)} d_{l,k}\psi_{l,k}, \tag{2.17}$$

onde

$$I(\varepsilon) = \{(l,k) \in \mathbb{Z}\times\mathbb{Z} :\ j_0 \le l \le j,\ |d_{l,k}| \ge \varepsilon\}. \tag{2.18}$$

Se $(l,k) \in I(\varepsilon)$, o coeficiente correspondente $d_{l,k}$ é denominado *significativo*.

Devido à identificação entre os detalhes e os pontos da malha diádica, podemos construir uma malha esparsa considerando apenas os pontos correspondentes aos detalhes significativos da representação wavelet esparsa e os pontos da malha grossa (Figura 2.8).

2.8 Wavelets no intervalo

No caso de funções f definidas no intervalo $[0,1]$, considera-se a malha

$$\Gamma_j \cap [0,1] = \{x_{j,k} = 2^{-j}k,\ k = 0,1,\ldots,2^j\}.$$

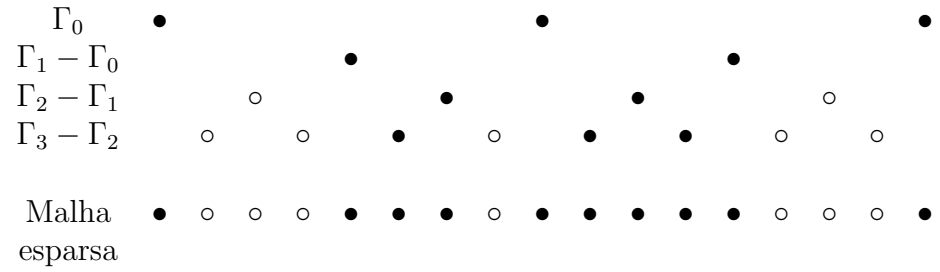


Figura 2.8: Pontos correspondentes aos detalhes significativos (•) numa representação esparsa.

Se $p = 2$ todos os dados necessários para o cálculo dos coeficientes $d_{j,k}$ são conhecidos. Se $p \geq 4$, nem sempre é possível interpolar com um esquema simétrico. Por exemplo, se $p = 4$, a interpolação cúbica centrada em $x_{j+1,2^{j+1}-1}$ requer um ponto de interpolação fora do intervalo. Por isso, são utilizados pontos de interpolação não centrados e que estão mais próximos. Neste caso, vale a fórmula

$$f_{j+1,2^{j+1}-1} = \frac{1}{16} f_{j,2^j-3} - \frac{5}{16} f_{j,2^j-2} + \frac{15}{16} f_{j,2^j-1} + \frac{5}{16} f_{j,2^j}.$$

Analogamente, próximo do extremo esquerdo, no ponto $x_{j+1,1}$ da malha, aplica-se a fórmula

$$f_{j+1,1} = \frac{5}{16} f_{j,0} + \frac{15}{16} f_{j,1} - \frac{5}{16} f_{j,2} + \frac{1}{16} f_{j,3}.$$

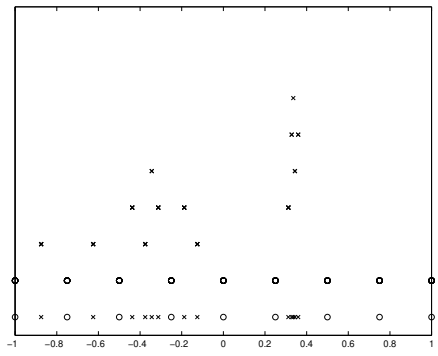
Na Figura 2.9 são apresentados os resultados da decomposição wavelet e truncamento dos detalhes da função

$$f(x) = -\tanh\left(\frac{x + x_0}{0.02}\right) + \exp\left(-64^2 (x - x_0)^2\right); \quad x_0 = \tfrac{1}{3}.$$

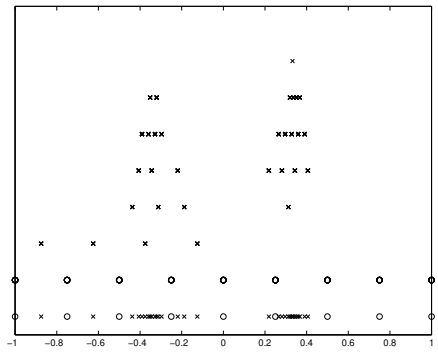
Três valores de ε (10^{-1}, 10^{-2} e 10^{-3}) foram considerados. A quantidade de pontos da malha foi 8193 em todos os casos (as malhas diádicas têm sempre $2^n + 1$ pontos). São mostrados os detalhes significativos agrupados por níveis. Cada nível é mostrado numa altitude diferente com respeito ao eixo das ordenadas. Ao nível mais fino corresponde a maior altitude. Na última linha de cada gráfico está representada a malha esparsa, que é a união da malha mais grossa e os pontos associados aos detalhes significativos. Na Figura 2.10 são apresentadas as funções reconstruídas partindo dos detalhes significativos representados na Figura 2.9. Devemos observar a pequena quantidade de

pontos necessários para representar a função de forma esparsa. No caso de $\varepsilon = 10^{-1}$ precisaremos de apenas 22 pontos, ou seja, 0.27% dos 8193 pontos originais. No caso de $\varepsilon = 10^{-2}$ serão 40 pontos (0.49%) e para $\varepsilon = 10^{-3}$, 66 pontos (0.81%). Já neste último caso a reconstrução da função é totalmente satisfatória.

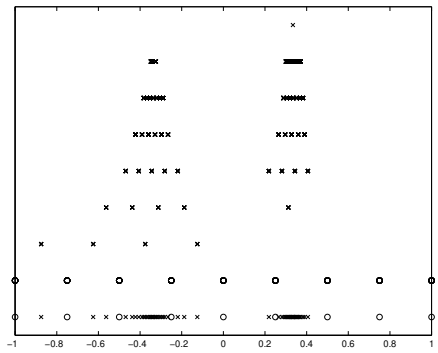
Para concluir esta seção, mostramos na Figura 2.11 a diferença entre as interpolações não esparsa e esparsa. Utilizamos uma malha com 8193 pontos novamente, embora mostremos só 129 deles na interpolação clássica para manter a clareza do gráfico (se fossem mostrados todos os pontos interpolados neste caso, seria impossível distingui-los dos pontos da função original). Nossos algoritmos representam de forma verdadeiramente esparsa a função, ou seja, são armazenados apenas os detalhes significativos e os valores da função na malha mais grossa, enquanto na literatura consultada [9] são armazenados *todos* os pontos, colocando um marcador `NaN` (*not-a-number*) naqueles que não contêm detalhes significativos. Isto sem dúvidas simplifica os algoritmos, mas nosso objetivo é, precisamente, tirar proveito do caráter esparso da representação wavelet. Veja na seção B.4 a implementação dos algoritmos utilizados para gerar os gráficos da Figura 2.11.



(a)

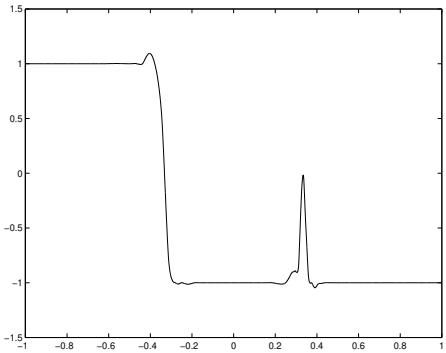


(b)

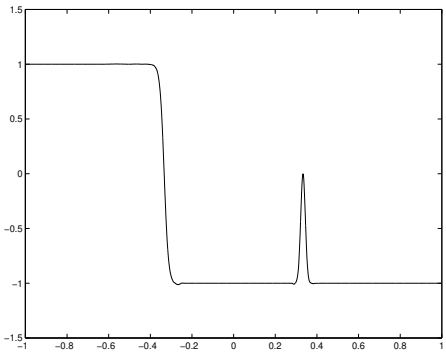


(c)

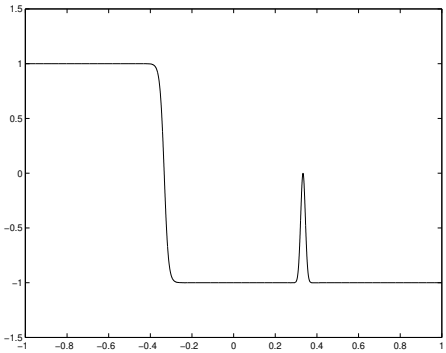
Figura 2.9: Decomposições da função f com (a) $\varepsilon = 10^{-1}$, (b) $\varepsilon = 10^{-2}$, (c) $\varepsilon = 10^{-3}$. Pontos da malha mais grossa ($\circ$), pontos com detalhes significativos ($\times$).



(a)

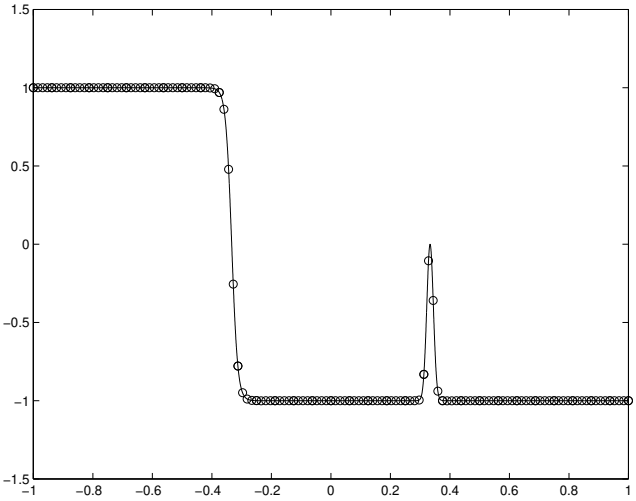


(b)

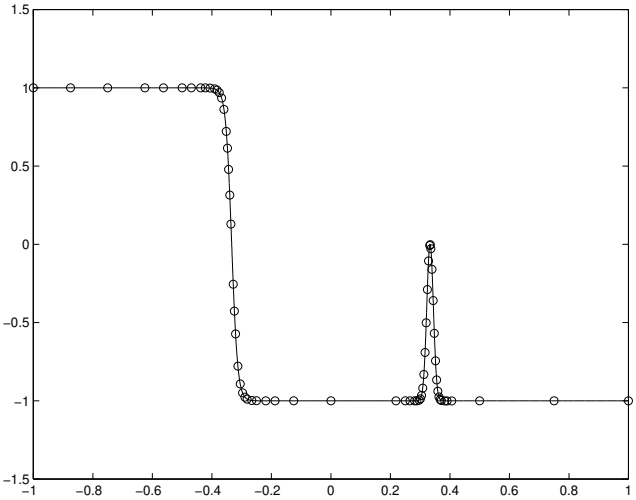


(c)

Figura 2.10: Função reconstruída. Foram truncados os detalhes menores em módulo que (a) $\varepsilon = 10^{-1}$, (b) $\varepsilon = 10^{-2}$, (c) $\varepsilon = 10^{-3}$.



(a)



(b)

Figura 2.11: Interpolação não esparsa (a) e esparsa (b). Função original ($-$) e pontos interpolados ($\circ$). O limitante é $\varepsilon = 10^{-3}$; a malha mais fina tem 8193 pontos e a mais grossa tem 5 pontos. A quantidade de pontos com detalhes significativos é 61. Ao todo são necessários 66 pontos para representar de forma esparsa a função. Observe-se que é nas zonas de mudança brusca da função onde se utilizam mais pontos; no entanto nas zonas suaves são necessários poucos.

Capítulo 3

Resolução numérica do problema da decomposição catalítica da hidrazina

3.1 Modelo da decomposição catalítica da hidrazina

Os micropropulsores a hidrazina são utilizados no controle de órbita e atitude de satélites artificiais. Na Figura 3.1 mostramos um diagrama simplificado de um micropropulsor. Suas partes básicas são um injetor, um leito fixo carregado com partículas de catalisador e uma tubeira convergente-divergente. A hidrazina líquida (N_2H_4) é injetada no leito catalítico onde se decompõe gerando uma mistura de gases a alta temperatura[1] que, ao acelerar-se através da tubeira, dá como resultado a propulsão desejada.

No que segue, os sobre-índices 1, 2, 3 e 4 referem-se à hidrazina, à amônia, ao hidrogênio e ao nitrogênio, respectivamente, assim como os sub-índices "s" e "b" (de *boiling*) se referem ao sólido e ao ponto de vaporização, respectivamente.

O objetivo é modelar o comportamento das magnitudes de interesse na reação química, que são:

T: temperatura da mistura gasosa,

[1]Hidrogênio, nitrogênio e amônia. Esta última se decompõe, gerando também hidrogênio e nitrogênio.

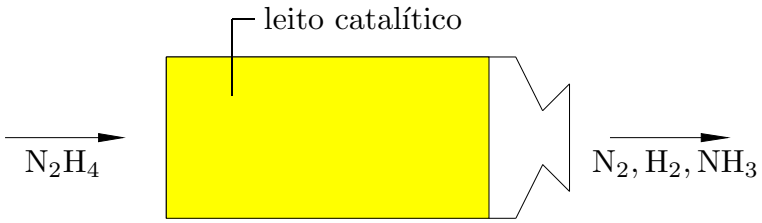


Figura 3.1: Esquema simplificado de um micropropulsor a hidrazina.

T_s: temperatura do sólido,

Y^1: fração de massa da hidrazina,

Y^2: fração de massa da amônia.

Em [6] são introduzidas variáveis adimensionais, e um modelo simplificado para governar tal sistema é apresentado. As variáveis adimensionais são:

$$\Theta = \frac{(T - T_b)c_p}{(-H^1)Y_b^1}, \qquad \Theta_s = \frac{(T_s - T_b)c_p}{(-H^1)Y_b^1},$$
$$\psi = \frac{32}{17}\frac{k^2}{k^1}\frac{Y^2}{Y_b^1}, \qquad \phi = \frac{Y^1}{Y_b^1},$$
$$z = \frac{Ah}{Gc_p}\,x, \qquad \tau = \frac{3h}{r\rho_s c_{p_s}}\,t,$$

em que

A: razão da superfície do catalizador por unidade de volume,

c_p: calor específico do gás,

c_{p_s}: calor específico do sólido,

G: vazão de massa,

H^1: calor de reação com respeito à hidrazina,

h: coeficiente de transferência de calor,

k^1, k^2: coeficientes de transferência de massa com respeito à hidrazina e à amônia,

P: pressão de operação do reator,

r: raio médio das partículas,

t: variável temporal,

T_b: temperatura de vaporização da hidrazina correspondente à pressão do reator P,

x: variável espacial,

Y_b^1: fração de massa da hidrazina no momento em que a temperatura T_b é atingida,

ρ_s: densidade do sólido.

O sistema de equações que modela o comportamento das variáveis anteriormente mencionadas é:

$$\frac{\partial \Theta}{\partial z} = -(\Theta - \Theta_s), \tag{3.1}$$

$$\frac{\partial \Theta_s}{\partial \tau} = \alpha\phi - \alpha\omega\zeta F(\Theta_s)\psi_s + (\Theta - \Theta_s), \tag{3.2}$$

$$\frac{\partial \psi}{\partial z} = -\mu(\psi - \psi_s), \tag{3.3}$$

$$\frac{\partial \phi}{\partial z} = -\alpha\phi, \tag{3.4}$$

onde

$$F(\Theta_s) = D\exp\left[\frac{-Z(\alpha - \Theta_s)}{(1+\beta\alpha)(1+\beta\Theta_s)}\right], \qquad (\psi - \psi_s) = \zeta F(\Theta_s)\psi_s - \phi.$$

Temos também as condições iniciais e de contorno:

$$\Theta(z,0) = \Theta_s^0(1 - e^{-\alpha z}),\ \psi(z,0) = \frac{\mu}{\alpha}(1 - e^{-\alpha z}) + \psi_b,\ \Theta_s(z,0) = \Theta_s^0, \tag{3.5}$$

$$\Theta(0,\tau) = 0, \quad \phi(0,\tau) = 1, \quad \psi(0,\tau) = \psi_b. \tag{3.6}$$

Da equação (3.4) e sua condição de contorno, segue-se que $\phi(z) = e^{-\alpha z}$. Nas equações anteriores, D, Z, α, β, μ e ω são parâmetros adimensionais. Os valores de todos os parâmetros e magnitudes físicas encontram-se no Apêndice C.

Além disso, $\zeta = \rho/\rho_b$. Para determinar a densidade utilizamos a equação de estado:

$$\rho = \frac{P}{RT\left[\dfrac{Y^1}{32} + \dfrac{Y^2}{17} + \dfrac{Y^3}{2} + \dfrac{Y^4}{28}\right]}. \tag{3.7}$$

No ponto de vaporização temos que $Y_b^1 = 0.87 - 0.0006\,T_b$. Como ainda não começou a decomposição da amônia, a estequiometria da reação nos informa que

$$Y_b^2 = \frac{17}{32}(1 - Y_b^1), \quad Y_b^3 = \frac{1}{32}(1 - Y_b^1), \quad Y_b^4 = \frac{14}{32}(1 - Y_b^1).$$

Assim, para determinar ρ_b utilizamos a equação (3.7) e os valores das frações de massa no ponto de vaporização. Uma vez que o processo começa, a amônia também decompõe-se em hidrogênio e nitrogênio e as novas relações para determinar a fração de massa destas substâncias são:

$$Y^3 = \frac{1}{8}(1 - Y^1) - \frac{3}{17}Y^2, \quad Y^4 = \frac{7}{8}(1 - Y^1) - \frac{14}{17}Y^2.$$

3.2 Esquemas de diferenças finitas em malhas uniformes

O objetivo do trabalho descrito nesta seção é obter soluções que sejam padrões de comparação na hora de resolver o sistema mediante o esquema adaptativo descrito na seção 3.3. Dois esquemas são considerados. O primeiro é implícito e incondicionalmente estável. Devido à dificuldade em utilizar este tipo de esquema em uma malha irregular adaptativa, considera-se também um esquema explícito que é condicionalmente estável mas que pode ser facilmente modificado para tratar a situação adaptativa.

São criadas malhas uniformes para o espaço e o tempo, com passos Δx e Δt, respectivamente. Seja $N_x + 1$ a quantidade de pontos da malha espacial, e L o comprimento do leito catalítico. Então $\Delta x = L/N_x$ e a malha espacial é formada por pontos da forma $x_i = (i-1)\,\Delta x$, com $i = 1, \ldots, N_x + 1$.

Para o tempo consideram-se valores discretos $t_m = (m-1)\,\Delta t$, com espaçamento Δt e $m = 1, 2, \ldots$

Na prática, são utilizadas as variáveis adimensionais z e τ, que são obtidas por escalamento das originais. Os passos também são escalonados na mesma

proporção que as variáveis, e denotados por δ e Δ para o espaço e o tempo, respectivamente.

De aqui em diante, adota-se a notação $f_i^m = f(z_i, \tau_m)$ para os valores de uma função f nos pontos da malha computacional.

3.2.1 Esquema implícito

Para resolver o sistema de equações (3.1)–(3.3) é proposto um esquema FTBS (*Forward in Time, Backwards in Space*) em que se aproximam as derivadas parciais de uma função f por

$$\frac{df}{d\tau}(z_i, \tau_{m+1}) \approx \frac{f(z_i, \tau_{m+1}) - f(z_i, \tau_m)}{\Delta} = \frac{f_i^{m+1} - f_i^m}{\Delta},$$

e

$$\frac{df}{dz}(z_i, \tau_{m+1}) \approx \frac{f(z_i, \tau_{m+1}) - f(z_{i-1}, \tau_{m+1})}{\delta} = \frac{f_i^{m+1} - f_{i-1}^{m+1}}{\delta}.$$

Utilizando estas aproximações, temos que

$$\frac{d\Theta}{dz}(z_i, \tau_{m+1}) = \frac{\Theta_i^{m+1} - \Theta_{i-1}^{m+1}}{\delta} = -\Theta_i^{m+1} + {\Theta_s}_i^{m+1},$$

portanto

$$-\Theta_{i-1}^{m+1} + (1+\delta)\Theta_i^{m+1} - \delta {\Theta_s}_i^{m+1} = 0, \quad i = 2, \ldots, N_x + 1; \tag{3.8}$$

$$\frac{d\Theta_s}{d\tau}(z_i, \tau_{m+1}) = \frac{{\Theta_s}_i^{m+1} - {\Theta_s}_i^m}{\Delta} = \\ = \alpha\phi_i - \alpha\omega\zeta_i^m \frac{\psi_i^{m+1} + \phi_i}{1 + \zeta_i^m F_i^{m+1}} F_i^{m+1} + \Theta_i^{m+1} - {\Theta_s}_i^{m+1},$$

logo,

$$(1+\Delta){\Theta_s}_i^{m+1} - \Delta\Theta_i^{m+1} = \\ = {\Theta_s}_i^m + \alpha\Delta\phi_i - \alpha\omega\Delta\zeta_i^m \frac{\psi_i^{m+1} + \phi_i}{1 + \zeta_i^m F_i^{m+1}} F_i^{m+1}, \quad i = 1, \ldots, N_x + 1; \tag{3.9}$$

$$\frac{d\psi}{dz}(z_i, \tau_{m+1}) = \frac{\psi_i^{m+1} - \psi_{i-1}^{m+1}}{\delta} = -\mu\psi_i^{m+1} + \mu\frac{\psi_i^{m+1} + \phi_i}{1 + \zeta_i^m F_i^{m+1}},$$

portanto

$$-\psi_{i-1}^{m+1} + (1 + \mu\delta)\psi_i^{m+1} = \mu\delta\frac{\psi_i^{m+1} + \phi_i}{1 + \zeta_i^m F_i^{m+1}}; \quad i = 2, \ldots, N_x + 1. \tag{3.10}$$

Nas equações (3.9) e (3.10), $F_i^{m+1} = F({\Theta_s}_i^{m+1})$. Este sistema pode ser representado matricialmente por

$$MU^{m+1} = R(U^m) + H(U^{m+1}), \tag{3.11}$$

onde U^{m+1} é um vetor contendo os valores de Θ, Θ_s e ψ, organizado em tríades:

$$\begin{aligned} U_1^{m+1} &= {\Theta_s}_1^{m+1}, \\ U_{3(i-1)-1}^{m+1} &= \Theta_i^{m+1}, \\ U_{3(i-1)}^{m+1} &= {\Theta_s}_i^{m+1}, \\ U_{3(i-1)+1}^{m+1} &= \psi_i^{m+1}, \end{aligned}$$

para $i = 2, \ldots, N_x + 1$. Note que os valores de Θ_1^{m+1} e de ψ_1^{m+1} são conhecidos (condições de contorno), e não é preciso calculá-los. A matriz M tem a estrutura seguinte:

$$M = \begin{bmatrix} A_1 & & & & & \\ & A & & & & \\ & B & A & & & \\ & & B & \ddots & & \\ & & & \ddots & A & \\ & & & & B & A \end{bmatrix},$$

onde

$$A_1 = [1 + \Delta], \quad A = \begin{bmatrix} 1+\delta & -\delta & 0 \\ -\Delta & 1+\Delta & 0 \\ 0 & 0 & 1+\mu\delta \end{bmatrix}, \quad B = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix}.$$

O vetor $R(U^m)$ é função de elementos de U^m, ou seja, de valores do passo anterior do tempo. Sua estrutura é:

$$\begin{aligned} R_1^m &= \Theta_{s1}{}^m, \\ R_{3(i-1)-1}^m &= 0, \\ R_{3(i-1)}^m &= \Theta_{si}{}^m, \\ R_{3(i-1)+1}^m &= 0, \end{aligned}$$

para $i = 2, \ldots, N_x + 1$. O vetor $H(U^{m+1})$ contém os elementos não lineares do sistema de equações (3.8)–(3.10), sua estrutura é:

$$\begin{aligned} H_1^{m+1} &= \Delta\Theta_1^{m+1} + \alpha\Delta\phi_1 - \alpha\Delta\omega\zeta_1^m \frac{\psi_1^{m+1} + \phi_1}{1 + \zeta_1^m F_1^{m+1}} F_1^{m+1}, \\ H_2^{m+1} &= \Theta_1^{m+1}, \\ H_3^{m+1} &= \alpha\Delta\phi_2 - \alpha\Delta\omega\zeta_2^m \frac{\psi_2^{m+1} + \phi_2}{1 + \zeta_2^m F_2^{m+1}} F_2^{m+1}, \\ H_4^{m+1} &= \psi_1^{m+1} + \mu\delta \frac{\psi_2^{m+1} + \phi_2}{1 + \zeta_2^m F_2^{m+1}}, \\ H_{3(i-1)-1}^{m+1} &= 0, \\ H_{3(i-1)}^{m+1} &= \alpha\Delta\phi_i - \alpha\Delta\omega\zeta_i^m \frac{\psi_i^{m+1} + \phi_i}{1 + \zeta_i^m F_i^{m+1}} F_i^{m+1}, \\ H_{3(i-1)+1}^{m+1} &= \mu\delta \frac{\psi_i^{m+1} + \phi_i}{1 + \zeta_i^m F_i^{m+1}}, \end{aligned}$$

para $i = 3, \ldots, N_x + 1$.

Para determinar U^{m+1} a partir do sistema (3.11) se utiliza o método de Newton, ou seja, o sistema linear

$$(M - H')(U_p^{m+1} - U_{p-1}^{m+1}) = R(U^m) - MU_{p-1}^{m+1} + H(U_{p-1}^{m+1})$$

é resolvido para U_p^{m+1}. O sub-índice p indica o número da iteração do método de Newton dentro do passo temporal correspondente a τ_{m+1}. Tal iteração é alimentada inicialmente com o valor de U que resulta do passo anterior:

$$U_0^{m+1} = U_{\text{final}}^m = U^m.$$

O passo τ_{m+1} se considera concluído se

$$\max \left| U_p^{m+1} - U_{p-1}^{m+1} \right| < \varepsilon,$$

sendo ε fixado *a priori* e comum para todos os passos temporais. A matriz jacobiana H' tem a seguinte estrutura:

$$H' = \begin{bmatrix} J_1 & & & & \\ & J_2 & & & \\ & & J_3 & & \\ & & & \ddots & \\ & & & & J_{N_x+1} \end{bmatrix},$$

onde

$$J_1 = [0], \quad J_i = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -\dfrac{\alpha\Delta\omega\zeta_i^m(\psi_i^{m+1}+\phi_i)}{(1+\zeta_i^m F_i^{m+1})^2}G_i^{m+1} & -\dfrac{\alpha\Delta\omega\zeta_i^m F_i^{m+1}}{1+\zeta_i^m F_i^{m+1}} \\ 0 & -\dfrac{\mu\delta\zeta_i^m(\psi_i^{m+1}+\phi_i)}{(1+\zeta_i^m F_i^{m+1})^2}G_i^{m+1} & \dfrac{\mu\delta}{1+\zeta_i^m F_i^{m+1}} \end{bmatrix},$$

com $G(\Theta_s) = F'(\Theta_s) = ZF(\Theta_s)/(1+\beta\Theta_s)^2$, e $i = 2, \ldots, N_x + 1$.

3.2.2 Esquema explícito

Neste caso, os termos não lineares são tratados explicitamente. Portanto, as equações resultantes são:

$$\Theta_i^{m+1} = \frac{1}{1+\delta}(\Theta_{i-1}^{m+1} + \delta\Theta_{si}^{\ m+1}); \tag{3.12}$$

$$\Theta_{si}^{\ m+1} = \alpha\Delta\phi_i - \alpha\Delta\omega\zeta_i^m\frac{\psi_i^m+\phi_i}{1+\zeta_i^m F_i^m}F_i^m + \Delta\Theta_i^m + (1+\Delta)\Theta_{si}^{\ m}; \tag{3.13}$$

$$\psi_i^{m+1} = \frac{1}{1+\mu\delta}\left[\psi_{i-1}^{m+1} + \mu\delta\frac{\psi_i^m+\phi_i}{1+\zeta_i^m F_i^m}\right]. \tag{3.14}$$

3.2.3 Resultados numéricos

Os programas para o esquema implícito encontram-se na seção B.5. Na Figura 3.2 apresentamos os resultados da aplicação deste esquema para uma malha uniforme de 129 pontos e $\Delta = 0.01$. Podemos observar que, com a rápida decomposição da hidrazina na entrada do reator, as temperaturas da mistura gasosa e do leito catalítico crescem, imediatamente nesta região. Com o passar do tempo, o aumento da temperatura se propaga ao longo do reator e a decomposição da amônia passa a ser também importante, o que é refletido no decaimento da concentração deste gás à medida que passa o tempo. Para $\tau = 50$ o problema já alcança o regime estacionário.

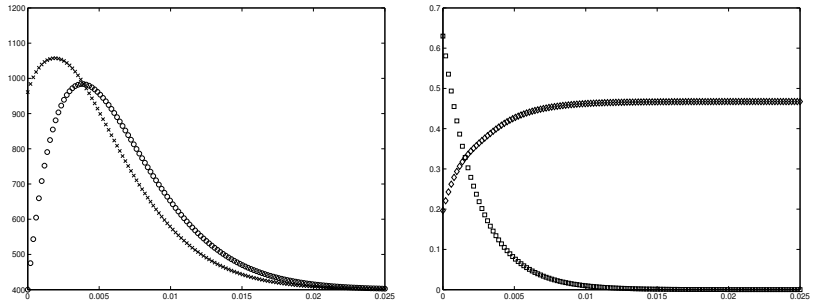
Tanto o esquema implícito quanto o esquema explícito foram implementados para diferentes valores de δ e Δ para efeito de análise de estabilidade. Como esperado, o esquema implícito é estável, produzindo resultados satisfatórios, mesmo com grandes passos de tempo. Já o esquema explícito somente produz bons resultados com uma escolha de Δ adequada com a escala espacial δ. Na Figura 3.3 são indicados a diferença máxima entre as distribuições de temperatura da mistura gasosa no estado estacionário ($\tau = 50$) obtidas em ambos esquemas, em função do parâmetro $\lambda = \Delta/\delta$.[2] Esses resultados sugerem que, para garantir uma boa precisão, pode-se adotar a relação $\lambda < \frac{1}{16}$ nas aplicações do esquema explícito.

3.3 Esquema adaptativo

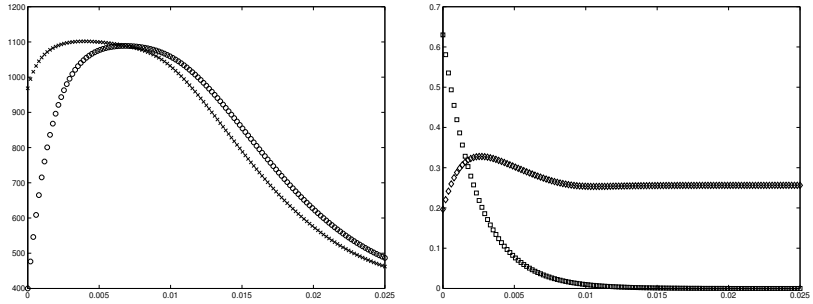
O objetivo fundamental deste trabalho é resolver o problema (3.1)–(3.6) mediante um esquema adaptativo por meio de uma combinação de um método de diferenças finitas e de ferramentas baseadas na análise wavelet. Usando coeficientes wavelets como indicadores de regularidade local, em cada passo de tempo, determina-se a distribuição de pontos que permita representar as funções com uma precisão desejada, de maneira a reduzir o custo computacional.

Como descrito na seção anterior, a solução do problema em questão apresenta diferentes padrões de comportamento no domínio espacial e ao longo do tempo. Observou-se que no começo do reator, onde ocorre a rápida decom-

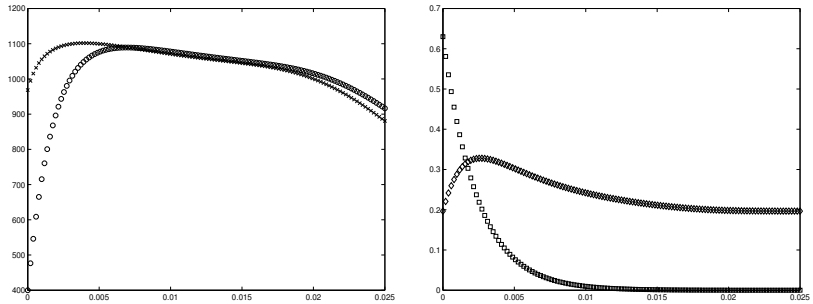
[2]Fixando a quantidade de pontos da malha (33, 65 e 129 pontos) foi analizado para vários valores de Δ $(0.1, 0.2, \ldots, 1.5)$ o comportamento da máxima diferença entre os valores da temperatura obtidos mediante o método explícito e o método implícito.



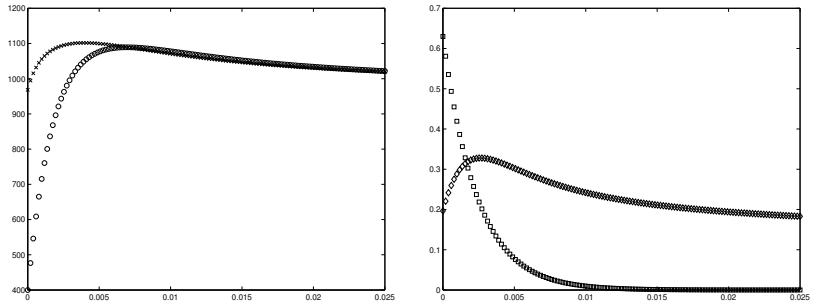
(a) $\tau = 4$.



(b) $\tau = 10$.



(c) $\tau = 20$.



(d) $\tau = 50$.

Figura 3.2: Comportamento de T ($\circ$), T_s ($\times$), Y^1 ($\square$) e Y^2 ($\diamond$) ao longo do leito catalítico para diferentes valores de τ (resultados obtidos mediante o método implícito, seção 3.2). Em todos os casos a malha tem 129 pontos, e $\Delta = 0.01$.

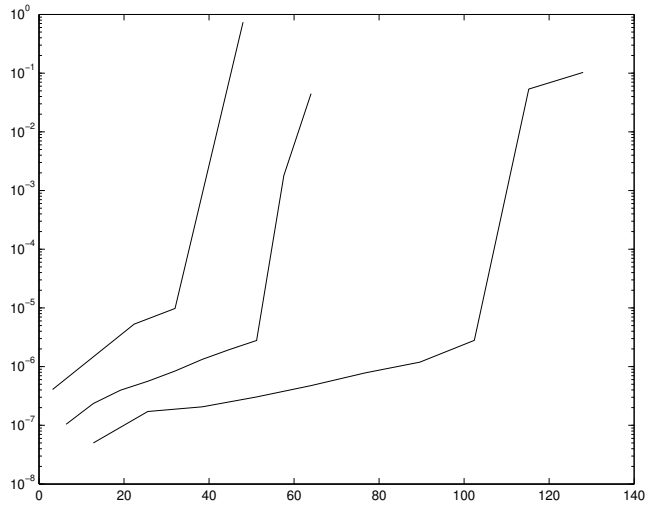


Figura 3.3: Diferenças máximas entre os resultados para o valor da temperatura obtidos mediante o esquema explícito, tomando como padrão os resultados obtidos mediante o esquema implícito (eixo das abscissas) com respeito a λ (eixo das ordenadas).

posição da hidrazina, com forte liberação de calor, a temperatura tem uma variação brusca. A concentração da amônia também está sujeita a variações à medida que a temperatura se eleva ao longo do reator. Portanto, espera-se que na representação da solução, algumas regiões precisem ser mais refinadas do que outras e que tal refinamento varie com o tempo. Por exemplo, apresentam-se na Figura 3.4 os perfis de ψ em $\tau = 0$ e $\tau = 6$ e a disposição de coeficientes wavelet com magnitude superior a 10^{-4}. São mostrados os detalhes significativos agrupados por níveis. Cada nível é mostrado numa altitude diferente com respeito ao eixo das ordenadas. Ao nível mais fino corresponde a maior altitude. Na última linha de cada gráfico está representada a malha esparsa, que é a união da malha mais grossa e os pontos associados aos detalhes significativos.

No método a ser considerado, em cada passo de tempo $\tau = \tau_m$, a solução é representada por um vetor U^m formado pelos valores das funções nos pontos de uma malha adaptativa X^m, que deve ser dinamicamente modificada, conforme a variação da solução. Seja Γ_j uma malha regular com espaçamento $\delta_j = \frac{L}{L_c} 2^{-j}$. Em cada nível de resolução, denota-se por Λ_l o conjunto de

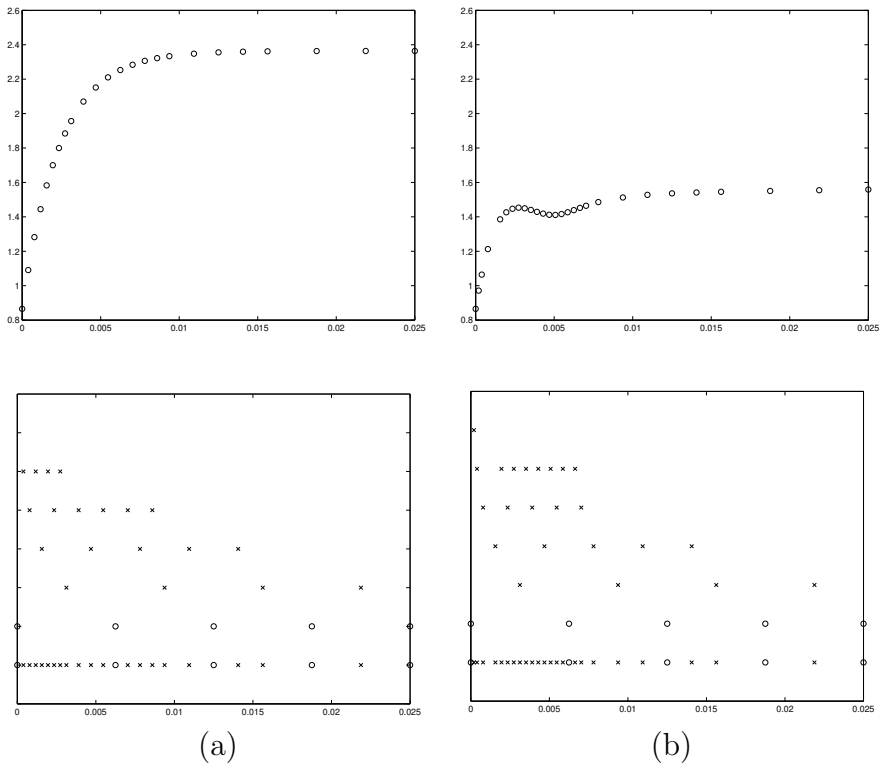


Figura 3.4: Gráficos de ψ para $\tau = 0$ (a) e $\tau = 6$ (b), e seus pontos com coeficientes significativos ($\times$) e da malha básica ($\circ$).

pontos de Γ_l que não estão em Γ_{l-1}. Forma-se a malha X^m pela união dos pontos de uma malha Γ_{j_0}, de um nível menos refinado, com aqueles pontos de $\Lambda_l, j_0 \leq l \leq j$, em que pelo menos uma das funções tenha o coeficiente wavelet significativo.

A evolução para o próximo passo de tempo é feita em três etapas:

- **Extensão da malha**: Como não sabemos *a priori* como será X^{m+1}, considera-se uma estimativa X^{m+}. Esta estimativa é crucial pois depende da velocidade com que a solução varia no intervalo de tempo. Na aplicação deste trabalho, adota-se sistematicamente um refinamento em que X^{m+} é obtida pela inclusão dos pontos intermédiarios entre pontos consecutivos da malha X^m. Nos pontos novos da malha estendida, o valor da solução em τ_m é interpolada.

- **Evolução**: Calcula-se a solução na malha estendida no tempo $\tau_{m+1} = \tau_m + \Delta_m$, sendo Δ_m escolhido de acordo com a menor distância entre dois pontos consecutivos de X^{m+}, utilizado o esquema BTBS (*Backwards in Time, Backwards in Space*) explícito. Para a temperatura do leito catalítico Θ_s, isto é feito diretamente usando a fórmula (3.13). Para as funções Θ e ψ, a derivada espacial em cada ponto da malha estendida $\nu \in X^{m+}$ é aproximada pela diferença finita

$$\frac{df}{dz}(\nu, \tau_{m+1}) \approx \frac{f(\nu, \tau_{m+1}) - f(\nu_-, \tau_{m+1})}{\delta_\nu}$$

com espaçamento δ_ν, determinado pela menor distância entre ν e seus vizinhos e $\nu_- = \nu - \delta_\nu$. Como eventualmente ν_- pode não coincidir com um ponto de X^{m+}, os valores das funções precisam ser interpolados em tais pontos (pontos *ghosts*). A evolução temporal se transforma num processo iterativo. As equações (3.12) e (3.14) são modificadas:

$$\Theta_{\nu,p}^{m+1} = \frac{1}{1+\delta_\nu}(\Theta_{\nu_-,p}^{m+1} + \delta_\nu {\Theta_s}_\nu^{m+1}), \tag{3.15}$$

$$\psi_{\nu,p}^{m+1} = \frac{1}{1+\mu\delta_\nu}\left[\psi_{\nu_-,p}^{m+1} + \mu\delta_\nu \frac{\psi_\nu^m + \phi_\nu}{1+\zeta_\nu^m F_\nu^m}\right], \tag{3.16}$$

sendo p o índice da iteração. Se para calcular o valor das funções em ν for necessário acrescentar um ponto *ghost*, então $\Theta_{\nu_-,p}^{m+1}$ e $\psi_{\nu_-,p}^{m+1}$, são interpolados utilizando à esquerda valores funcionais determinados na atual iteração, e à direita valores funcionais determinados na iteração anterior. Para começar o processo iterativo, definem-se $\Theta_{\nu,0}^{m+1} = \Theta_\nu^m$ e $\psi_{\nu,0}^{m+1} = \psi_\nu^m$. O processo termina quando a diferença entre os valores consecutivos é menor que um parâmetro de precisão determinado *a priori*.

- **Redução da malha**: A malha X^{m+1} é determinada retirando-se de X^{m+} aqueles pontos em que todos os coeficientes wavelet da solução calculada são desprezíveis.

3.3.1 Resultados numéricos

Na seção B.6 são descritos os programas e funções de MATLAB que implementam o esquema adaptativo. Nos exemplos apresentados a seguir, o esquema

	129 pontos		257 pontos	
τ	**Implícito**	**Adaptativo**	**Implícito**	**Adaptativo**
4	7.5	7.3	26.7	12.0
10	19.6	21.6	70.9	38.5
20	39.5	35.8	142.2	64.7
50	77.7	77.6	278.8	142.4

Tabela 3.1: Comparação do tempo utilizado pelos métodos implícito e adaptativo, em segundos, para malhas regulares de 129 e 257 pontos, respectivamente.

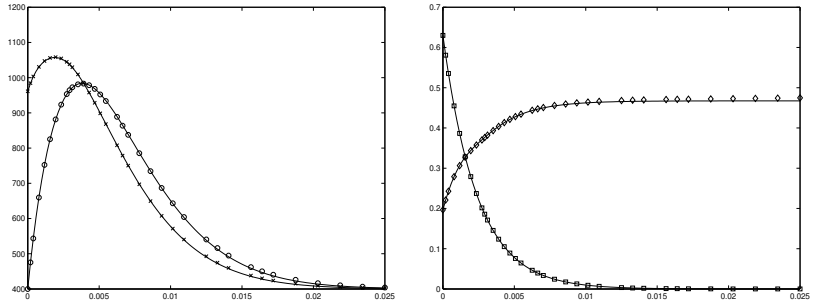
de interpolação é polinomial de grau 3 e o parâmetro de truncamento dos coeficientes wavelet é 10^{-4}.

Na Tabela 3.1 é apresentado o tempo necessário para obter as soluções mediante os métodos implícito e adaptativo, sendo $\Delta = 0.01$. Note que se a quantidade de pontos da malha regular é relativamente pequena, o método adaptativo não é tão eficiente, devido aos complicados cálculos que têm que ser realizados. Já com o aumento da quantidade de pontos da malha regular resulta evidente a vantagem que representa o uso da malha esparsa: o método adaptativo pode achar as soluções do problema na metade do tempo requerido pelo método implícito!

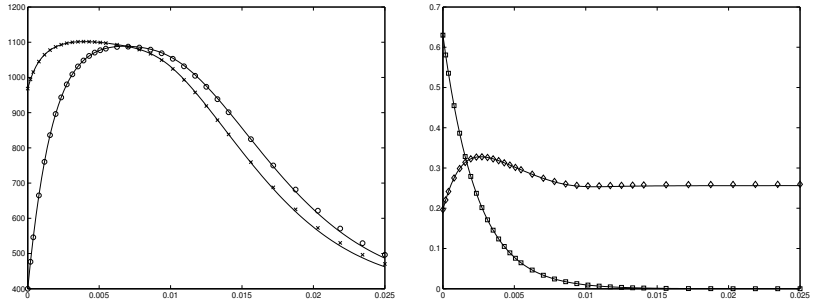
Os resultados da Figura 3.5 permitem uma comparação das soluções obtidas mediante o esquema implícito numa malha regular de 129 pontos e o esquema adaptativo. Para esta comparação, adotou-se um passo de tempo fixo $\Delta = 0.01$, em ambos casos. Observa-se a reduzida quantidade de pontos que são utilizados no método adaptativo, mantendo a qualidade das soluções com reduzido custo computacional.

Resultados análogos na Figura 3.6 permitem uma comparação das soluções obtidas mediante o esquema explícito numa malha regular de 1025 pontos e o esquema adaptativo. Observa-se que, neste caso, o esquema implícito fica inviável (em termos da capacidade do MATLAB para resolver sistemas de equações envolvendo mais de 15 000 equações, no computador utilizado para realizar os cálculos.) Para esta comparação, adotou-se um passo de tempo fixo $\Delta = 0.001$, em ambos casos.

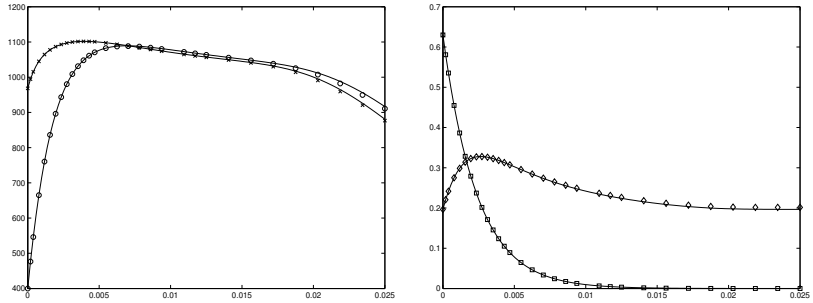
O número de pontos das malhas efetivamente utilizados no método adaptativo ao longo do tempo é apresentado na Figura 3.7. Os diferentes gráficos correspondem a aplicações em que a malha regular mais fina possui $65,257$



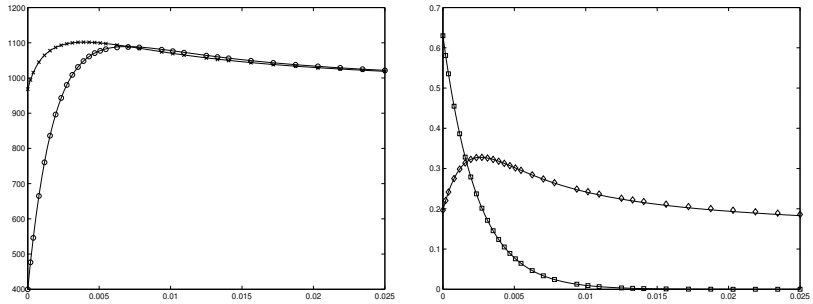
(a) $\tau = 4$.



(b) $\tau = 10$.

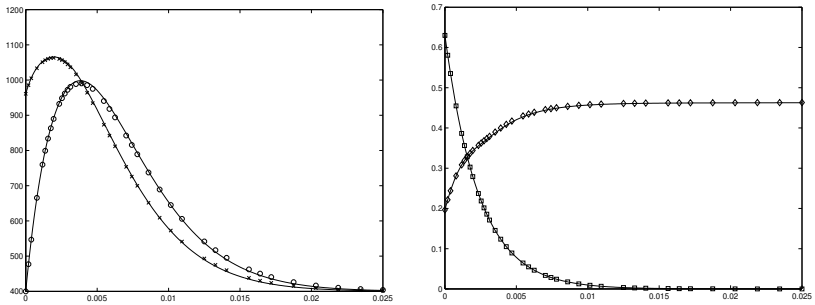


(c) $\tau = 20$.

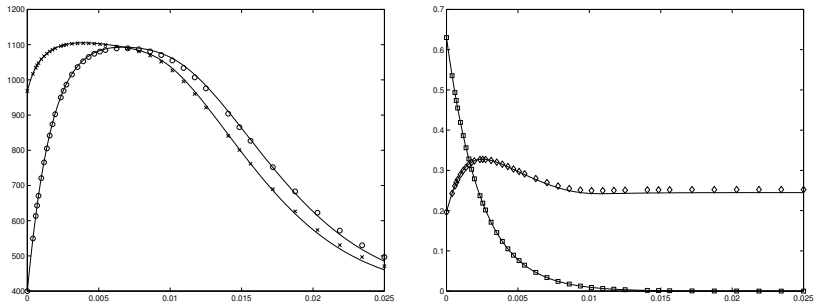


(d) $\tau = 50$.

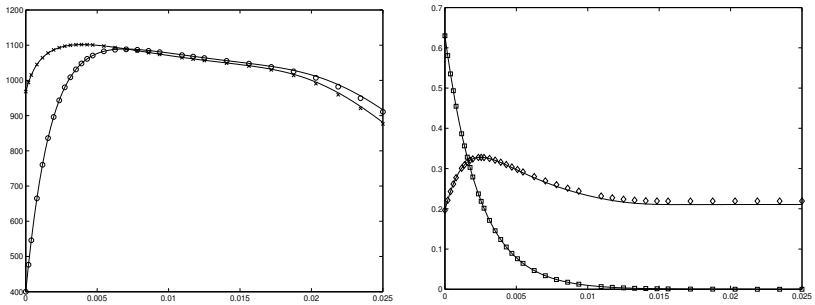
Figura 3.5: Comportamento de T ($\circ$), T_s ($\times$), Y^1 ($\square$) e Y^2 ($\diamond$) ao longo do leito catalítico para diferentes valores de τ (resultados obtidos mediante o esquema adaptativo, seção 3.3). Em todos os casos a malha original tem 129 pontos, e $\Delta = 0.01$. As linhas contínuas representam os valores destas funções calculados mediante o esquema implícito.



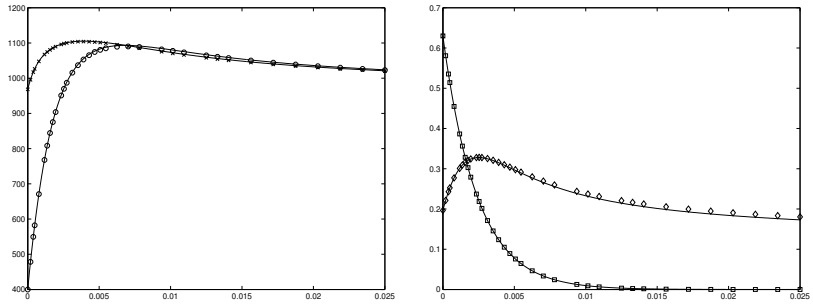
(a) $\tau = 4$.



(b) $\tau = 10$.



(c) $\tau = 20$.



(d) $\tau = 50$.

Figura 3.6: Comportamento de T ($\circ$), T_s ($\times$), Y^1 ($\square$) e Y^2 ($\diamond$) ao longo do leito catalítico para diferentes valores de τ (resultados obtidos mediante o esquema adaptativo, seção 3.3). Em todos os casos a malha original tem 1025 pontos, e $\Delta = 0.001$. As linhas contínuas representam os valores destas funções calculados mediante o esquema explícito.

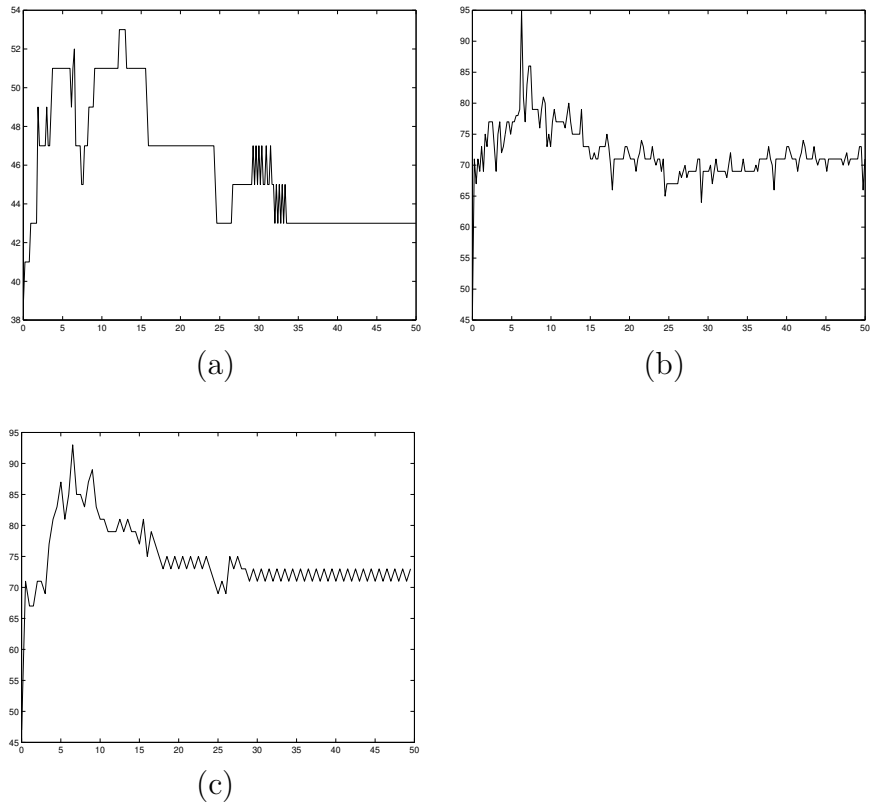


Figura 3.7: Quantidade de pontos significativos que ao longo do tempo ($0 \leq \tau \leq 50$) são utilizados para representar as funções. As malhas originais têm (a) 65, (b) 257 e (c) 1025 pontos.

e 1025 pontos. Pode-se observar que, mesmo permitindo um alto nível de refinamento em regiões de variações bruscas, o número total de pontos nas malhas adaptativas fica reduzido durante a evolução temporal.

Observa-se também que, como esperado, o esquema adaptativo gera automaticamente uma malha esparsa, sendo nas zonas de maior variação das funções onde se concentra a maior quantidade de pontos da malha, havendo menos pontos nas zonas suaves das funções.

Capítulo 4

Conclusões

Foram estudadas técnicas wavelet para a decomposição e reconstrução de funções. Tais técnicas foram utilizadas com êxito na resolução de um sistema de equações diferenciais parciais que modelam a decomposição catalítica da hidrazina num micropropulsor. Quando comparados os resultados do método adaptativo com os que provêm de um método clássico, aprecia-se a qualidade daqueles. O método adaptativo apresenta vantagens se comparado com o clássico: aproveitando o caráter esparso da representação wavelet foram utilizados pouquíssimos pontos significativos, o que resulta na redução palpável de operações efetuadas e de dados armazenados, sem sacrificar a qualidade dos resultados. Recomenda-se portanto a continuação do estudo destas técnicas e ferramentas, poderosas e versáteis, aplicáveis às mais diversas áreas do conhecimento científico e técnico.

Apêndice A

Duas ferramentas implementadas

Além do análise wavelet interpolador, foram estudadas e implementadas duas ferramentas: o esquema ENO e o esquema *lifting*. Apesar de suas propriedades relevantes, elas ainda não foram aplicadas no método adaptativo implementado. No entanto, e com a esperança de que sejam de utilidade no futuro, para nós ou para quem possa se interessar nesta área, deixamo-las aqui apresentadas.

A.1 Algoritmo essencialmente não-oscilatório

Em interpolação polinomial de grau $r > 1$ de uma função contínua por partes, pode aparecer um fenômeno de oscilação não desejado, conhecido como *fenômeno de Gibbs*. Pela importância de evitar este erro na resolução aproximada de problemas de evolução, com soluções descontínuas, como as que aparecem em leis de conservação, adota-se uma implementação computacional de um esquema de subdivisão interpolador chamado ENO (essencialmente não oscilatório) proposto em [8].

Nosso espaço de funções aproximantes será o espaço de funções polinomiais por partes na malha uniforme $\Gamma_N = \{a = x_0, x_1, \ldots, x_{N-1}, x_N = b\}$:

$$V = \left\{ Q(x) \in \mathcal{C}[a,b],\ \ Q(x)|_{[x_i, x_{i+1}]} = p(x) \in P,\, i = 0, \ldots, N-1 \right\},$$

onde P é um espaço de funções polinomiais. Dizemos que $Q(x) \in V$ é um *interpolador da função* $f(x)$ se $Q(x_i) = f(x_i)$. Se $P = \operatorname{span}\{1, x\}$, Q é um

interpolador linear por partes; se $P = \text{span}\{1, x, x^2\}$, Q é um *interpolador quadrático por partes* e se $P = \text{span}\{1, x, x^2, x^3\}$, Q é dito ser um *interpolador cúbico por partes.*

Definição A.1.1. *O interpolante Q é* essencialmente não-oscilatório *se a quantidade de seus extremos locais não excede os de f.*

No caso simples de interpolação linear por partes, no intervalo $[x_i, x_{i+1}]$, tem-se que

$$Q(x) = \gamma_0 + \gamma_1(x - x_i)$$

em que

$$\gamma_0 = [x_i] = f(x_i),$$
$$\gamma_1 = [x_{i+1}\ x_i] = \frac{[x_{i+1}] - [x_i]}{x_{i+1} - x_i}.$$

Neste caso, um extremo de Q só pode ocorrer em algum dos pontos de interpolação x_i, onde Q muda de monotonicidade. Isto é,

$$Q(x_{i-1}) < Q(x_i) \quad \text{e} \quad Q(x_i) > Q(x_{i+1})$$

ou

$$Q(x_{i-1}) > Q(x_i) \quad \text{e} \quad Q(x_i) < Q(x_{i+1}).$$

Mas então f também muda de monotonicidade no intervalo $[x_{i-1}, x_{i+1}]$. Por isso, f tem pelo menos um extremo local no intervalo $[x_{i-1}, x_{i+1}]$.

Seja $Q(x)$ um interpolante quadrático por partes da função $f(x)$. Vamos escrever $Q|_{[x_i, x_{i+1}]} = p(x)$ na forma de Newton:

$$p(x) = \gamma_0 + \gamma_1(x - x_i) + \gamma_2(x - x_i)(x - x_{i+1}), \quad x_i \le x \le x_{i+1}. \qquad \text{(A.1)}$$

Observa-se que, para qualquer escolha do coeficiente γ_2, as condições

$$p(x_i) = f(x_i), \qquad p(x_{i+1}) = f(x_{i+1})$$

são satisfeitas. No esquema de interpolação clássico, tomando

$$\gamma_2 = \gamma^+ = [x_{i+2}\ x_{i+1}\ x_i] = \frac{[x_{i+2}\ x_{i+1}] - [x_{i+1}\ x_i]}{x_{i+2} - x_i},$$

resulta que $p(x)$ também interpola f em x_{i+2}. Da mesma forma, tomando

$$\gamma_2 = \gamma^- = [x_{i+1}\ x_i\ x_{i-1}] = \frac{[x_{i+1}\ x_i] - [x_i\ x_{i-1}]}{x_{i+1} - x_{i-1}},$$

então $p(x)$ também interpola f em x_{i+2}.

No caso do esquema ENO, γ_2 também será eleito a partir das diferenças divididas de ordem 2, procurando que $Q = Q_{\text{ENO}}$ oscile tanto ou menos que f no intervalo $[x_i, x_{i+1}]$. Desta forma, o método ENO propõe a seguinte escolha:

$$\gamma_2 = \text{minmod}(\gamma^+, \gamma^-) = \begin{cases} \text{sgn}(\gamma^-)\min\{|\gamma^-|, |\gamma^+|\}, & \text{se } \text{sgn}(\gamma^-) = \text{sgn}(\gamma^+), \\ 0, & \text{se } \text{sgn}(\gamma^-) \neq \text{sgn}(\gamma^+). \end{cases}$$

Teorema A.1.1. *Seja $Q = Q_{\text{ENO}} \in V$ o interpolante ENO quadrático de f associado à malha Γ_N. Existe uma correspondência 1-1 entre os extremos locais de Q e aqueles do interpolador linear por partes de f.*

Com este resultado, garantimos que o método é essencialmente não oscilatório, porque a função Q_{ENO} tem a mesma quantidade de extremos locais que a função linear por partes que coincide com f nos x_i, cujos extremos são em quantidade igual ou inferior que os da própria f. A demonstração está em [8]. Nesse artigo também está definido o esquema ENO para interpolação cúbica. No entanto, apesar das evidências favoráveis dos resultados numéricos, ainda não se conhece uma demostração de um resultado equivalente ao Teorema A.1.1 no caso de interpolação ENO cúbica.

O esquema ENO foi preparado em MATLAB, em duas variantes: para interpolação quadrática e cúbica, sendo esta última uma extensão a grau 3 do algoritmo para grau 2 anteriormente descrito.

Para testar os programas foi utilizada a função

$$f(x) = \begin{cases} 1, & 0 \leq x \leq \frac{1}{3}, \\ 0, & \frac{1}{3} < x \leq \frac{1}{2}, \\ -1, & \frac{1}{2} < x \leq 1. \end{cases}$$

Nas Figuras A.1 e A.2 aparecem os resultados. Em todos os casos a malha original tinha 17 pontos. Além de apresentar os gráficos obtidos mediante os ENOs, também aparecem aqueles que resultaram da não utilização da função

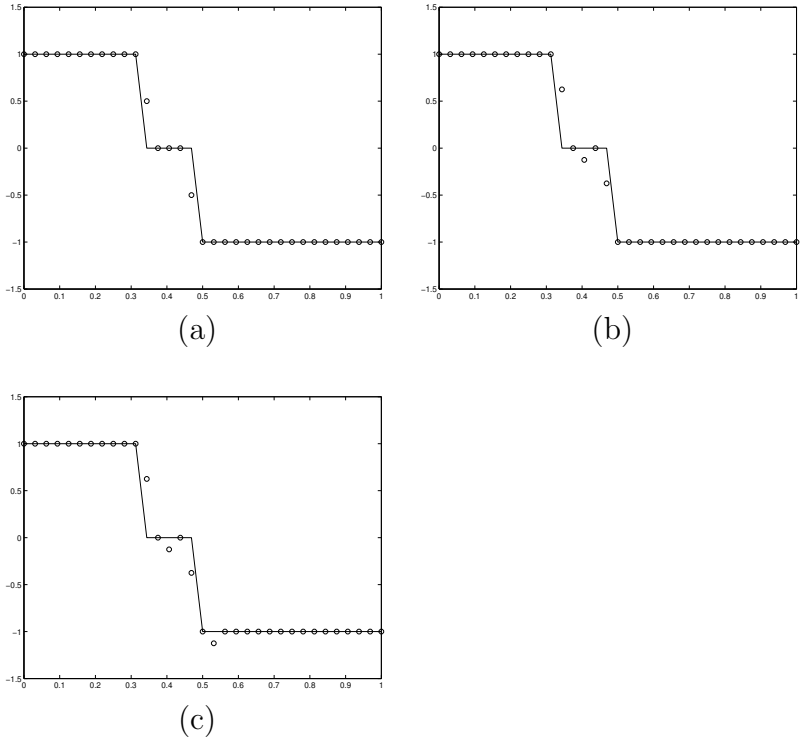


Figura A.1: Interpolação quadrática: (a) ENO, (b) pseudo-ENO, (c) clássica.

minmod, mas apenas se decidiam pela diferença dividida de menor módulo, desconsiderando o caso em que o sinal delas fosse diferente (este esquema, variação do ENO, é denominado "pseudo-ENO" nas Figuras A.1 e A.2). Para finalizar, também são apresentados os gráficos obtidos mediante interpolação quadrática e cúbica clássicas. Na seção B.1 são descritos os programas criados em MATLAB para efetuar a interpolação ENO.

Como pode ser visto, o ENO conseguiu eliminar as oscilações nas zonas críticas. Entretanto, o custo computacional do método é elevado. Em [7] foi sugerida por A. Harten a idéia de combinar o esquema ENO com um algoritmo que indique a localização de zonas críticas, e assim empregar o ENO apenas nelas, enquanto um método clássico é empregado no restante dos pontos.

A.2 O esquema *lifting*

Às vezes é preciso modificar o esquema descrito na seção 2.5 para melhorar em algum sentido as propriedades das nossas wavelets. Havendo tal necesidade, é introduzido um filtro de atualização, denotado por "U" (de *update*), que

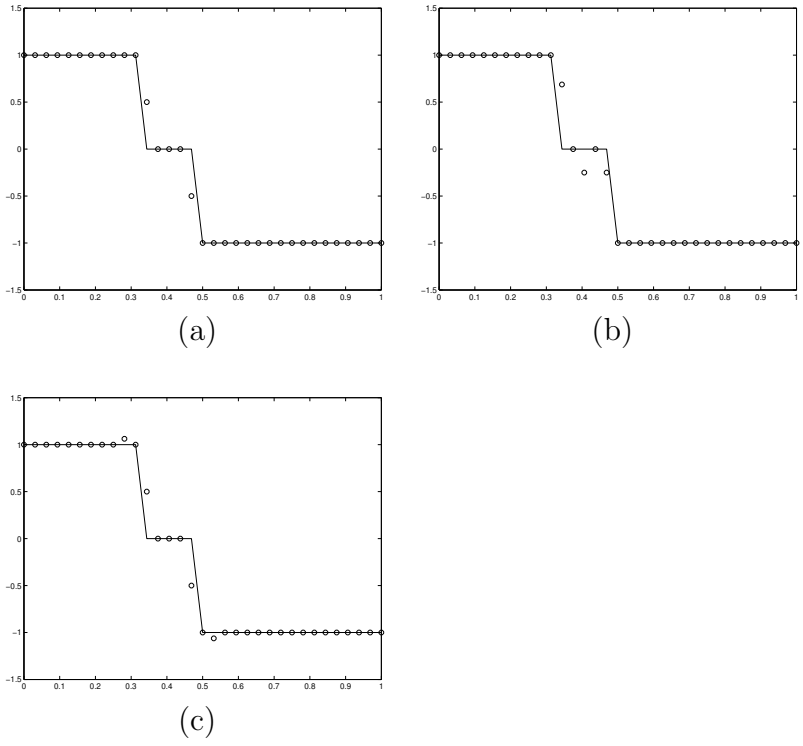


Figura A.2: Interpolação cúbica: (a) ENO, (b) pseudo-ENO, (c) clássica.

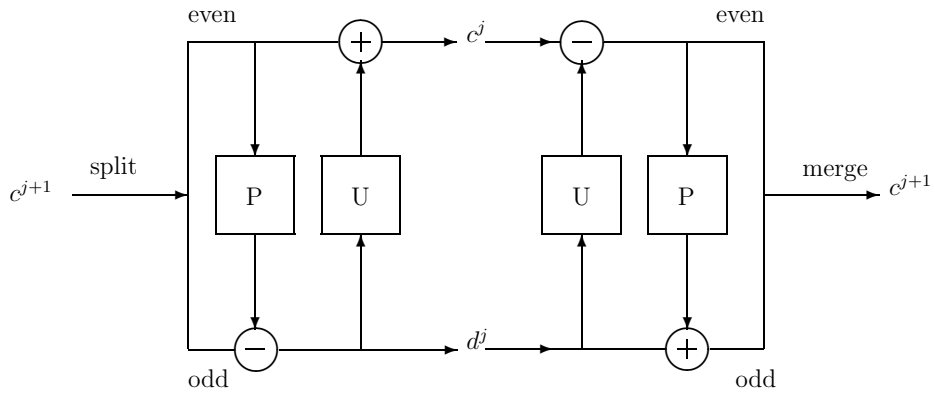


Figura A.3: Ciclo de análise e síntese do *lifting*.

modificará os c^j utilizando os d^j. Na Figura A.3 está representado o banco de filtros que corresponde a esta transformada modificada, que é chamada de *lifting*.

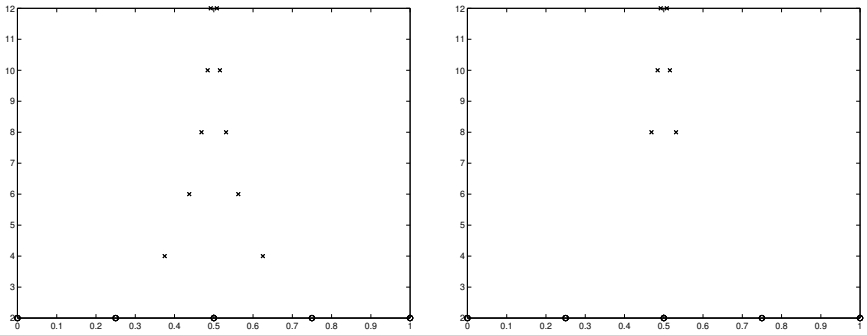
O esquema *lifting* é uma ferramenta para construir wavelets de segunda geração, que são uma generalização da teoria apresentada no Capítulo 2. A base de Riesz de V_j numa análise multirresolução de segunda geração, que seguiremos denotando por $\varphi_{j,k}$, não é necessáriamente formada por dilatações e translações da função $\varphi_{0,0}(x)$.

O objetivo é reduzir o fenômeno conhecido como *aliasing* mediante uma transformada wavelet atualizada (ou seja, um *lifting*). As malhas mais finas contêm os pontos das malhas mais grossas. Quando descontinuidades ou mudanças bruscas da função ocorrem nestes pontos, elas são transmitidas das malhas finas até as mais grossas, no sentido de que quando são calculados os coeficientes $d_{j,k}$ das malhas grossas na vizinhança do ponto, estes serão *não* nulos. Este é o *aliasing* (Figura A.4). O número de momentos nulos da wavelet está relacionado com seu cancelamento e portanto com o *aliasing* transmitido dos níveis mais finos aos grossos.

Se propõe então que

$$\begin{aligned}\varphi(x) &= \varphi^{\text{old}}(x),\\ \psi(x) &= \psi^{\text{old}}(x) - \sum_k u_k \varphi(x-k),\end{aligned}$$

onde os coeficientes u_k são chamados de *update*. Procuram-se momentos



(a) (b)

Figura A.4: Decomposição de um pulso. (a) Fenômeno de *aliasing*; (b) *aliasing* corrigido mediante *lifting*.

nulos, ou seja,

$$\int_{\mathbb{R}} x^p \psi(x)\ dx = 0, \qquad p = 0, 1, \ldots, 2N-1.$$

Então

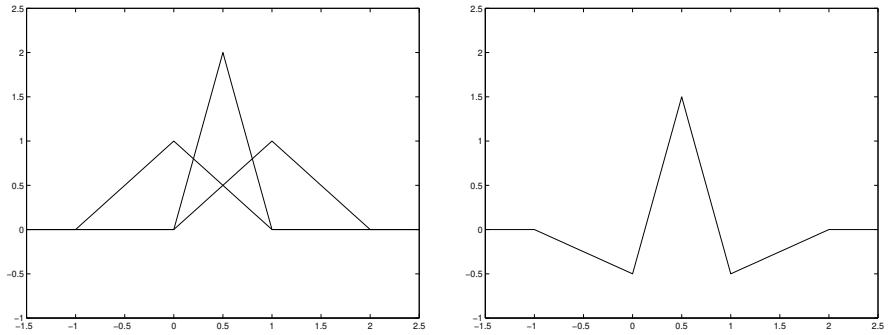
$$\int_{\mathbb{R}} x^p \psi^{\text{old}}(x)\ dx - \sum_k u_k \int_{\mathbb{R}} x^p \varphi(x-k)\ dx = 0, \qquad p = 0, 1, \ldots, 2N-1.$$

Estamos na presença de um sistema linear de $2N$ equações e $2N$ incógnitas para determinar o valor dos coeficientes u_k. De forma matricial:

$$b_p - \sum_k u_k a_{pk} = 0 \quad \text{ou} \quad Au = b,$$

onde $b_p = \int_{\mathbb{R}} x^p \psi^{\text{old}}(x)\ dx$ e $a_{pk} = \int_{\mathbb{R}} x^p \varphi(x-k)\ dx$. É habitual exigir que os coeficientes u_k sejam simétricos (ou seja, $u_{-k} = u_{k+1}$), assim o número de incógnitas fica reduzido à metade. Exemplifiquemos com o caso de $\varphi(x)$ linear, onde

$$\varphi(x) = \begin{cases} 1+x, & x \in [-1, 0), \\ 1-x, & x \in [0, 1], \\ 0, & \text{no restante dos pontos,} \end{cases}$$



(a) (b)

Figura A.5: (a) Funções $\varphi(x)$ à esquerda, $\psi^{\text{old}}(x)$ no centro e $\varphi(x-1)$ à direita; (b) a nova $\psi(x)$.

e $\psi^{\text{old}}(x) = 2\varphi(2x-1)$. Teremos que

$$u_0 = u_1; \qquad b_0 - \sum_k u_k a_{0k} = 0,$$

onde $a_{00} = a_{01} = 1$, $b_0 = 1$, portanto $u_0 = u_1 = \frac{1}{2}$. Na Figura A.5 temos estas funções. Assim se consegue que a nova $\psi(x)$ satisfaça

$$\int_{\mathbb{R}} \psi(x)\, dx = 0; \qquad \int_{\mathbb{R}} x\psi(x)\, dx = 0.$$

Apêndice B

Códigos implementados

B.1 Implementação do esquema ENO

Os dois esquemas ENO (quadrático e cúbico) são básicamente iguais, e utilizam portanto a mesma árvore de dependência:

$$\texttt{eno2r}, \texttt{eno3r} \begin{cases} \texttt{difdiv} \left\{ \texttt{difdiv} \right. \\ \texttt{minmod} \\ \texttt{where\_minmod} \left\{ \texttt{minmod} \right. \end{cases}$$

A função `difdiv` é recursiva. Ver nas Figuras A.1 e A.2 os resultados da aplicação destes programas.

Função `difdiv`**:** Calcula recursivamente as diferenças divididas de qualquer ordem dado um vetor de pontos da malha e os dados associados.

```
function [gamma] = difdiv(x)

n = length(x);

if n == 1
   gamma = f(x);
else
   x1 = x(1:n - 1);
   x2 = x(2:n);
   gamma = (difdiv(x2) - difdiv(x1))/(x(n) - x(1));
end
```

Função `minmod`**:** Dados dois valores, decide qual é o menor em módulo se os dois têm o mesmo sinal. Se o sinal deles é diferente adota o valor zero.

```
function [res] = minmod(x,y)

if sign(x) == sign(y)
   res = sign(x)*min(abs(x),abs(y));
else
   res = 0;
end
```

Função `where_minmod`**:** Determina se o valor de `minmod` corresponde ao primeiro ou ao segundo dos valores comparados.

```
function [where] = where_minmod(x,y)

s = minmod(x,y);

if s == x
   where = 0;
else
   where = 1;
end
```

Função `eno2r`**:** Interpola os pontos de `x` mediante o esquema ENO quadrático.

```
function [fint] = eno2r(x)

x1 = x(1); xn = x(length(x)); nivel = log2(length(x) - 1);

N = 2^(nivel + 1) + 1; xint = linspace(x1,xn,N);

for i=1:2:N,
   fint(i) = f(xint(i));
end

%Extremo izquierdo
```

```
fint(2) = difdiv(xint(1)) + ...
   difdiv([xint(1) xint(3)])*(xint(2) - xint(1)) + ...
   difdiv([xint(1) xint(3) xint(5)]) * ...
   (xint(2) - xint(1)) * (xint(2) - xint(3));

%Centro
for i=4:2:N - 3,
   gl = difdiv([xint(i - 1) xint(i + 1) xint(i - 3)]);
   gr = difdiv([xint(i - 1) xint(i + 1) xint(i + 3)]);

   fint(i) = difdiv(xint(i - 1)) + ...
      difdiv([xint(i-1) xint(i+1)])*(xint(i)-xint(i-1))+...
      minmod(gl,gr)*(xint(i)-xint(i-1))*(xint(i)-xint(i+1));
end

%Extremo derecho
fint(N - 1) = difdiv(xint(N)) + ...
   difdiv([xint(N) xint(N-2)]) * (xint(N-1)-xint(N)) + ...
   difdiv([xint(N) xint(N - 2) xint(N - 4)]) * ...
   (xint(N - 1) - xint(N)) * (xint(N - 1) - xint(N - 2));
```

Função `eno3r`**:** Interpola os pontos de `x` mediante o esquema ENO cúbico.

```
function [fint] = eno3r(x)

x1 = x(1); xn = x(length(x)); nivel = log2(length(x) - 1);

N = 2^(nivel + 1) + 1; xint = linspace(x1,xn,N);

for i=1:2:N,
   fint(i) = f(xint(i));
end

%Extremo izquierdo
fint(2) = difdiv(xint(1)) + ...
   difdiv([xint(1) xint(3)])*(xint(2) - xint(1)) + ...
   difdiv([xint(1) xint(3) xint(5)])*...
   (xint(2) - xint(1))*(xint(2) - xint(3)) + ...
```

```
   difdiv([xint(1) xint(3) xint(5) xint(7)])*...
   (xint(2)-xint(1))*(xint(2)-xint(3))*(xint(2)-xint(5));

%Segundo punto a la izquierda
gl = difdiv([xint(3) xint(5) xint(1)]);
gr = difdiv([xint(3) xint(5) xint(7)]);

fint(4) = difdiv(xint(3)) + ...
   difdiv([xint(3) xint(5)])*(xint(4)-xint(3))+...
   minmod(gl,gr)*(xint(4)-xint(3))*(xint(4)-xint(5));

if where_minmod(gl,gr) == 0
   fint(4) = fint(4) +...
      difdiv([xint(3) xint(5) xint(1) xint(7)])*...
      (xint(4)-xint(3))*(xint(4)-xint(5))*(xint(4)-xint(1));
else
   g3l=difdiv([xint(3) xint(5) xint(7) xint(1)]);
   g3r=difdiv([xint(3) xint(5) xint(7) xint(9)]);
   fint(4)=fint(4) + minmod(g3l,g3r)*...
      (xint(4)-xint(3))*(xint(4)-xint(5))*(xint(4)-xint(7));
end

%Centro
for i = 6:2:N - 5
   gl = difdiv([xint(i - 1) xint(i + 1) xint(i - 3)]);
   gr = difdiv([xint(i - 1) xint(i + 1) xint(i + 3)]);

   fint(i) = difdiv(xint(i - 1)) + ...
      difdiv([xint(i-1) xint(i+1)])*(xint(i)-xint(i-1))+...
      minmod(gl,gr)*(xint(i)-xint(i-1))*(xint(i)-xint(i+1));

   if where_minmod(gl,gr) == 0
      g3l=difdiv([xint(i-1) xint(i+1) xint(i-3) xint(i-5)]);
      g3r=difdiv([xint(i-1) xint(i+1) xint(i-3) xint(i+3)]);
      fint(i) = fint(i) + minmod(g3l,g3r)*...
         (xint(i)-xint(i-1))*(xint(i)-xint(i+1))*...
         (xint(i)-xint(i-3));
   else
```

```
      g3l=difdiv([xint(i-1) xint(i+1) xint(i+3) xint(i-3)]);
      g3r=difdiv([xint(i-1) xint(i+1) xint(i+3) xint(i+5)]);
      fint(i) = fint(i) + minmod(g3l,g3r)*...
         (xint(i)-xint(i-1))*(xint(i)-xint(i+1))*...
         (xint(i)-xint(i+3));
   end
end

%Segundo punto a la derecha
gl = difdiv([xint(N-4) xint(N-2) xint(N-6)]);
gr = difdiv([xint(N-4) xint(N-2) xint(N)]);

fint(N-3) = difdiv(xint(N-4)) + ...
   difdiv([xint(N-4) xint(N-2)])*(xint(N-3)-xint(N-4))+...
   minmod(gl,gr)*(xint(N-3)-xint(N-4))*...
   (xint(N-3)-xint(N-2));

if where_minmod(gr,gl) == 0
   fint(N-3) = fint(N-3) +...
      difdiv([xint(N-4) xint(N-2) xint(N) xint(N-6)])*...
      (xint(N-3)-xint(N-4))*(xint(N-3)-xint(N-2))*...
      (xint(N-3)-xint(N));
else
   g3l=difdiv([xint(N-4) xint(N-2) xint(N-6) xint(N-8)]);
   g3r=difdiv([xint(N-4) xint(N-2) xint(N-6) xint(N)]);
   fint(N-3)=fint(N-3) + minmod(g3l,g3r)*...
      (xint(N-3)-xint(N-4))*(xint(N-3)-xint(N-2))*...
      (xint(N-3)-xint(N-6));
end

%Extremo derecho
fint(N - 1) = difdiv(xint(N)) + ...
   difdiv([xint(N) xint(N - 2)])*(xint(N - 1) - xint(N)) + ...
   difdiv([xint(N) xint(N - 2) xint(N - 4)])*...
   (xint(N - 1) - xint(N))*(xint(N - 1) - xint(N - 2)) + ...
   difdiv([xint(N) xint(N - 2) xint(N - 4) xint(N - 6)])*...
   (xint(N-1)-xint(N))*(xint(N-1)-xint(N-2))*...
   (xint(N-1)-xint(N-4));
```

B.2 Implementação do esquema wavelet interpolador

Estas funções foram criadas de tal forma que, se desejado, seja possível modificar uma ou várias delas sem afetar as outras. Repetimos assim o esquema de filtros descrito na seção 2.5 e representado na Figura 2.7. A árvore de dependência das funções é:

$$\texttt{analisis} \begin{cases} \texttt{split} \\ \texttt{interpolar} \end{cases}$$

$$\texttt{sintesis} \begin{cases} \texttt{interpolar} \\ \texttt{merge} \end{cases}$$

Função `split`: Dado o vetor `C` com $2^k + 1$ elementos ($k \in \mathbb{Z}_+$), seus elementos ímpares são armazenados em `c`, e os pares em `d`.

```
function [c,d] = split(C)

c = C(1:2:length(C));
d = C(2:2:length(C) - 1);
```

Função `merge`: Dados dois vetores `a` e `b`, sendo que o primeiro tem um elemento a mais que o segundo, seus elementos são intercalados e o resultado armazenado em `res`.

```
function [res] = merge(a,b)

res(1:2:2*length(a) - 1) = a;
res(2:2:2*length(a) - 2) = b;
```

Função `interpolar`**:** Dado o vetor `x`, com pelo menos p elementos, é gerado o vetor `res`, cujos elementos são o valor interpolado dos elementos de `x`. A interpolação é de grau $p-1$.

```
function [res] = interpolar(x)

%Interpolaci\'on lineal
%
%for i = 1:length(x) - 1
%    res(i) = (x(i) + x(i + 1))/2;
%end

%Interpolaci\'on c\'ubica
%
N = length(x);
pizq = [5 15 -5 1];
yizq = [x(1); x(2); x(3); x(4)];
res(1) = pizq*yizq/16;

pcentro = [-1 9 9 -1]/16;
for i = 2:N - 2
   ycentro = [x(i - 1); x(i); x(i + 1); x(i + 2)];
   res(i) = pcentro*ycentro;
end

pder = [1 -5 15 5];
yder = [x(N - 3); x(N - 2); x(N - 1); x(N)];
res(N - 1) = pder*yder/16;
```

Função `analisis`**:** Dado o vetor `C` com 2^k+1 elementos ($k \in \mathbb{Z}_+$), é feita a análise dele, e gerados `c` e `d`.

```
function [c,d] = analisis(C)

[c,d] = split(C);
d = d - interpolar(c); %P
```

Função `sintesis`**:** Dados os vetores `c` e `d`, é realizada a síntese deles, e o resultado é armazenado em `C`.

```
function [C] = sintesis(c,d)

d = d + interpolar(c); %P
C = merge(c,d);
```

B.3 Implementação do esquema *lifting*

O esquema *lifting* também foi implementado em forma de várias funções programadas em MATLAB. A aplicação do *update* é obtida computacionalmente de forma simples, acrescentando apenas uma linha de código (comparando as funções `analisis` e `analisis_lifting`). Para exemplificar, utilizamos interpolação linear, como na seção A.2. A árvore de dependência das funções é:

$$\texttt{analisis\_lifting} \begin{cases} \texttt{analisis} \\ \texttt{update} \end{cases}$$

$$\texttt{sintesis\_lifting} \begin{cases} \texttt{update} \\ \texttt{sintesis} \end{cases}$$

As funções `analisis` e `sintesis` já foram descritas na seção B.2.

Função `update`**:** Dados os vetores `c` e `d` e o escalar `signo`, é feita a atualização de `c`, e armazenada em `res`.

```
function [res] = update(c,d,signo)

N = length(c); u = [1/2 1/2];

for i = 2:N - 1
    y = [d(i - 1); d(i)];
    c(i) = c(i) + signo*u*y;
end

res = c;
```

Função `analisis_lifting`**:** Dado o vetor `C` é feita a análise dele, e o vetor `c` é atualizado.

```
function [c,d] = analisis_lifting(C)

[c,d] = analisis(C); c = update(c,d,1);
```

Função `sintesis_lifting`**:** Dado o vetor `c`, é obtido o vetor original (sem atualização) e sintetizado com `d`. O resultado é armazenado em `C`.

```
function [C] = sintesis_lifting(c,d)

c = update(c,d,-1); C = sintesis(c,d);
```

B.4 Decomposição wavelet

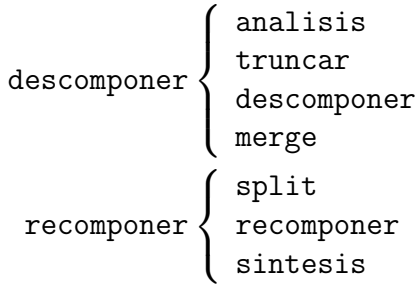
Para decompor e recompor uma função (mais exatamente, um vetor cujos elementos são os valores de uma função numa malha diádica) mediante técnicas wavelet, foram criadas as seguintes funções de MATLAB. Apresentamos dois grupos de códigos. No primeiro temos a decomposição e recomposição de uma função sem aproveitar o caráter esparso desta representação. No segundo, tiramos proveito dele.

A quantidade de elementos que são armazenados é bem menor no caso da decomposição esparsa, porque só preservamos os detalhes significativos e o valor da função na malha mais grossa, ao passo que na decomposição não esparsa são armazenados, também, os pontos onde os detalhes são não significativos (este é o preço a pagar por um algoritmo simples).[1] Os programas apresentados em [9] são do primeiro tipo (armazenam todos os pontos). Os programas utilizados pelo esquema adaptativo descrito na seção 3.3 são baseados numa estrutura de dados verdadeiramente esparsa, o que reduz a quantidade de dados armazenados (e a quantidade de operações efetuadas, se são armazenados todos os dados é preciso determinar primeiro se o dado representa um ponto significativo ou não).

[1] Na seção 2.8 é apresentada a quantidade de pontos necessários para representar uma função com uma determinada qualidade; verifica-se que uma representação satisfatória pode precisar de menos do 1% dos pontos originalmente utilizados para representar tal função. Ou seja, partindo de uma malha regular muito fina, que permite representar a função com muita resolução, é possível (e desejável) obter uma malha esparsa, tipicamente com poucos pontos, mas suficientes para representar a função com boa qualidade.

B.4.1 Decomposição e recomposição não esparsas

A árvore de dependência das funções é:



As funções `descomponer` e `recomponer` são recursivas. As funções `split`, `merge`, `analisis` e `sintesis` já foram descritas na seção B.2.

Função `truncar`**:** Dado o vetor de detalhes `d` e o limitante `epsilon`, os elementos de `d` são truncados, ou seja, aqueles que são menores que `epsilon` são anulados.

```
function [res] = truncar(d,epsilon)

for i = 1:length(d)
   if (abs(d(i))) >= epsilon
      res(i) = d(i);
   else
      res(i) = 0;
   end
end
```

Função `descomponer`**:** Dados o vetor `c` (com $2^k + 1$ elementos, $k > 1$), o escalar `n` $= 2^k + 1, k > 1$ (a quantidade de elementos da malha mais grossa) e o limitante `epsilon`, é obtida a decomposição de `c`, ou seja, é efetuada a análise do vetor `c` e os detalhes são truncados. Em seguida é efetuada a análise do novo vetor `c`, até chegar ao nível desejado, aquele que tem `n` elementos. O nível mais grosso não pode ter mais elementos do que o mais fino. Os detalhes são intercalados com o novo vetor `c`, ou seja, nos pontos onde são calculados os detalhes, o valor da função é sustituido pelo detalhe correspondente.

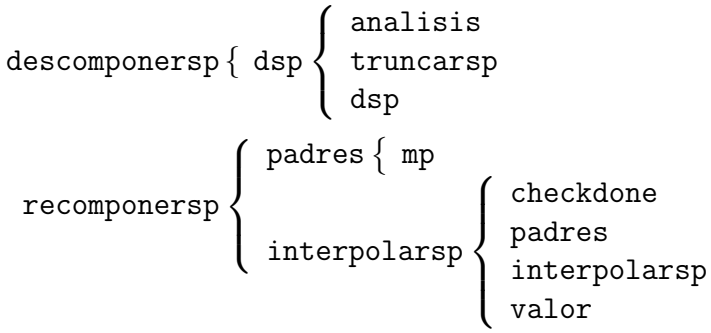
```
function [res] = descomponer(c,n,epsilon)

if length(c) == n
   res = c;
else
   [c,d] = analisis(c);
   d = truncar(d,epsilon);

   c = descomponer(c,n,epsilon);
   res = merge(c,d);
end
```

Função `recomponer`**:** Dados o vetor `c` e `n` (a quantidade de elementos do nível mais grosso), é obtido o vetor `res`, que contém os valores da função reconstruida. A função é reconstruida por níveis, desde o mais grosso até o mais fino, com ajuda dos detalhes significativos. Onde o detalhe foi zerado, a interpolação não é corregida. Os resultados de recompor uma função após ter truncados os detalhes para diferentes valores de ε, encontram-se na Figura 2.10.

```
function [res] = recomponer(c,n)

if length(c) == n
   res = c;
else
   [c,d] = split(c);

   c = recomponer(c,n);
   res = sintesis(c,d);
end
```

B.4.2 Decomposição e recomposição esparsas

A árvore de dependência das funções é:



As funções `dsp` e `interpolarsp` são recursivas. Nas funções a seguir, `N` é a quantidade de pontos da malha mais fina e `n` é a quantidade de pontos na malha mais grossa. Ambos os escalares são da forma $2^k + 1, k > 1$, e $\texttt{N} \geq \texttt{n}$. A função `analisis` já foi descrita na seção B.2.

Função `truncarsp`**:** Dados o vetor de detalhes `d` e os escalares `epsilon` e `c1`, é efetuado o truncamento de `d`. Cumpre-se que $\texttt{c1} = \log_2 \frac{N_\texttt{c}-1}{\texttt{N}-1}$, onde $N_\texttt{c}$ é a quantidade de pontos do nível que está sendo analizado. Portanto, o oposto de `c1` é a "distância" entre o nível atual e o nível mais fino. O resultado desta função, o vetor `md`, só contém os detalhes significativos, e tem duas linhas. Na primeira são colocadas as posições *na malha mais fina* dos pontos que estão sendo analizados. Na segunda linha são colocados os detalhes.

```
function [md] = truncarsp(d,epsilon,c1)

k = 0; md = [];
for i = 1:length(d)
   if abs(d(i)) >= epsilon
      k = k + 1;
      md = [md [1 + 2^(-c1) + (i - 1)*2^(-c1 + 1);d(i)]];
   end
end
```

Função `dsp`**:** Esta função é o motor de `descomponersp`, e devolve dois resultados. O primeiro é o vetor `res`, que tem três linhas. Na primeira é armazenada a distância entre o nível mais fino e o nível onde está o ponto, cuja posição na malha mais fina é armazenada na segunda linha. Na terceira linha é armazenado o valor da função (se for atingido o nível mais grosso) ou o detalhe (são armazenados apenas os detalhes significativos) se ainda não foi atingido o nível mais grosso. Entre os argumentos de `dsp` está `c`. Primeiro `dsp` confere se este vetor corresponde ao nível mais grosso que se deseja atingir. Nesse caso, todos os pontos de `c` devem ser pré-acrescentados ao vetor `ab`. Se não for o caso, é efetuada a análise do vetor `c`, truncados os detalhes e, se o vetor `md` não for vazio (ou seja, se existirem detalhes significativos no nível analizado), este é pós-acrescentado ao vetor `ab`. Este processo é repetido até ser atingido o nível mais grosso, e `ab` é atualizado em cada um deles. O segundo resultado de `dsp` é `c1`, o oposto da distância entre o nível atualmente sob análise e o mais fino.

```
function [res,c1] = dsp(ab,c,N,n,epsilon,contador)

c1 = contador - 1;

if length(c) == n
   res = [[-c1*ones(1,n);(1:(N - 1)/(n - 1):N);c] ab];
else
   [c,d] = analisis(c);
   md = truncarsp(d,epsilon,c1);
   if ~isempty(md)
      ab = [ab [-c1*ones(1,length(md(1,:)));md]];
   end

   [res,c1] = dsp(ab,c,N,n,epsilon,c1);
end
```

Função `descomponersp`**:** Esta função é a interface que permite decompor de forma esparsa o vetor original `c`, invocando o motor `dsp`, que é alimentado inicialmente com `ab` vazio. O resultado é o vetor `ab`, com o formato herdado de `dsp`.

```
function [ab] = descomponersp(c,n,epsilon)

N = length(c); contador = 1;

[ab,contador] = dsp([],c,N,n,epsilon,contador);
```

As funções que permitem recompor a função a partir de `ab` são:

Função `mp`**:** Dado `x` devolve a maior potência de 2 que o divide. Por exemplo, `mp`$(9) = 0$; `mp`$(20) = 2$.

```
function [res] = mp(x)

k = -1; z = 1;

while x/z == floor(x/z)
   z = z*2; k = k + 1;
end

res = k;
```

Função `padres`**:** Dado o vetor `punto` (que contém, em duas linhas, a distância entre o nível a que um ponto pertece e o nível mais fino, e sua posição na malha original), devolve seus "pais", ou seja, aqueles pontos cujos valores, devidamente interpolados, devolvem o valor da função na posição armazenada em `punto`. Também são devolvidas as distâncias entre o nível de cada pai e o nível mais fino, e o escalar `shift`, que indica se `punto` deve ser interpolado com pontos que pertecem aos extremos ou ao centro do domínio.

```
function [y,shift] = padres(punto,N,n)

nivel = punto(1);
```

```
pos = punto(2);
lag = 2^nivel;

y = [(pos - lag) (pos + lag)];

if (pos - 3*lag) < 1
   y = [y (pos + 3*lag) (pos + 5*lag)];
   shift = -1;
elseif (pos + 3*lag) > N
   y = [(pos - 5*lag) (pos - 3*lag) y];
   shift = 1;
else
   y = [(pos - 3*lag) y (pos + 3*lag)];
   shift = 0;
end

for i = 1:4
   a = (y(i) - 1)*(n - 1)/(N - 1);
   if a == fix(a)
      niv(i) = log2((N - 1)/(n - 1));
   else
      niv(i) = mp(y(i) - 1);
   end
end

y = [niv;y];
```

Função `checkdone`**:** Dados o vetor `done` (que armazenará os valores da função já calculados) e a posição `y`, esta função determina se o valor da função num ponto já é conhecido. Se for o caso, o valor é devolvido.

```
function [y_temp,yes] = checkdone(done,y)

for i = 1:length(done)
   if y == done(1,i)
      y_temp = done(2,i);
      yes = 1;
      return
```

```
   end
end

y_temp = 0; yes = 0;
```

Função `valor`**:** Dada a posição de um ponto na malha mais fina, é conferido se ele contém um detalhe significativo. Sendo o caso, seu valor é devolvido.

```
function [v] = valor(x,y)

for i = 1:length(x)
   if y == x(2,i)
      v = x(3,i);
      return
   end
end

v = 0;
```

Função `interpolarsp`**:** O motor de `recomponersp`. Dado um ponto `y`, são achados seus pais, interpolado o valor da função nele e corrigido com o detalhe correspondente, se existir. Cada pai é analizado. Se o valor da função nele já foi calculado, então é utilizado, caso contrário é repetido o processo. Desta forma cada pai cujo valor é desconhecido gera uma árvore de pais, e `interpolarsp` é o encarregado de determinar o valor da função neles. Finalmente, é devolvido um escalar, `z`, que contém o valor da função no ponto dado, e o vetor `done` com aqueles pontos onde o valor da função já foi calculado e o próprio valor.

```
function [z,done] = interpolarsp(x,y,shift,N,n,ready)

if shift == -1
   c = [5 15 -5 1];
elseif shift == 0
   c = [-1 9 9 -1];
else
   c = [1 -5 15 5];
```

```
end

done = ready;

for i = 1:4
   [y_temp,yes] = checkdone(done,y(2,i));
   if yes == 1
      y_t(i) = y_temp;
   else
      [y2,s2] = padres([y(1:2,i)],N,n);
      [y_t(i),done] = interpolarsp(x,y2,s2,N,n,done);
      y_t(i) = y_t(i) + valor(x,y(2,i));
      done = [done [y(2,i);y_t(i)]];
   end
end

z = c*y_t'/16;
```

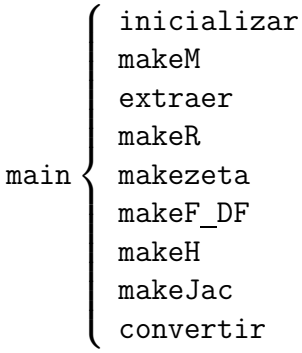
Função `recomponersp`**:** A interface que permite, dado um vetor `x`, no formato devolvido por `descomponersp`, reconstruir a função apenas naqueles pontos onde os detalhes foram significativos. Neles são determinados os pais e o valor da função é interpolado a partir deles e corrigido. São devolvidos também os valores funcionais nos pontos da malha mais grossa. Esta função devolve os valores funcionais e sua posição na malha original. Na Figura 2.11 mostra-se a diferença entre a interpolação não esparsa e a esparsa.

```
function [malla,vector] = recomponersp(x,N,n)

vector = x(3,1:n); malla = x(2,1:n);
ready = x(2:3,1:n);

if length(x) > n
   for i = n + 1:length(x)
      [y,shift] = padres(x(1:2,i),N,n);
      [z,ready] = interpolarsp(x,y,shift,N,n,ready);
      z = z + x(3,i)
```

```
        vector = [vector z];
        malla = [malla x(2,i)];
    end
end
```

B.5 Implementação do esquema implícito

Foram criados programas de MATLAB. A árvore de depêndencia é:



Programa `main`**:** Programa principal, que efetua as iterações de Newton e traça os resultados.

```
%begin{main}

clear all
inicializar
makeM

while tau <= tau_f
    extraer
    makeR
    makezeta

    newtonit = 0;
    Vm_ant = 2*Vm;

    while max(abs(Vm - Vm_ant)) >= epsilon & newtonit < 11
        newtonit = newtonit + 1
```

```
        Vm_ant = Vm;
        extraer
        makeF_DF
        makeH
        makeJac
        Vm = Vm_ant + sparse((M - Jac_H))\...
           (R - M*Vm_ant + H);
    end

    tau = tau + Delta

end

extraer
convertir

plot(x,T,'bo')
hold on
plot(x,Ts,'rx')

figure
plot(x,Y2,'kd')
hold on
plot(x,Y1,'gs')

%end{main}
```

Programa `inicializar`**:** Neste programa são inicializadas as diversas constantes que serão usadas ao longo do processo, assim como o vetor com os dados iniciais, U^0. Todas as magnitudes físicas têm unidades do MKS.

```
%begin{inicializar}

%Constantes
P = 1500000; Tb = 400; Ts = 400; Rgas = 8.31451;

H1 = -4.41d6; cp = 2.66d3; L = 2.5/100; G = 17.5;
```

```
mu_visc = .256d-4; D1 = .8825d-5; D2 = 1.579d-5;
k1_2 = (D1/D2)^(2/3);

alpha = .55; beta = (1.41 + 2.61)/2;
mu = .824; omega = .36464;

r = .3/1000; A = 6787;

Re = G/(mu_visc*A);
Lc = Re^.41/(.74*A); h = .74*cp*G*Re^(-.41);

Z = 155; D = .0195;

Nx = 100; x = [0:L/Nx:L]'; z = x/Lc; delta = L/(Nx*Lc);
Delta = .1; tau = 0; tau_f = 5; epsilon = .1;

%Gases
Y1b = .87 - .0006*Tb;
Y3b = (1 - Y1b)/32; Y2b = 17*Y3b; Y4b = 14*Y3b;

%Conds. contorno
Theta_contour = 0;
psi_contour = 32*Y2b/(17*Y1b*k1_2);

%Cond. inicial
Theta_s = zeros(size(z));

%Vectores
phi = exp(-alpha*z); Y1 = Y1b*phi;
Theta_s = zeros(size(z)); Theta = Theta_s;
psi = (1 - phi)*mu/alpha + psi_contour;
Vm = zeros(3*Nx + 1,1);
Vm(1) = Theta_s(1); Vm(3:3:3*Nx) = Theta_s(2:Nx + 1);
Vm(4:3:3*Nx + 1) = psi(2:Nx + 1);

%end{inicializar}
```

Programa `makeM`**:** Gera a matriz M.

```
%begin{makeM}
d(1) = 1 + Delta;

%Diagonal principal
for i = 1:Nx
    d(3*i - 1) = 1 + delta;
    d(3*i)     = 1 + Delta;
    d(3*i + 1) = 1 + mu*delta;
end

%Diagonal superior
for i = 1:Nx
    dup1(3*i - 2) = -delta;
end
dup1 = [0 dup1 0];

%Primera diagonal inferior
for i = 1:Nx
    ddown1(3*i - 2) = -Delta;
end
ddown1 = [0 ddown1 0];

%Tercera diagonal inferior
for i = 1:Nx - 1
    ddown3(3*i - 2) = -1;
    ddown3(3*i) = -1;
end
ddown3 = [0 ddown3];

M = diag(d,0) + diag(dup1,1) + diag(ddown1,-1)...
    + diag(ddown3,-3);
%end{makeM}
```

Programa `extraer`**:** Extrai os vetores Θ, Θ_s e ψ a partir de U^m.

```
%begin{extraer}

Theta = [Theta_contour; Vm(2:3:3*Nx - 1)];
Theta_s = [Vm(1); Vm(3:3:3*Nx)];
psi = [psi_contour; Vm(4:3:3*Nx + 1)];

%end{extraer}
```

Programa `makeR`**:** Gera o vetor $R(U^m)$.

```
%begin{makeR}
R = zeros(3*Nx,1);
R(2:3:3*Nx - 1) = Theta_s(2:Nx + 1);

R = [Theta_s(1); R];
%end{makeR}
```

Programa `makezeta`**:** Gera um vetor contendo os valores de ζ ao longo do leito catalítico.

```
%begin{makezeta}

T = Tb - H1*Y1b*Theta/cp;
Y2 = 17*k1_2*psi*Y1b/32;
Y3 = (1 - Y1)/8 - 3*Y2/17;
Y4 = 7*(1 - Y1)/8 - 14*Y2/17;

zeta = (Tb*(Y3b/2 + Y4b/28 + Y2b/17 + Y1b/32))./...
    (T.*(Y3/2 + Y4/28 + Y2/17 + Y1/32));

%end{makezeta}
```

Programa `makeF_DF`**:** Gera vetores contendo os valores de $F(\Theta_s)$ e $F'(\Theta_s)$ ao longo do leito catalítico.

```
%begin{makeF_DF}

F = D*exp(-Z*(alpha - Theta_s)./...
    ((1 + beta*alpha)*(1 + beta*Theta_s)));
DF =Z*F./((1 + beta*Theta_s).^2);

%end{makeF_DF}
```

Programa `makeH`**:** Gera o vetor $H(U^{m+1})$.

```
%begin{makeH}
H = zeros(3*Nx,1);
H(2:3:3*Nx - 1) = Delta*alpha*phi(2:Nx + 1) - ...
    alpha*omega*Delta*zeta(2:Nx + 1).*(psi(2:Nx + 1) + ...
    phi(2:Nx + 1)).*F(2:Nx + 1)./(1 + zeta(2:Nx + 1).*...
    F(2:Nx + 1));
H(3:3:3*Nx) = mu*delta*(psi(2:Nx + 1) + phi(2:Nx + 1))./...
    (1 + zeta(2:Nx + 1).*F(2:Nx + 1));

H(1) = Theta_contour;
H(3) = H(3) + psi_contour;
H = [Delta*Theta_contour + Delta*alpha*phi(1) - ...
    Delta*alpha*omega*zeta(1)*(psi(1) + phi(1))*F(1)/...
    (1 + zeta(1)*F(1)); H];
%end{makeH}
```

Programa `makeJac`**:** Gera a matriz H'.

```
%begin{makeJac}
Jac_H = zeros(3*Nx);

for i = 2:Nx + 1
    J = zeros(3);
    J(2,2) = -alpha*omega*Delta*zeta(i)*(psi(i) + ...
```

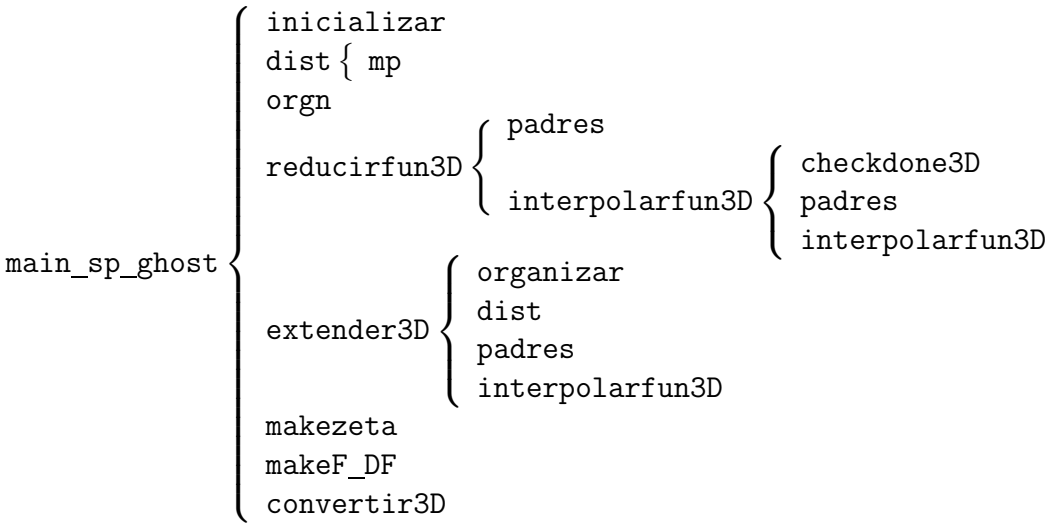
```
        phi(i))*DF(i)/((1 + zeta(i)*F(i))^2);
    J(2,3) = -alpha*omega*Delta*zeta(i)*F(i)/...
        (1 + zeta(i)*F(i));

    J(3,2) = -mu*delta*zeta(i)*(psi(i) + phi(i))*DF(i)/...
        ((1 + zeta(i)*F(i))^2);
    J(3,3) = mu*delta/(1 + zeta(i)*F(i));
    Jac_H(3*(i - 2) + 1:3*(i - 2) + 3,...
        3*(i - 2) + 1:3*(i - 2) + 3) = J;
end

Jac_H = [zeros(1,3*Nx + 1); zeros(3*Nx,1) Jac_H];
%end{makeJac}
```

Programa `convertir`**:** Gera vetores contendo os valores de Y^1, Y^2, T e T_s ao longo do leito catalítico.

```
%begin{convertir}

Y1 = Y1b*phi;
Y2 = 17*k1_2*psi*Y1b/32;
T = Tb + Theta*Y1b*(-H1)/cp;
Ts = Tb + Theta_s*Y1b*(-H1)/cp;

%end{convertir}
```

B.6 Implementação do esquema adaptativo

A árvore de dependência dos programas e funções é:



A função `interpolarfun3D` é recursiva. As funções `mp` e `padres` já foram descritas na seção B.4.2; os programas `inicializar`, `makezeta` e `makeF_DF` já foram descritos na seção B.5.

Função `dist`**:** Dados a posição de um ponto na malha mais fina e a quantidade de pontos das malhas mais fina e mais grossa, é calculada a distância do ponto à malha mais fina. Esta função admite um vetor de posições como argumento.

```
function [res] = dist(p,N,n)

for i = 1:length(p)
    a = (p(i) - 1)*(n - 1)/(N - 1);
    if a == floor(a)
        res(i) = log2((N - 1)/(n - 1));
    else
        res(i) = mp(p(i) - 1);
    end
end
```

Função `orgn`**:** Dados uma matriz `x` e um escalar `n`, a matriz é organizada segundo os valores da sua primeira linha, e as `n` últimas colunas são colocadas ao início da matriz.

```
function [res] = orgn(x,n)

[y,i] = sort(x(1,:));

y = x(:,i); N = length(y);

res = [y(:,N - n + 1:N) y(:,1:N - n)];
```

Função `checkdone3D`**:** Dada uma matriz `done` (que armazena posições na primeira linha e valores das funções Θ, Θ_s e ψ nas restantes três linhas), verifica-se se as funções no ponto com índice `y` já foram calculadas. Sendo o caso, o valor delas é devolvido.

```
function [y_temp,yes] = checkdone3D(done,y)

for i = 1:length(done)
   if y == done(1,i)
      y_temp = done(2:4,i);
      yes = 1;
      return
   end
end

y_temp = 0; yes = 0;
```

Função `interpolarfun3D`**:** O valor das três funções é determinado no ponto cujos pais têm índices armazenados em `y`. O escalar `shift` indica se a interpolação deve ocorrer nos extremos ou no centro do domínio, e `ready` armazena valores funcionais já calculados (evita-se assim ter que recalculá-los). Se algum dos valores funcionais que devem ser utilizados na interpolação não é conhecido, procede-se à sua interpolação, recursivamente.

```
function [z,done] = interpolarfun3D(y,shift,N,n,ready)

if shift == -1
   c = [5 15 -5 1];
elseif shift == 0
   c = [-1 9 9 -1];
else
   c = [1 -5 15 5];
end

done = ready;

for i = 1:4
   [y_temp,yes] = checkdone3D(done,y(2,i));
   if yes == 1
      y_t(:,i) = y_temp;
   else
      [y2,s2] = padres([y(1:2,i)],N,n);
      [y_t(:,i),done] = interpolarfun3D(y2,s2,N,n,done);
      done = [done [y(2,i);y_t(:,i)]];
   end
end

z(1,1) = c*y_t(1,:)'/16;
z(2,1) = c*y_t(2,:)'/16;
z(3,1) = c*y_t(3,:)'/16;
```

Função `reducirfun3D`**:** Dada uma matriz `ab` que armazena na segunda linha as posições dos valores funcionais (armazenados nas três últimas linhas) e na primeira linha a distância destes pontos à malha mais fina, e uma tolerância `epsilon`, é efetuada a redução da malha segundo é descrito na seção 3.3.

```
function [red] = reducirfun3D(ab,N,n,epsilon)

red = ab(1:5,1:n); ready = ab(2:5,:);

for i = n + 1:length(ab)
    [y,shift] = padres(ab(1:2,i),N,n);
    [z,done] = interpolarfun3D(y,shift,N,n,ready);

    if max(abs(z(:,1) - ab(3:5,i))) > epsilon
        red = [red ab(1:5,i)];
    end
end
```

Função `organizar`**:** Dada uma matriz `ab`, suas colunas são organizadas segundo os valores da sua segunda linha.

```
function [res] = organizar(x)

[y,i] = sort(x(2,:));
res = x(:,i);
```

Função `extender3D`**:** Dada uma matriz `ab`, contendo distâncias, posições e valores funcionais nas suas cinco linhas, é estendida a malha por ela representada, havendo necessidade de algum valor funcional, ele é interpolado.

```
function [mh] = extender3D(ab,N,n)

ab = organizar(ab); done = ab(2:5,:);

mh = ab(:,1);
```

```
for i = 1:size(ab,2) - 1
    p = (ab(2,i) + ab(2,i + 1))/2;
    if p == floor(p) %Distancia par
        d = dist(p,N,n);
        [pp,s] = padres([d;p],N,n);
        [f,done] = interpolarfun3D(pp,s,N,n,done);
        mh = [mh [d;p;f] ab(:,i + 1)];
    elseif (ab(2,i + 1) - ab(2,i)) == 1
        mh = [mh ab(:,i + 1)];
    else
        p = floor(p);
        d = dist(p,N,n);
        [pp,s] = padres([d;p],N,n);
        [f,done] = interpolarfun3D(pp,s,N,n,done);
        mh = [mh [d;p;f] ab(:,i + 1)];
    end
end
```

Programa `convertir3D`**:** Converte as três últimas linhas de `ab` nos valores das funções T, T_s e Y^2. Também determina os valores de Y^1 que correspondem aos pontos da malha esparsa.

```
%begin{convertir3D}
Theta = ab(3,:);
Theta_s = ab(4,:);
psi = ab(5,:);

clear phi_sp
for i = 1:size(ab,2)
    phi_sp(i) = phi(ab(2,i));
end

Y1 = Y1b*phi_sp;
Y2 = 17*k1_2*psi*Y1b/32;
T = Tb + Theta*Y1b*(-H1)/cp;
Ts = Tb + Theta_s*Y1b*(-H1)/cp;
%end{convertir3D}
```

Programa `main_sp_ghost`**:** Programa principal. Realiza a evolução temporal do sistema, segundo é descrito na seção 3.3. Para cada passo temporal a malha é reduzida e estendida, e *ghosts* são acrescentados se necessário. Finalmente, gráficos das funções são gerados.

```
%begin{main_sp_ghost}
clear all
inicializar

N = Nx + 1; n = 5; eps_sp = 1d-4;
eps_ghost = 1d-6;

ab = [dist([1:N],N,n);[1:N];Theta';Theta_s';psi'];

while tau < tau_f
    ab = orgn(ab,n);
    ab = reducirfun3D(ab,N,n,eps_sp);
    ab = extender3D(ab,N,n);

    convertir3D
    makezeta
    makeF_DF

    %Actualizaci\'on de $\Theta_s$:
    for i = 1:size(ab,2)
        ab(4,i) = alpha*Delta*phi(ab(2,i)) - alpha*...
            Delta*omega*zeta(i)*F(i)*(psi(i) + ...
            phi_sp(i))/(1 + zeta(i)*F(i)) + Delta*...
            Theta(i) + (1 - Delta)*Theta_s(i);
    end

    %Actualizaci\'on de $\Theta$ y $\psi$:
    ghost_in_use = 1;
    ghost1 = ab(3,:); ghost_old1 = ones(1,size(ab,2));
    ghost2 = ab(5,:); ghost_old2 = ones(1,size(ab,2));

    ab(3,1) = Theta_contour; ab(5,1) = psi_contour;
```

```
    while (max(abs(ghost1 - ghost_old1)) > eps_ghost | ...
            max(abs(ghost2 - ghost_old2)) > eps_ghost) & ...
            ghost_in_use == 1

        ghost_old1 = ghost1; ghost_old2 = ghost2;
        ghost_in_use = 0;
        for i = 2:size(ab,2)
            left = ab(3:5,i - 1);
            delta_sp = delta*(ab(2,i) - ab(2,i - 1));
            if i < size(ab,2)
                if ab(2,i) - ab(2,i - 1) > ab(2,i + 1) - ab(2,i)
                    %disp('Ghost needed!')
                    ghost_in_use = 1;
                    p = ab(2,i) - (ab(2,i + 1) - ab(2,i));
                    d = dist(p,N,n); [pp,s] = padres([d;p],N,n);
                    [left,done] = interpolarfun3D(pp,s,N,n,ab(2:5,:));
                    delta_sp = delta*(ab(2,i + 1) - ab(2,i));
                end
            end

            ab(3,i) = (delta_sp*ab(4,i) + left(1))/(1 + delta_sp);
            ab(5,i) = (left(3) + mu*delta_sp*(psi(i) + phi_sp(i))/...
                (1 + zeta(i)*F(i)))/(1 + mu*delta_sp);
        end
        ghost1 = ab(3,:); ghost2 = ab(5,:);
    end

    tau = tau + Delta
end

ab = orgn(ab,n);
ab = reducirfun3D(ab,N,n,eps_sp);

convertir3D
for i = 1:size(ab,2)
    x_sp(i) = x(ab(2,i));
end
```

```
plot(x_sp,T,'bo')
hold on
plot(x_sp,Ts,'rx')

figure
plot(x_sp,Y2,'kd')
hold on
plot(x_sp,Y1,'gs')

%end{main_sp_ghost}
```

Apêndice C

Magnitudes físicas e parâmetros adimensionais

- Superfície do catalizador por unidade de volume: $A = 6787\,\mathrm{m}^{-1}$.
- Calor específico do gás: $c_p = 2.66 \times 10^3\,\mathrm{J/\,kg\ K}$.
- Número catalítico de Damkohler: $D = 0.019456$.
- Coeficiente de difusão binário da hidrazina: $D^1 = 0.8825 \times 10^{-5}\,\mathrm{m^2/\,s}$.
- Coeficiente de difusão binário da amônia: $D^2 = 1.579 \times 10^{-5}\,\mathrm{m^2/\,s}$.
- Vazão de massa: $G = 17.5\,\mathrm{kg/\,m^2\,s}$.
- Calor de reação com respeito à hidrazina: $H^1 = -4.41 \times 10^6\,\mathrm{J/\,kg}$.
- Coeficiente de transferência de calor: $h = 5198.4\,\mathrm{J/\,m^2\,s\ K}$.
- Razão entre os coeficientes de transferência de massa com respeito à hidrazina e à amônia: $k^1/k^2 = 0.6785$.
- Pressão de operação do reator: $P = 1500\,\mathrm{kPa}$.
- Constante dos gases: $R = 8.31451\,\mathrm{m^2\,kg/\,s^2\,K\ mol}$.
- Raio médio das partículas: $r = 0.30\,\mathrm{mm}$.
- Temperatura de vaporização da hidrazina correspondente à pressão do reator P: $T_b = 400\,\mathrm{K}$.

- Fração de massa da hidrazina no momento em que a temperatura T_b é atingida: $Y_b^1 = 0.63$.
- Número de Zeldovich: $Z = 155$.
- Constante do modelo adimensional: $\alpha = 0.55$.
- Constante do modelo adimensional: $\beta = 2.01$.
- Constante do modelo adimensional: $\mu = 0.824$.
- Viscosidade: $\mu_{\text{visc}} = 0.256 \times 10^{-4}\,\text{kg/ m s}$.
- Constante do modelo adimensional: $\omega = 0.36464$.

Bibliografia

[1] H. Aimar, A. Bernardis, I. Hernández, *Construcción de Bases de Onditas en espacios funcionales a partir de un Análisis Multirresolución. Comparación elemental con el sistema de Fourier*, CIMEC-PEMA, 1998.

[2] A. Cohen, "Wavelets methods in Numerical Analysis" em *Handbook of Numerical Analysis*, vol. VII, P. G. Ciarlet and J. L. Lions, 2000.

[3] A. Crespo, E. Fraga, A. Muñoz "Steady state behavior of hydrazine catalytic thrusters", Reporte AFOSR-TR-75-1238, Instituto Nacional de Técnica Aeroespacial Esteban Torrados, Madrid, 1975.

[4] D. Donoho, "Interpolating Wavelet Transforms", Technical Report 408, Stanford University, 1992.

[5] S. M. Gomes, R. B. Devloo, "Numerical simulation model for hydrazine attitude control thrusters. Part I: Model Description", INPE-4952-RPE/601, 1989.

[6] S. M. Gomes, C. Treviño, "Simulação numérica de modelo transiente de micro-propulsores a hidrazina", ENCIT 88, Águas de Lindóia, SP, 1988.

[7] A. Harten, "Adaptive multirresolution schemes for shock computations", *J. Compt. Phys.* 115: 319–338, 1994.

[8] A. Harten, S. Osher, "Uniformly High-Order Accurate Nonoscillatory Schemes. I" em *SIAM J. Numer. Anal.* Vol. 24, No. 2, abril 1987.

[9] M. Holmström, "Solving Hyperbolic PDEs using Interpolating Wavelets", Technical Report 189, Uppsala University, 1996.

[10] Meyer, Y., *Wavelets. Algorithms & Applications*, SIAM 1993.

[11] V. Shankar, K. Anantha Ram, K. A. Bhaskaran, "Prediction of the concentration of hydrazine decomposition products along a granular catalytic bed", *Acta Astronautica*, Vol. 11, No. 6, pp. 287–299, 1984.

[12] W. Sweldens, P. Scroeder, "Building your own wavelets at home" em *Wavelets in Computer Graphics*, pp. 15–87, ACM, Siggraph course notes, 1996.

[13] O. V. Vasilyev, C. Bowman, "Second generation wavelet collocation method for the solution of partial differential equations", *J. Comput. Phy.* 165, pp. 660–693, 2000.