On the Convergence of Multiwavelet Thresholding *

Lixin Shen and Hwee Huat Tan†
Department of Mathematics
National University of Singapore
2 Science Drive 2, 117543 Singapore
e-mail: {shenlx, mattanhh}@wavelets.math.nus.edu.sg

ABSTRACT
In this paper we introduce a general thresholding operator based on multiwavelets. It offers the flexibility that multiwavelet coefficients generated by different component multiwavelet functions in a multiwavelet vector can be filtered by different thresholding rules. A convergence result of the proposed operator is given. Numerical results for denoising simulation are also presented for showing its robustness.

1 Introduction
Thresholding methods based on wavelet decompositions have been studied in the context of statistics and data compression. Donoho and Johnstone [4] introduced waveshrink which is a method for obtaining nonparametric statistical function estimators based on shrinking of empirical wavelet coefficients. The waveshrink method compares the transform coefficients to a prescribed threshold and sets them to zero if their magnitudes are less than the threshold; otherwise, they are kept or modified depending on the thresholding rule. Waveshrink is thus effective for signals with sparse representation in the transform domain.

Traditionally, the above thresholding algorithms are implemented using scalar wavelets. Let \( \psi \) be an orthonormal scalar wavelet with compactly support. The family of its translates and dilates \( \{ \psi_{j,k} \}_{j,k \in \mathbb{Z}} = \{ 2^{j/2} \psi(2^j \cdot -k) \}_{j,k \in \mathbb{Z}} \) forms an orthogonal basis for \( L^2(\mathbb{R}) \). The thresholding algorithm introduced in [4] can be represented as follows:

\[
T_\lambda f = \sum_{j \in \mathbb{Z}} \sum_{k \in \mathbb{Z}} \eta_\lambda(\langle f, \psi_{j,k} \rangle) \psi_{j,k}
\]  

(1)

where \( \eta_\lambda \) is the thresholding function, \( \lambda \) the prescribed threshold value, and the inner product is defined via

\[
\langle f, g \rangle = \int f(x) g(x) \, dx.
\]

There are two commonly used thresholding functions: (i) hard-threshold function \( \eta_\lambda(x) = x\chi_{|x| > \lambda} \) and (ii) soft-threshold function \( \eta_\lambda(x) = \text{sgn}(x) \max(|x| - \lambda, 0) \), where \( \chi \) is the characteristic function. The corresponding nonlinear approximation operators \( T_\lambda f \) are called the soft thresholding operator and hard thresholding operator respectively. Thresholding essentially allows the data to decide which coefficients are significant and to exploit the local nature of wavelet bases. The hard thresholding is a “keep” or “kill” rule, while the soft thresholding is a “shrink” or “kill” rule.

For the past decade, there have been considerable interests in the studies of multiwavelets [7]. Multiwavelets have more freedoms in their construction and thus can combine more useful properties than the scalar wavelets. It has been shown that multiwavelet bases could perform better in several wavelet applications than other wavelet bases (see [1], [5], [10], [13], [14]). Recently, Downie et al [5] and Strela et al [11] extended Donoho’s denoising method to the multiwavelet case and claimed that multiwavelet based schemes can outperform scalar wavelet schemes for signal denoising. We pursue along this direction and propose a new and general thresholding operator based on multiwavelet decompositions, and in particular we give a convergence result for the proposed operator.

The rest of this paper is organized as follows. In Section 2, we introduce our proposed multiwavelet thresholding operator associated with a multi-resolution analysis with multiplicity \( r \). When \( r = 1 \) the corresponding thresholding operator will reduce to (1). The main result of the pointwise convergence of the proposed thresholding operator is given in Section 3. Results of applying the operator to denoising simulations are presented in Section 4.
follows: a new thresholding (denoising) scheme based on multiwavelet analysis (MRA) \( \{V_j\}_{j \in \mathbb{Z}} \) of multiplicity \( r \) (see [7] for a detailed theoretical treatment of multiwavelet theory). Without loss of generality, we also assume that \( \text{supp}(\phi) = \bigcup_{j=1}^{J_0} \text{supp}(\phi_j) \subset [-N, N] \).

In the following, we always assume that \( \phi_x, \psi_x, \ell = 1, \ldots, r \), are continuous functions on the real line, and thus it is necessary that (see [3])

\[
1 = \sum_{k \in \mathbb{Z}} \sum_{\ell=1}^{r} p_{\ell,k} \phi_x(x-k), \quad \text{for any } x \in \mathbb{R}, \tag{2}
\]

where \( p_{\ell,k} = \int \phi_x(x-k) \, dx \). Obviously \( p_{\ell,0} = p_{\ell,k} \) for any \( k \in \mathbb{Z} \). Equation (2) means that any constant can be reproduced by linear combinations of \( \{\phi_x(x-k)\}_{k \in \mathbb{Z}}, \ell = 1, \ldots, r \), constitutes the orthogonal bases of subspaces \( V_j \), \( j \in \mathbb{Z} \), which form the multi-resolution analysis (MRA) \( \{V_j\}_{j \in \mathbb{Z}} \) of multiplicity \( r \).

In this paper, we propose the following multiwavelet thresholding operator

\[
T_\lambda f = \sum_{j \in \mathbb{Z}} \sum_{k \in \mathbb{Z}} \sum_{\ell=1}^{r} \eta_{\ell, \lambda}(f, \psi_{x,j,k}) \psi_{x,j,k} \tag{3}
\]

where \( \eta_{\ell, \lambda}, \ell = 1, \ldots, r, \) are the thresholding functions, and each can be chosen as either a hard-threshold function or a soft-threshold function, depending on the parameter \( \ell \).

The mixed thresholding operator \( T_\lambda \) defined by (3) is a new thresholding (denoising) scheme based on multiwavelets where multiwavelet coefficients generated by different component multiwavelet functions can be filtered by different thresholding rules. It is different from the scalar case where all wavelet coefficients are filtered by the same thresholding rules [4]. The case of the multiwavelet thresholding operator (3) where the thresholding function \( \eta_{\ell, \lambda} \) is independent of \( \ell \) has been investigated in [5] and [11]. As each component \( \psi_x \) of the vector \( \psi \) may have different compact supports, smoothness, and symmetry properties, it might be advantageous to use different thresholding functions for the \( r \) component multiwavelet functions \( \psi_x \). Such flexibility could lead to improvement in the performance of multiwavelet thresholding applications.

3 Pointwise convergence of the proposed thresholding operator

The main result of this paper is given the following theorem:

Theorem. If \( f \in L^2(\mathbb{R}) \), then for almost every \( x \in \mathbb{R} \), we have

\[
\lim_{\lambda \to 0} (T_\lambda f)(x) = f(x) \tag{4}
\]

where \( T_\lambda f \) is as defined in (3).

Essentially, the above theorem points out that if a solution \( f \) is approximated in terms of an expansion in a multiwavelet basis and that its multiwavelet coefficients are modified by the proposed thresholding operator, then this approximate solution will converge to the true solution as we decrease the value of the threshold \( \lambda \) to zero.

For the special case when \( r = 1 \), i.e. the scalar case, T. Tao [12] has proved that the above theorem holds true. The proof of the stated general case is given in [9].

4 Denoising simulation

To assess the performance and robustness of the proposed thresholding operator for denoising applications, we conducted a simulation study. Sets of test data were generated from the following regression model

\[
y_n = f(x_n) + \epsilon_n, \quad n = 1, 2, \ldots, N, \tag{5}
\]

where \( \epsilon_n \) is the random noise. There are three components associated with this model: the true function \( f \), the error distribution for \( \epsilon_n \), and the design points \( x_n \). In this section, we only consider the cases where \( \epsilon_n \) are independent and identically distributed normal random variables with mean 0 and variance \( \sigma^2 \). We also restrict our test study to uniformly distributed design points \( x_n \). The goal is to recover the underlying function \( f \) from the noisy data \( y_n \), without assuming any particular parameter structure for \( f \).

For each given test function \( f \) on \([a, b]\) and the variance \( \sigma^2 \), we generated \( M \) samples each of size \( N \) from the model (5). For each sample, an estimate \( f_N \) is obtained from the denoising application and the following root mean squared error

\[
rmse = \left( \frac{1}{N} \sum_{n=1}^{N} (f_N(x_n) - f(x_n))^2 \right)^{1/2} \tag{6}
\]

is computed. The average value of \( rmse \) is then obtained from the \( M \) values of \( rmse \) resulted from the trials.

As we could apply either the soft or the hard thresholding rule to the coefficients generated from each of the
Step 1. Preprocess the input data to generate a vector.

Step 2. Apply the discrete multiwavelet transform.

Step 3. Apply a thresholding procedure to the resulting multiwavelet coefficients. The thresholding step was carried out as a vector thresholding operation ([5], [11]). For each thresholding step, all four possible thresholding schemes for \( r = 2 \) are chosen, namely \((H, H), (H, S), (S, H), \) and \((S, S)\). Here, \( H \) denotes hard thresholding while \( S \) denotes soft thresholding. The first position, and the second position correspond to the first and second component of the generated coefficients from the multiwavelet transforms. For example, \((S, H)\) is a scheme which applies soft thresholding rule to the first component, and hard thresholding rule to the second component of the multiwavelet coefficient vectors \((f, \psi_{1,j,k}), (f, \psi_{2,j,k})^T\).

Step 4. Apply the inverse discrete multiwavelet transform.

Step 5. Apply the postprocessing operator to produce a denoised estimate \( \hat{f}_N \) of the input signal.

We adopt the same definition of signal-to-noise ratio (SNR) as used in [4] which has the SNR given by \( \sqrt{\text{var}(f)/\sigma^2} \), i.e., the ratio of signal to noise standard deviations. The average values of \( \text{rmse} \) (6) obtained from our simulation experiments are given in Table 1 for five different noise levels, two preprocessing methods, four different thresholding schemes and for the two stated test functions. For each filter, and for the same SNR and preprocessing method, the best (the least) scheme is indicated with its average \( \text{rmse} \) shown in bold.

The results show that, in general, (i) for higher SNR cases, the ‘pure’ thresholding scheme \((H, H)\) performs better than other three schemes and (ii) repeated row preprocessing is better than critically-sampled preprocessing. In all cases, we see that intrinsic noise was reduced effectively. This is consistent with the findings in [5] and [11]. Moreover, our results also reveal that for lower SNR cases, the ‘mixed’ thresholding schemes have an edge over the \((H, H)\) scheme, and that the ‘pure’ thresholding scheme can be useful for certain low SNR cases too. The results also indicate that the multiwavelet filter SA4 in general outperforms the other two filters GHM and CL.

The above observations were also seen in other simulation studies (not explicitly presented here due to page length consideration) involving another three commonly-used test functions, ‘Doppler’, ‘Blocks’ and ‘Bumps’, and also for results obtained when we replaced the vector thresholding step by the scalar thresholding step throughout.

From the simulation results obtained, one see that the proposed multiwavelet thresholding operator offers the advantage of more choices of thresholding schemes for the same multiwavelet filter used. For an application with a particular type of signal and a certain level of noise, one of these thresholding schemes can be chosen for the denoising task via some test runs prior to the actual implementation.

References


### Table 1: Average values of rmse for denoising the contaminated “Heavisine” test function using three multiwavelet filters with two preprocessing methods: (a) critically-sampled preprocessing and (b) repeated-row preprocessing, over a range of SNR values.

<table>
<thead>
<tr>
<th>Filter</th>
<th>SNR</th>
<th>(H, H)</th>
<th>(H, S)</th>
<th>(S, H)</th>
<th>(S, S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>1</td>
<td>0.7829</td>
<td>0.7698</td>
<td>0.7666</td>
<td>0.7549</td>
</tr>
<tr>
<td></td>
<td>1.5</td>
<td>0.5265</td>
<td>0.5178</td>
<td>0.5154</td>
<td>0.5077</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.3981</td>
<td>0.3943</td>
<td>0.3872</td>
<td>0.3877</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0.2130</td>
<td>0.2101</td>
<td>0.2083</td>
<td>0.2065</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>0.1118</td>
<td>0.1190</td>
<td>0.1138</td>
<td>0.1176</td>
</tr>
<tr>
<td>(b)</td>
<td>1</td>
<td>0.7692</td>
<td>0.7518</td>
<td>0.7554</td>
<td>0.7497</td>
</tr>
<tr>
<td></td>
<td>1.5</td>
<td>0.5178</td>
<td>0.5061</td>
<td>0.5085</td>
<td>0.5047</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.3952</td>
<td>0.3907</td>
<td>0.3826</td>
<td>0.3833</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0.2083</td>
<td>0.2053</td>
<td>0.2075</td>
<td>0.2053</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>0.1074</td>
<td>0.1102</td>
<td>0.1144</td>
<td>0.1136</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.7733</td>
<td>0.7526</td>
<td>0.7567</td>
<td>0.7504</td>
</tr>
<tr>
<td></td>
<td>1.5</td>
<td>0.5207</td>
<td>0.5066</td>
<td>0.5096</td>
<td>0.5050</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.3899</td>
<td>0.3867</td>
<td>0.3818</td>
<td>0.3833</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0.2073</td>
<td>0.2073</td>
<td>0.2077</td>
<td>0.2047</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>0.1092</td>
<td>0.1124</td>
<td>0.1159</td>
<td>0.1145</td>
</tr>
</tbody>
</table>

(a)

<table>
<thead>
<tr>
<th>Filter</th>
<th>SNR</th>
<th>(H, H)</th>
<th>(H, S)</th>
<th>(S, H)</th>
<th>(S, S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>1</td>
<td>0.5537</td>
<td>0.5445</td>
<td>0.5443</td>
<td>0.5413</td>
</tr>
<tr>
<td></td>
<td>1.5</td>
<td>0.3823</td>
<td>0.3757</td>
<td>0.3759</td>
<td>0.3735</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.2998</td>
<td>0.2980</td>
<td>0.2934</td>
<td>0.2924</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0.1711</td>
<td>0.1790</td>
<td>0.1735</td>
<td>0.1780</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>0.0890</td>
<td>0.1056</td>
<td>0.0950</td>
<td>0.1085</td>
</tr>
<tr>
<td>(b)</td>
<td>1</td>
<td>0.5474</td>
<td>0.5391</td>
<td>0.5402</td>
<td>0.5396</td>
</tr>
<tr>
<td></td>
<td>1.5</td>
<td>0.3755</td>
<td>0.3699</td>
<td>0.3707</td>
<td>0.3702</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.2941</td>
<td>0.2922</td>
<td>0.2845</td>
<td>0.2869</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0.1705</td>
<td>0.1728</td>
<td>0.1715</td>
<td>0.1721</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>0.0867</td>
<td>0.0977</td>
<td>0.0986</td>
<td>0.1061</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.5672</td>
<td>0.5484</td>
<td>0.5501</td>
<td>0.5372</td>
</tr>
<tr>
<td></td>
<td>1.5</td>
<td>0.3872</td>
<td>0.3753</td>
<td>0.3761</td>
<td>0.3677</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.2942</td>
<td>0.2928</td>
<td>0.2882</td>
<td>0.2859</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0.1710</td>
<td>0.1704</td>
<td>0.1731</td>
<td>0.1691</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>0.0908</td>
<td>0.0948</td>
<td>0.1043</td>
<td>0.1039</td>
</tr>
</tbody>
</table>

(b)