Journal of Statistical Software

July 2007, Volume 21, Issue 2. *http://www.jstatsoft.org/*

The WaveD Transform in R: Performs Fast Translation-Invariant Wavelet Deconvolution

Marc Raimondo
University of Sydney

Michael Stewart
University of Sydney

Abstract

This paper provides an introduction to a software package called **waved** making available all code necessary for reproducing the figures in the recently published articles on the WaveD transform for wavelet deconvolution of noisy signals. The forward WaveD transforms and their inverses can be computed using any wavelet from the Meyer family. The WaveD coefficients can be depicted according to time and resolution in several ways for data analysis. The algorithm which implements the translation invariant WaveD transform takes full advantage of the fast Fourier transform (FFT) and runs in $O(n(\log n)^2)$ steps only. The **waved** package includes functions to perform thresholding and fine resolution tuning according to methods in the literature as well as newly designed visual and statistical tools for assessing WaveD fits. We give a **waved** tutorial session and review benchmark examples of noisy convolutions to illustrate the non-linear adaptive properties of wavelet deconvolution.

Keywords: WaveD, wavelets, vaguelettes, deconvolution, Meyer wavelet.

1. Introduction

In this paper we present the WaveD transform in R and illustrate some statistical applications of the WaveD transform to the deconvolution of noisy signals. The aim of deconvolution is to recover an unknown function f from a noisy observation of $g * f(t) = \int_T g(t-u)f(u)du$,

$$Y(t) = g * f(t) + \varepsilon\, \xi(t), \quad t \in T = [0,1], \tag{1}$$

where the convolution kernel g is observed with noise (taking $\epsilon = \varepsilon$ below) or without noise (taking $\epsilon = 0$ below),

$$g_\epsilon(t) = g(t) + \epsilon\, \zeta(t), \quad t \in T = [0,1], \tag{2}$$

and ξ, ζ are independent white noises and $0 < \varepsilon, \epsilon < 1$ are noise levels. Both f and g are supposed to be periodic on T and $g * f(t)$ denotes the circular convolution. In the finite sample implementation of model (1) at points $t_i = i/n, i = 1, ..., n$, we let

$$\varepsilon = \sigma/\sqrt{n}, \qquad (A_\varepsilon),$$

where σ is the noise standard deviation and n is the sample size. Let y be a vector with elements $(y_1, ..., y_n)$ where $y_i = Y(t_i)$, $i = 1, \dots, n$. An illustration of model (1) is given in Figure 3 using the test functions of Figure 1. As for model (2) we consider the two cases: (a) $\epsilon = 0$ in which case $g_\epsilon(t) = g(t)$ (known kernel); (b) $\epsilon = \varepsilon = \sigma/\sqrt{n}$ (noisy kernel). We denote g a vector $(g_1, ..., g_n)$ with elements $g_i = g_\epsilon(t_i)$, $i = 1, \dots, n$. An illustration of model (2) in the Fourier domain is given in Figure 8. In its simplest form, the WaveD transform as discussed in this paper requires only y and g as input, other arguments are optional.

1.1. What can the WaveD transform offer?

Over the last decade many wavelet methods have been developed to recover f from indirect observations, among others: Donoho (1995); Abramovich and Silverman (1998); Pensky and Vidakovic (1999); Walter and Shen (1999); Johnstone (1999); Cavalier and Koo (2002); Fan and Koo (2002); Kalifa and Mallat (2003). Applications and general references on deconvolution models may be found in O'Sullivan (1986), Bertero and Boccacci (1998) and Johnstone and Raimondo (2004).

The WaveD transform was first introduced in Johnstone, Kerkyacharian, Picard, and Raimondo (2004) and later refined in Donoho and Raimondo (2004), Kerkyacharian, Picard, and Raimondo (2007) and Cavalier and Raimondo (2007). Some extensions of the WaveD transform to the 2-dimensional setting are discussed in Donoho and Raimondo (2005) and Cavalier and Raimondo (2006).

A specific feature of the WaveD method (when compared with existing wavelet deconvolution methods as listed above) is to address the deconvolution problem in the periodic setting (1) using band-limited wavelets. As a result most of the WaveD computations can be carried out in the Fourier domain. While sharing near-optimal properties with some of the existing wavelet methods listed earlier, we list below some features which are specific to the WaveD method:

- The fast algorithm which implements the translation invariant version of WaveD takes full advantage of the Fast Fourier Transform and is computed in $O(n(\log n)^2)$ steps.
- The WaveD method is easy to use with only two tuning parameters required.
- The WaveD method can be used with noisy eigen values (without modification).
- The WaveD fine resolution level is chosen according to the degree of ill-posedness in a data-driven fashion and in agreement with the optimal theory.
- The WaveD method can be used with non-homogeneous operators such as in boxcar convolution.

From the statistical point of view, when comparing WaveD with non-wavelet methods, we note that WaveD enjoys the statistical properties specific to wavelet thresholding estimators:

- WaveD is a truly non-linear adaptive algorithm which has near optimal asymptotic properties over a wide range of function classes and for a variety of L^p-loss functions.

- WaveD is capable of representing functions with discontinuities or with non-homogeneous time and frequency behaviour.
- The translation invariant version of WaveD improves upon the numerical performance of ordinary WaveD by cycle spinning over all circulant shifts.
- WaveD is an non-iterative deconvolution technique.

1.2. What's new in the waved package?

Earlier versions of the WaveD method have been implemented through various small Matlab packages, corresponding to various existing WaveD transforms. For example one package uses the algorithm of Kolaczyk (1994) to compute the ordinary Meyer wavelet transform. Another uses the algorithm of Donoho and Raimondo (2004) to compute the translation invariant Meyer transform. These small WaveD packages are not self-contained and are intented for use with **WaveLab** (Buckheit, Donoho, Johnstone, and Sargle 1995).

This paper describes a unified setting where all the WaveD transforms are implemented in the software environment R (R Development Core Team 2007) via a contributed package named **waved** (Raimondo and Stewart 2006). The aims of the **waved** package are:

1. To make available, in one self-contained package all code necessary to compute the various WaveD transforms with optimal data-driven tuning for wavelet deconvolution.
2. To take full advantage of the object-oriented R environment: the (top) function, called `WaveD`, produces objects of class `wvd`. The `wvd` class of objects are R lists containing the various WaveD transforms as well as all the WaveD estimate characteristics such as threshold, fine resolution level, degree of Meyer wavelet and so on.
3. To introduce visual and statistical tools to assess the validity and the quality of a `WaveD` fit. Special features of the **waved** package include a `summary` and a `plot` function specifically designed for objects of class `wvd`.
4. To allow a user to reproduce illustrative figures and analyses from the literature.

Finally, we discuss how the **waved** package differs from existing R packages for wavelet analysis. Existing wavelet R packages include: the **wavethresh** package of Nason, Kovac, and Mächler (2006): a software to perform wavelet statistics and transforms; the **waveslim** package of Whitcher (2006): basic wavelet routines for one, two and three dimensional signal analysis; the **wavelets** package of Aldrich (2007): a package of functions for computing wavelet filters. These packages offer a wide range of compactly supported wavelet transforms, typically Daubechies wavelets, for direct data analysis. On the other hand the **waved** package is designed for indirect data analysis (such as noisy-convolution) and uses band-limited wavelets, typically Meyer wavelets.

1.3. Paper organisation

In section 2 we give a brief introduction to the WaveD transform based on the Fourier transform. Section 3 is concerned with setting-up the **waved** software and its `demo`. We also present the `WaveD` function in R and introduce objects of class `wvd`. In Section 4, we discuss some more advanced features of the `WaveD` function in R, this includes statistical applications, fine tuning of the parameters and `WaveD` fit assessment. Section 5 contains a list of **waved** main functions.

2. The WaveD transform

2.1. Fourier transforms

Convolution products are naturally represented in the Fourier domain. In the periodic setting, we can write the model (1) in terms of Fourier coefficients,

$$y_\ell = g_\ell f_\ell + \varepsilon \xi_\ell, \quad \ell \in \mathbb{Z}, \tag{3}$$

where, with $e_\ell(t) = e^{2\pi i \ell t}$ and $\langle f, g\rangle = \int_T f\bar{g}$, $f_\ell = \langle f, e_\ell\rangle, g_\ell = \langle g, e_\ell\rangle$ and $\xi_\ell = \langle \xi, e_\ell\rangle$ are i.i.d. standard (complex-valued) normal random variables. As for the model (2) we have

$$x_\ell = g_\ell + \epsilon z_\ell, \quad \ell \in \mathbb{Z}, \tag{4}$$

where z_ℓ are i.i.d. standard (complex-valued) Gaussian r.v.'s independent of ξ_ℓ, and noise level $0 < \epsilon < 1$. This model includes cases where the eigen-values (g_ℓ) are not fully known but are observed with noise as illustrated in Figure 8.

In this paper ψ denotes a Meyer wavelet and ϕ denotes the corresponding scaling function. Typically ψ is a band limited function whose Fourier transform $F(\psi) := \hat{\psi}$ is smooth (Mallat 1998, p.247). In practice, we use a polynomial function to define the so-called Meyer window (Mallat 1998, p.248). Throughout this paper we use $\hat{\psi}$ and $\hat{\phi}$ corresponding to a polynomial of degree 3, as in the software default setting. As usual $\psi_\kappa(x) = 2^{j/2}\psi(2^j x - k)$ where $\kappa = (j, k)$ denotes the translated-dilated version of ψ, and $\phi_\kappa(x)$ denotes the translated-dilated version of ϕ. In the periodic setting, the **waved** package utilises the periodised scaling function $\Phi_\kappa(x) := \sum_{\ell\in\mathbb{Z}} \phi_\kappa(x+\ell)$, and periodised wavelets $\Psi_\kappa(x) := \sum_{\ell\in\mathbb{Z}} \psi_\kappa(x+\ell)$, whose Fourier coefficients satisfy

$$\Psi_\ell^{j,0} = \langle \Psi_{j,0}, e_\ell\rangle = 2^{-j/2}\hat{\psi}(\ell/(2^j \times 2\pi)),$$

and

$$\Psi_\ell^\kappa = \langle \Psi_\kappa, e_\ell\rangle = \exp(2\pi i \ell k/2^j)\Psi_\ell^{j,0}.$$

2.2. The WaveD paradigm

The WaveD paradigm (Johnstone *et al.* 2004) stipulates that one can perform deconvolution and wavelet transforms simultaneously. To see this we write wavelet coefficients in terms of Fourier coefficients using Plancherel's formula. This is illustrated in the next diagram using the noise-free input function $h(t) = (f * g)(t)$. In this diagram (and in the sequel) $\rightarrow^F, \leftarrow^{F^{-1}}$ denotes the Fourier transform and its inverse. We define the Forward WaveD transform and its inverse as

$$\text{FWaveD}(h, g) = (\beta_\kappa)_\kappa, \quad \text{IWaveD}(\beta_\kappa) = \sum_\kappa \beta_\kappa \Psi_\kappa, \tag{5}$$

where $\beta_\kappa = \int f\Psi_\kappa$, $\kappa = (k, j)$, $j = -1, 0, 1, ...,$ $k = 0, ..., 2^j - 1$, $(\Psi_{-1,0} = \Phi)$.

Illustration of the WaveD paradigm in the time, Fourier and wavelet domain,

$$\begin{tikzcd}
\text{Time domain} & \text{Fourier domain} & & \text{Wavelet domain} \\
h(t) = (f * g)(t) \arrow[r, "F"] & h_\ell = f_\ell \times g_\ell & & \\
\Psi_\kappa(t) & (\Psi^\kappa_\ell) \arrow[rr] & & \int h \bar{\Psi}_\kappa \\
& & \begin{matrix} \sum_\ell (h_\ell) \bar{\Psi}^\kappa_\ell \\ h_\ell \div g_\ell \\ \sum_\ell f_\ell \bar{\Psi}^\kappa_\ell \end{matrix} & \textit{(elementwise division)} \\
& & & \int f \bar{\Psi}_\kappa = \beta_\kappa \\
\sum_\kappa \beta_\kappa \Psi_\kappa = f(t) & & & \arrow[lll]
\end{tikzcd}$$

2.3. Adaptive denoising via non-linear WaveD transform

In the case of noisy data (1), (2) we note that

$$\mathrm{FWaveD}(y, g) = \Big(\sum_\ell (\frac{y_\ell}{g_\ell}) \Psi^\kappa_\ell \Big)_\kappa := (\tilde{\beta}_\kappa)_\kappa,$$

provides an unbiased estimator of $(\beta_\kappa)_\kappa$. The **waved** software uses statistical techniques to perform wavelet regression and smoothing. The main idea is to remove small wavelet coefficients (noise) and keep large wavelet coefficients (signal). Optimal and data driven choices of WaveD tuning parameters are further discussed in section 4; here we shall only present the WaveD method in broad terms using a generic threshold function

$$\eta(\tilde{\beta}_\kappa) := \tilde{\beta}_\kappa \times \mathbb{I}(|\tilde{\beta}_\kappa| \geq \lambda), \tag{6}$$

where λ is a threshold parameter. The WaveD estimator (Johnstone *et al.* 2004) is

$$\mathrm{WaveD}(y, g) := \sum_\kappa \eta(\tilde{\beta}_\kappa) \Psi_\kappa(t) := \hat{f}(t), \tag{7}$$

which with a slight abuse of terminology we also call the WaveD transform. We summarise the main steps in the diagram below

$$\begin{tikzcd}
\text{Time domain} & \text{Fourier domain} & \text{Wavelet domain} \\
Y(t) \arrow[r, "F"] & y_\ell = f_\ell \, g_\ell + \varepsilon \xi_\ell \arrow[r] & \Big(\sum_\ell (\frac{y_\ell}{g_\ell}) \Psi^\kappa_\ell \Big)_\kappa \arrow[d, "\textit{thresholding}"] \\
\hat{f}(t) & & (\eta(\tilde{\beta}_\kappa))_\kappa \arrow[ll]
\end{tikzcd}$$

2.4. The translation invariant WaveD transform

Numerical (and computational) properties of the WaveD transform are improved using cycle spinning (Donoho and Raimondo 2004). For any $h > 0$, we denote $T_h f(x) = f(x + h)$ the shift operator and $\rightarrow^{\texttt{FWaveD}}$, $\leftarrow^{\texttt{IWaveD}}$ the Forward WaveD transform and its inverse (5). For an arbitrary time shift h we define one cycle-spin of the WaveD transform as

$$\begin{tikzcd}
& \text{Time domain} & \text{Wavelet domain} & \\
& Y(t) \arrow[d, "\textit{shift}"'] & & \\
& T_hY(t) \arrow[r, "\texttt{FWaveD}"] & (\beta^h_\kappa) \arrow[d, "\eta"] & \textit{thresholding} \\
& \tilde{f}_h(t) \arrow[d, "\textit{unshift}"'] & \eta(\beta^h_\kappa) \arrow[l, "\texttt{IWaveD}"'] & \\
& T_{-h}(\tilde{f}_h)(t) & &
\end{tikzcd}$$

Let $H_n = \{1/n, 2/n, ..., 1 - 1/n, 1\}$ be the set of all possible circulant shifts. The translation invariant WaveD transform is

$$\tilde{f}_{TI} = \text{Ave}_{h \in H_n} T_{-h}(\tilde{f}_h) = \frac{1}{|H_n|} \sum_{h \in H_n} T_{-h}(\tilde{f}_h)\,. \tag{8}$$

3. The WaveD transform in R

3.1. Software access

The **waved** software (Raimondo and Stewart 2006) is provided as an R package obtainable from the Comprehensive R Archive Network at `http://CRAN.R-project.org/`. Installation instructions are provided there also.

3.2. Getting help

Once the **waved** package has been installed detailed help pages for basic functions may be obtained within R using the `help()` function. For example `help(WaveD)` gives the help page of the main **waved** function. Note that **waved** refers to the R package whereas `WaveD` is the main function which performs wavelet deconvolution. See section 5 for a list of basic **waved** functions.

3.3. The waved `demo`

From now on we assume that the **waved** package has been attached. Typing `demo(waved)` provides a series of examples which illustrate various applications of the WaveD transform. To simulate data according to (1) and (A_ε) one needs to specify: (a) a target function f; (b) a convolution kernel g; (c) a sample size n; (d) a standard deviation σ. The simplest way to get started is to use the **waved** package `demo`. Just type `demo(waved)` and answer questions at the prompt. Alternatively, the function `waved.example()` can be used (recommended) to generate the data sets and figures in this paper by setting its two arguments to `TRUE` (default).

```
R> data <- waved.example(TRUE, TRUE)
```

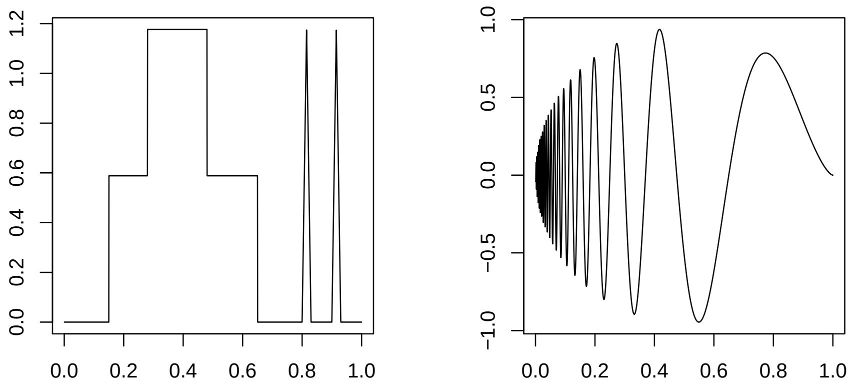

Figure 1: two signals $t \to f(t)$, $t_i = i/n, i = 1, ..., n = 2048$. Left: `lidar`; right: `doppler`.

```
-------------------------------------------------------
Initializing noisy-blurred signals model:
sample size n = 2048
noise  sd =  0.05
Convolution kernel g:
gamma-distribution with shape paremeter= 0.5
and scale parameter=  0.25
(effective) Degree of Ill-Posedness (DIP)=  0.5
The seed number has been set to  11
Blurred Signals to Noise Ratios:
Lidar   BSNR(dB) = 15.3
Doppler   BSNR(dB) = 13.8
-------------------------------------------------------
```

This creates a list `data` which contains the data used in this paper,

```
R> attach(data)
R> names(data)
```

```
 [1] "lidar.noisy"    "lidar.noisyT"   "doppler.noisyT" "lidar.blur"
 [5] "doppler.noisy"  "doppler.blur"   "t"              "n"
 [9] "g"              "lidar"          "doppler"        "seed"
[13] "sigma"          "g.noisy"        "g.noisyT"       "dip"
[17] "k.scale"
```

The various data sets are depicted in Figure 1 (target signals), Figure 2 (blurred signals) and Figure 3 (noisy blurred signals). The noise standard deviation default setting (as shown in

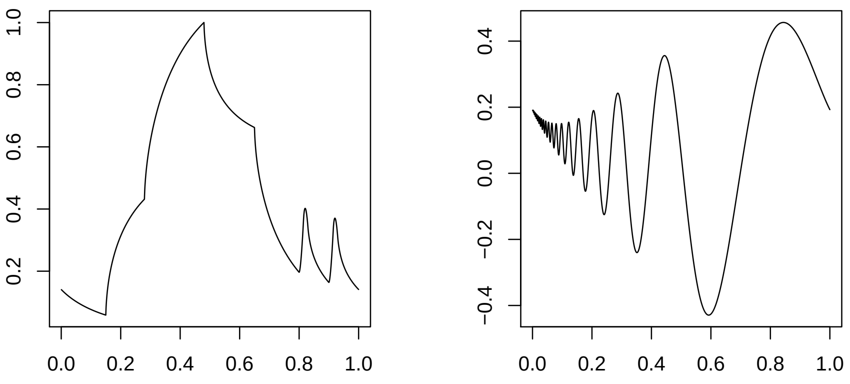

Figure 2: signals of Figure 1 after smooth blurring with DIP=ν=0.5, see (10).

Figure 3) is $\sigma = 0.05$ with sample size $n = 2048$ so that the blurred-signal-to-noise-ratios (BSNR), in dB, for signals of Figure 3 is approximately $15dB$ where

$$\mathrm{BSNR}_{dB} = 10 \log_{10}\Big(\frac{||f * g||^2}{\sigma^2}\Big). \tag{9}$$

In the default setting the convolution kernel g is defined using the density of a Gamma distribution with shape and scale parameters set to 0.5 and 0.25 respectively. In this setting, the eigen-values (g_ℓ) satisfy

$$|g_\ell| \sim |l|^{-\nu}, \tag{10}$$

with $\nu = 0.5$. The parameter ν which drives the decay of the eigen-values is often referred as the Degree of Ill-Posedness (DIP) of the convolution problem (Johnstone *et al.* 2004).

3.4. Setting up your examples

Once you become more familiar with the **waved** package you may want to generate your own data by modifying the default parameters of the `demo`. The function `waved.example()` can be used to generate simulated examples with different model parameters: sample size, noise level, Degree of Ill-Posedness, seed and so on. This is done by setting its first argument to `FALSE` and answering questions at the prompt. For example to set the sample size to $n = 4096$,

```
R> my.own.simulation <- waved.example(FALSE)

Please enter the sample size (must be a power of 2)

R> 4096
```

and so on to keep or change the other model parameters. To recover the data sets used in this paper just set the first argument of `waved.example()` to `TRUE`; the second argument refers to graphics display, the default setting is `TRUE`, hence

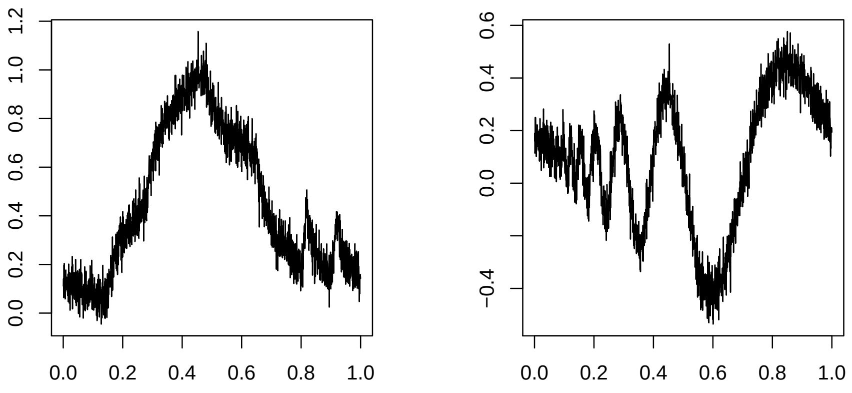

Figure 3: Blurred signals of Figure 2 *plus* noise with s.d $\sigma = 0.05$, BSNR $\approx 15\,dB$ see (9).

```
R> original.data.with.figures <- waved.example(TRUE, TRUE)
R> original.data.without.figures <- waved.example(TRUE, FALSE)
```

Alternatively, the figures can be produced by calling in **waved** `data` names

```
R> plot(t, lidar, type="l")
R> plot(t, lidar.blur, type="l")
R> plot(t, lidar.noisy, type="l")
```

3.5. The `WaveD` function and `wvd` objects

The function `WaveD` creates R objects of class `wvd`. The `wvd` class objects are lists which contain the various `WaveD` transforms as well as all the `WaveD` estimate characteristics such as threshold, resolution level, degree of the Meyer wavelet and so on. Statistical properties of objects of class `wvd` are discussed in Section 4. The `summary` and `plot` functions for objects of class `wvd` are discussed in Section 4.5. In its simplest version the `WaveD` function requires two input arguments: `y` a vector with elements $(y_1, ..., y_n)$ where $y_i = Y(t_i)$, $i = 1, \dots, n$, see (1), and `g` a vector $(g_1, ..., g_n)$ with elements $g_i = g_\epsilon(t_i)$, $i = 1, \dots, n$, see (2). Optional arguments to `WaveD` include: `F` the finest resolution level j used in the expansion (7) as well as the threshold value λ at (6). The parameter `F` may take any value within the range `L`$, ..., (\log_2(n) - 1)$ where `L` is a low resolution level (default `L = 3`). In our examples $n = 2048$ so that `F` may take any value within the range 3,...,10. For illustration purposes,

```
R> lidar.wvd <- WaveD(lidar.blur, g, F=6, thr=0)
R> multires(lidar.wvd$w, lo=3, hi=6)
R> lidar.noisy.wvd <- WaveD(lidar.noisy, g, F=6, thr=0)
R> multires(lidar.noisy.wvd$w, lo=3, hi=6)
```

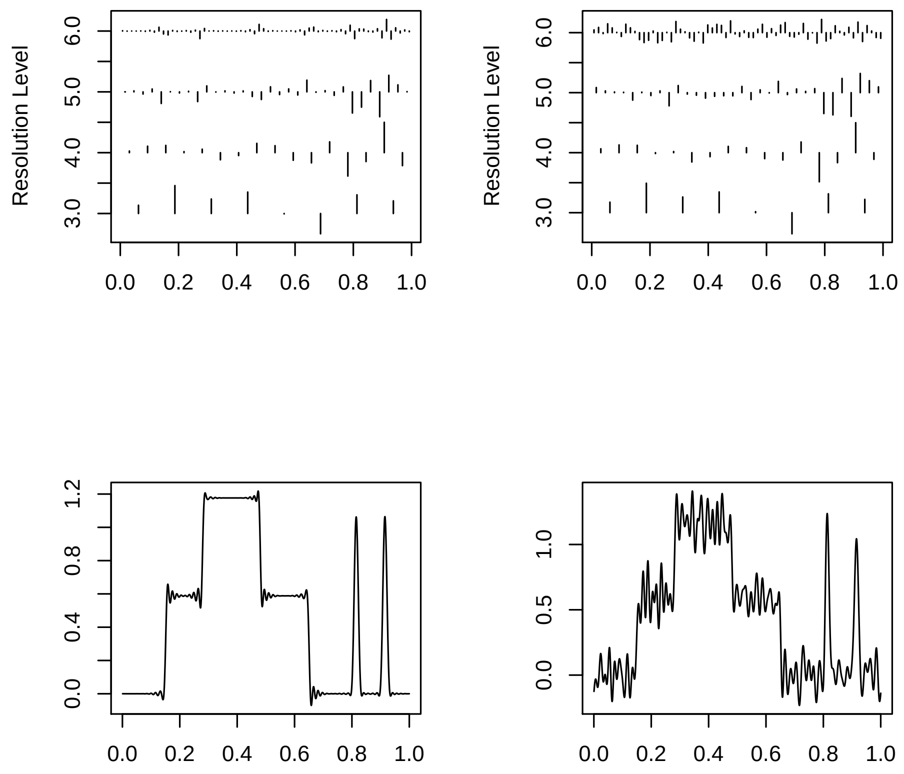

Figure 4: top plots depict the Forward WaveD transform of `lidar.blur` (left) and `lidar.noisy` (right). Bottom plots depict (corresponsding) inverse WaveD transforms (5).

these commands produce the WaveD transform of the blurred lidar data of Figure 2 as well as the WaveD transform of the noisy-blurred lidar data of Figure 3. In both cases using `F=6` as the finest resolution level and threshold `thr=0` (no thresholding). The corresponding plots can be seen in Figure 4.

The forward WaveD transform can be obtained by typing

```
R> lidar.w <- FWaveD(lidar.blur, g, F=6)
```

or `lidar.wvd$w` which returns the same output as `lidar.w` as defined above. The vector `lidar.w` is a vector of wavelet coefficients stored from the lowest resolution level to the highest resolution level. The function `dyad(j)` may be used to access wavelet coefficients at a given resolution level

```
R> lidar.wavelet.coef.at.level.5 <- lidar.w[dyad(5)]
```

A useful property of wavelet coefficients is that they are large (in absolute value) near discontinuities, see e.g. the top RHS plot of Figure 4. Another feature of wavelet coefficients is that they become more and more sensitive to noise as the resolution level increases. See e.g. the top LHS plots of Figure 4. In Figure 4 (top plots), the function `multires()` is used to depict wavelet coefficients according to time and frequency. More details about the `multires()` function and the data structure of the `WaveD` transforms are given in Section 5.

The inverse WaveD transform can be obtained by typing

```
R> inverse.waved <- IWaveD(lidar.w)
```

or `lidar.wvd$iw`. The vector `lidar.wvd$iw` returns the inverse WaveD transform (5) computed from `lidar.w` without any thresholding. Two illustrations of the inverse WaveD transform are depicted on the bottom plots of Figure 4 (with corresponding Forward WaveD transforms depicted on the top plots).

The ordinary WaveD transform is a combination of the `FwaveD` and `IWaveD` transforms together with some thresholding options (7). The ordinary WaveD transform is obtained from a `wvd` object by typing `lidar.wvd$ord` which, here, returns approximations to `lidar` as seen in Figure 4. If no thresholding is performed (`thr=0`) the ordinary WaveD transform returns the same output as the inverse WaveD transform. If a non-zero threshold is used the ordinary WaveD transform returns the inverse WaveD transform after thresholding. For noisy data it is desirable to improve WaveD approximations such as depicted on the RHS bottom plot of Figure 4 by using a non-zero threshold. This is detailed next.

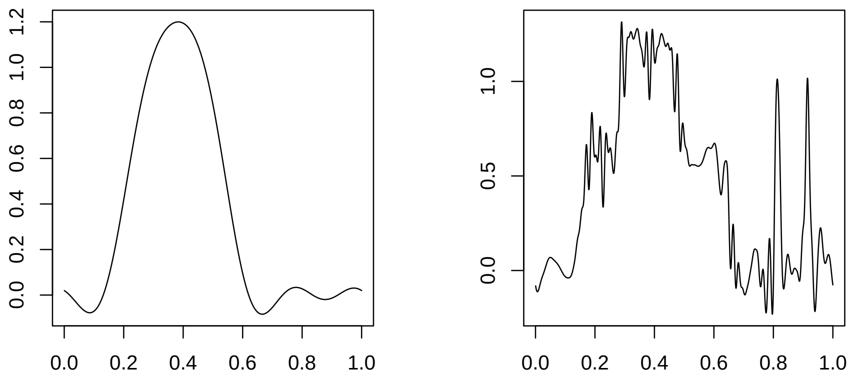

Figure 5: WaveD transform (7) of `lidar.noisy` with `F`=6, $\lambda = 0.2$ (left) and $\lambda = 0.02$ (right).

4. Statistical applications of the WaveD transform

In this section we discuss some more advanced features of the WaveD transform when dealing with noisy data. We use the simulated data of Figure 3 to illustrate how the `WaveD` function chooses the fine tuning parameters `F` and `thr` in a data-driven fashion in agreement with the optimal choices prescribed in the literature (Cavalier and Raimondo 2007).

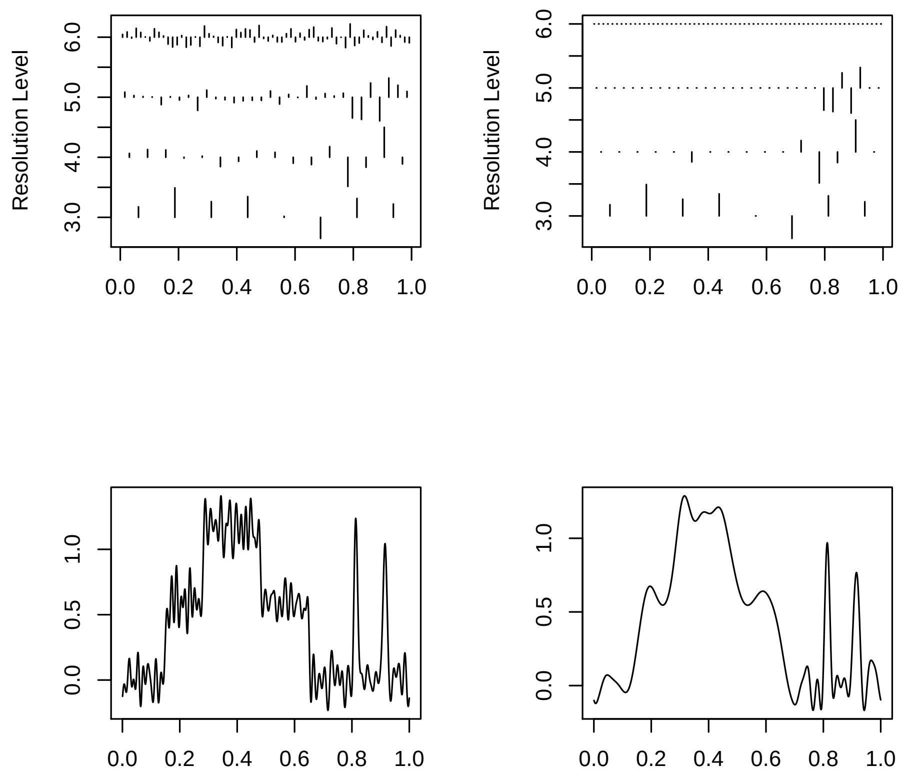

Figure 6: illustration of thresholding (11) using the `lidar.noisy` data. Top left: Forward WaveD transform (un-thresholded). Top right: Forward WaveD transform after maxiset thresholding. Bottom plots: corresponding WaveD estimates (7).

4.1. Choosing a threshold

The threshold value λ at (6) may be thought of as a smoothing parameter since it dictates the amount of smoothing in the estimate, large λ yields smoother estimates and vice-versa. A single threshold value λ may be entered directly in the `WaveD` function

```
R> plot(t, WaveD(lidar.noisy, g, F=6, thr=0.2)$ord, type="l")
R> plot(t, WaveD(lidar.noisy, g, F=6, thr=0.02)$ord, type="l")
```

with corresponding plots depicted in Figure 5. Alternatively a set of level dependent thresholds may be entered as a vector,

```
R> my.thr <- c(0.01, 0.02, 0.03, 0.04)
R> lidar.my.thr.wvd <- WaveD(lidar.noisy, g, L=3, F=6, thr=my.thr)
```

will use threshold $\lambda = 0.01$ at level $j = 3$, $\lambda = 0.02$ at resolution level $j = 4$ and so on.

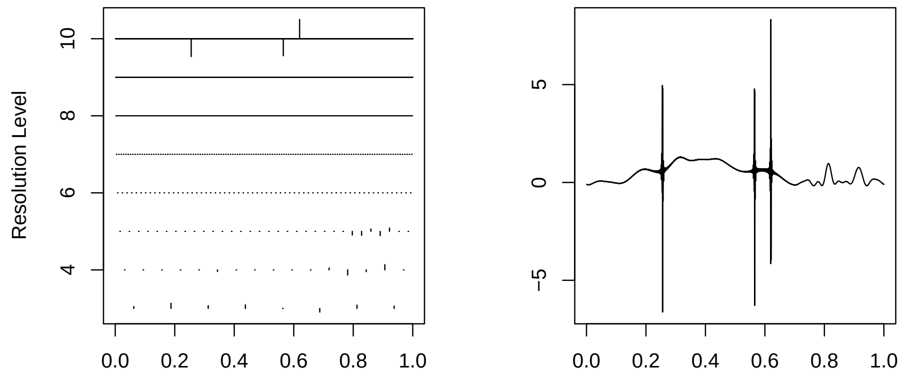

Figure 7: the maxiset threshold (11) do not always prevent noise in high resolution level. Left: the Forward WaveD transform of `lidar.noisy` after maxiset threshodling when `F=10`. Right: corresponding WaveD estimate (7).

The maxiset threshold. If no threshold parameter is specified the `WaveD` function will use the so-called "maxiset threshold" (11). This level-dependent threshold is derived from the maxiset theory (Johnstone *et al.* 2004). For example,

```
R> lidar.maxi.wvd <- WaveD(lidar.noisy, g)
```

will use the follwing threshold values

```
R> round(maxithresh(lidar.noisy, g, L=3, F=6), 4)

[1]  0.0134 0.0198 0.0298 0.0459
```

corresponding to a vector `thr` with entries $(\lambda_3, \lambda_4, ..., \lambda_6)$ of level dependent thresholds (11). The effect of the maxiset threshold is illustrated in Figure 6. As seen in the RHS plots of Figure 6, the WaveD estimate with the maxiset threshold automatically select significant coefficients to be kept for the reconstruction. This process removes noise (small coefficients) and smoothes the estimate. The un-thresholded and the thresholded Forward WaveD transforms may be obtained from a `wvd` object as follows,

```
R> unthresholded.w <- lidar.maxi.wvd$w
R> multires(unthresholded.w, lo=3, hi=6)
R> thresholded.w <- lidar.maxi.wvd$w.thr
R> multires(thresholded.w, lo=3, hi=6)
```

which produce the plots of Figure 6. The maxiset threshold is computed as follows

$$\lambda_j = \hat{\sigma}\gamma\,\sigma_j\,c_n. \tag{11}$$

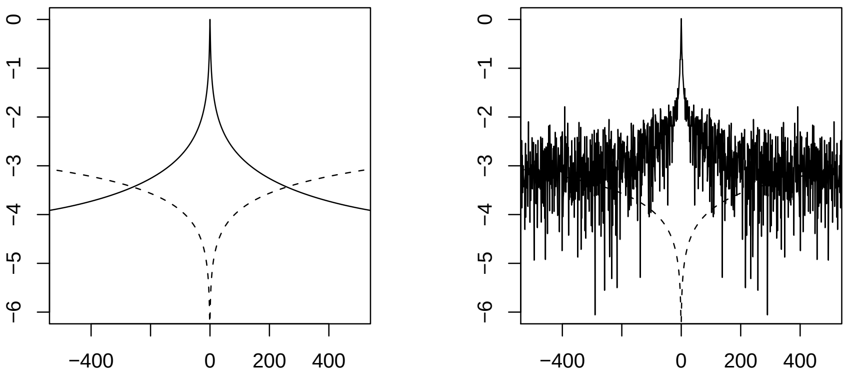

Figure 8: illustration of the fine level selection in the Fourier domain (13), a log-scale is used on the vertical-axis. In both plots the dashed curve represent the noise level $\ell \to \log|\ell^{1/2}\varepsilon(\log(1/\varepsilon^2))|$. Left, solid curve: $\ell \longrightarrow \log|g_\ell|$, where g_ℓ are noise-free eigen-values ($\epsilon = 0$). Right: noisy-eigen-values (solid) $\ell \longrightarrow \log|x_\ell|$ where $x_\ell = g_\ell + \epsilon\xi_\ell$ with $\epsilon = \varepsilon = 0.05/\sqrt{2048}$.

- $\hat{\sigma}$: estimate of the noise standard deviation, σ. If $y_{J,k} = \langle Y, \Psi_{J,k}\rangle$, denote the finest scale wavelet coefficients of the observed data, then $\hat{\sigma} = m.a.d.\{y_{J,k}\}/.6745$, where $m.a.d.$ is median absolute deviation. Type `scale(lidar.noisy)` to get $\hat{\sigma}$ for the `lidar.noisy` data.

- γ: constant which depends on the tail of the noise distribution. For Gaussian noise, the range $\sqrt{2} \le \gamma \le \sqrt{6}$ gives good results in practice. The default setting for `WaveD` is the conservative choice $\gamma = \sqrt{6}$.

- σ_j: level-dependent scaling factor which depends on the convolution kernel.

$$\sigma_j := \tau_j(x_\ell) = \Big(|C_j|^{-1} \sum_{\ell \in C_j} |x_\ell|^{-2}\Big)^{1/2},$$

  where $C_j = \{l : \Psi_\ell^\kappa \ne 0\} \subset (2\pi/3) \cdot [-2^{j+2}, -2^j] \bigcup [2^j, 2^{j+2}]$.

- c_n: sample size-dependent scaling factor reminiscent of the Universal threshold:

$$c_n = \Big(\frac{\log n}{n}\Big)^{1/2}.$$

4.2. Choosing the finest resolution level

The fine resolution level `F` is related to the highest (Fourier) frequency M allowed in the WaveD estimator $2^{\mathtt{F}} \approx \mathtt{M}$. The tuning parameter `F` stipulates the range of resolution levels where the approximations (7) or (8) are used:

$$\Lambda_n = \{(j,k),\ \mathtt{L} \le j \le \mathtt{F}, 0 \le k \le 2^j\}.$$

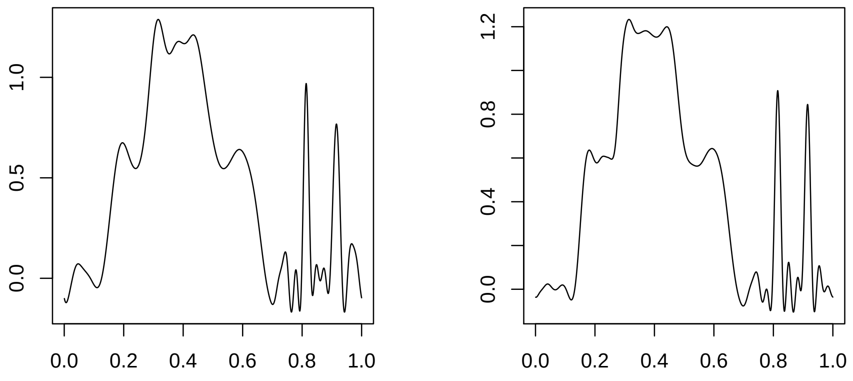

Figure 9: left, ordinary WaveD transform of `lidar.noisy`. Right, translation-invariant WaveD transform of `lidar.noisy`.

Here `L` is a low resolution parameter (default is `L = 3`). A numerical value for `F` within the range $\mathtt{L} \leq \mathtt{F} \leq \log_2(n) - 1$ may be entered directly in the `WaveD` function as in the examples of Section 3.5 where we used `F = 6`. If $n = 2048$ the maximum value allowed is `F`=10,

```
R> lidar.Fmax.wvd <- WaveD(lidar.noisy, g, F=10)
R> multires(lidar.Fmax.wvd$w.thr)
R> plot(t, lidar.Fmax.wvd$ord, type="l")
```

Unlike direct estimation problems (Donoho, Johnstone, Kerkyacharian, and Picard 1995) where it is customary to keep all resolution levels setting $\mathtt{F} = \log_2(n) - 1$, the asymptotic theory for deconvolution (Johnstone *et al.* 2004) shows that one should stop at a fine resolution level $\mathtt{F} = j_1$ where j_1 depends on the degree of ill-posedness of the convolution kernel (10)

$$2^{j_1} = O\Big((\frac{n}{\log n})^{\frac{1}{1+2\nu}} \Big). \tag{12}$$

The last condition shows that the faster the eigen values go to zero the sooner the wavelet expansion should stop. In pratical terms this means that the maxiset threshold will prevent noise in the estimate up until a high resolution level j_1 which depends on the degree of difficulty of the convolution as well as the noise level. It is important to note that, even after maxiset thresholding, the WaveD estimate based on all resolution levels may, sometimes, contain high noise perturbations. This is illustrated on Figure 7 where there are large noise residuals in the WaveD estimate due to large (unthresholded) coefficients at resolution level `F`=10.

Data driven fine level selection. To prevent noise perturbation at high resolution level, `WaveD` is fitted with a function `find.j1` which implement the data-driven method of Cavalier and Raimondo (2007) to find the optimal fine resolution level j_1 for noisy deconvolution based on the maxisets threshold. The idea is to keep all (Fourier) frequencies until (the moduli of)

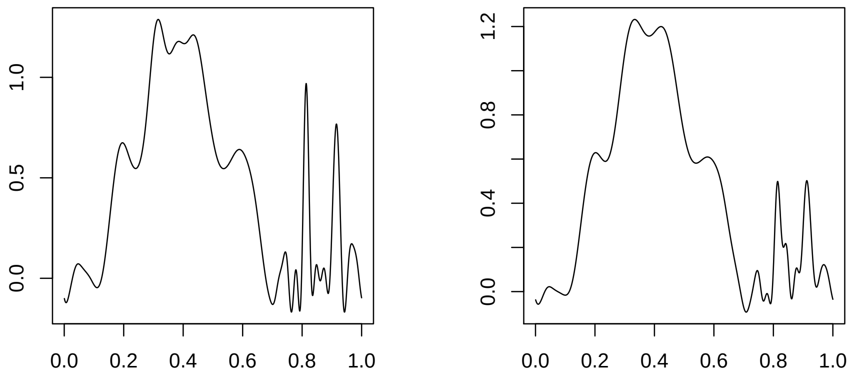

Figure 10: ordinary WaveD transform of `lidar.noisy` with `HARD` thresholding (left) and `SOFT` thresholding (right).

the eigen values fall below an appropriate noise level. This is illustrated on Figure 8. Let

$$M = \min\left\{\ell, \ell \geq 0 : |x_\ell| \leq \ell^{1/2}\,\varepsilon\,(\log 1/\varepsilon^2)\right\}, \tag{13}$$

denote the maximum Fourier frequency allowed in the WaveD formula (5). Then we define the maximum wavelet resolution level as

$$\hat{j}_1 = \lfloor \log_2(M) \rfloor - 1, \tag{14}$$

where $\lfloor x \rfloor$ is the largest integer below x. As seen on Figure 8 this process works for both noise-free eigen-values ($\epsilon = 0$) and noisy eigen-values. For example,

```
R> print(find.j1(g, scale(lidar.noisy)))

[1] 198    6
```

returns the optimal Fourier frequency `M` and fine resolution level `F`. The `plotspec90` function produces plots of the fine level selection process (13)

```
R> plotspec(g, scale(lidar.noisy))
R> plotspec(g.noisy, scale(lidar.noisy))
```

as seen in Figure 8.

4.3. Improving the fit using the translation invariant WaveD transform

While thresholding wavelet coefficients reduces the noise and smoothes the WaveD estimate it also introduces Gibbs phenomenon near discontinuities see e.g. RHS bottom plots of Figure 6.

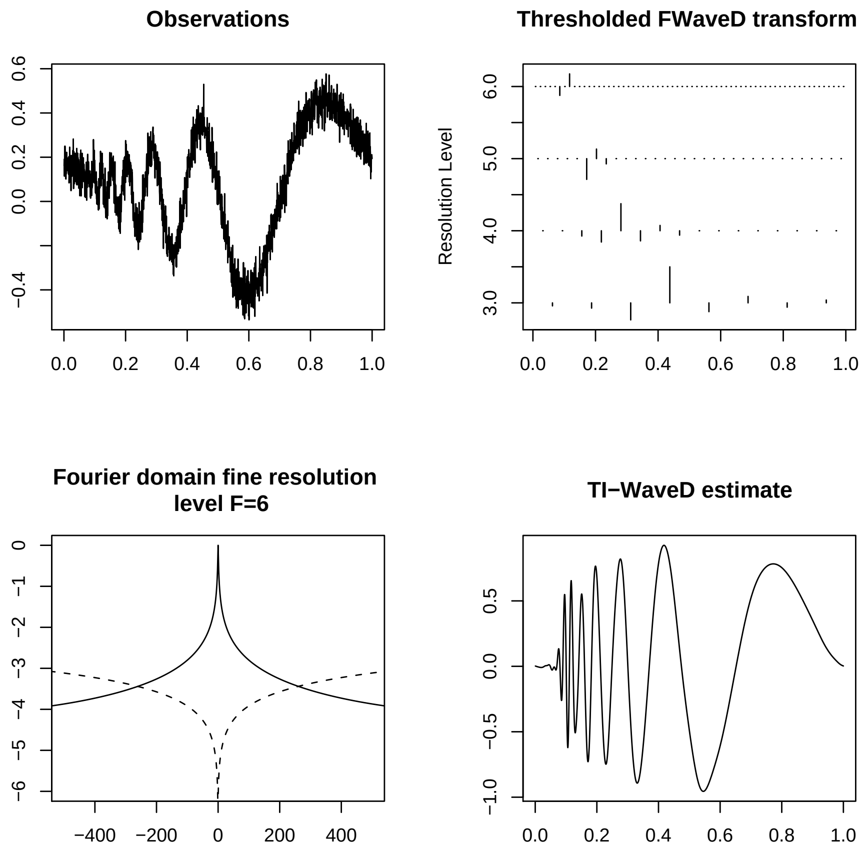

Figure 11: a typical plot of an object of class `wvd`, `plot(doppler.wvd)`.

Such Gibbs effects can be reduced by cycle spining (Donoho and Raimondo 2004). Both the ordinary (7) and the translation invariant (8) WaveD transform can be obtained from a `wvd` object,

```
R> plot(t, lidar.maxi.wvd$ord, type="l")
R> plot(t, lidar.maxi.wvd$waved, type="l")
```

with corresponding outputs depicted in Figure 10. In any case where some thresholding is performed we recommend using the translation invariant WaveD transform as it reduces visual artifacts.

The MC-option. The algorithm which implements the translation-invariant WaveD transform takes full advantage of the Fast Fourier Transform and requires only $O(n(\log n)^2)$ steps. This is faster than the algorithm which implements the ordinary WaveD transform. For convenience we provide an `MC` (Monte Carlo) option in the `WaveD` function. The default setting is `MC=FALSE` so that a `wvd` object like `WaveD(lidar.noisy, g)` contains both the ordinary and the translation invariant WaveD transforms. For faster computations in simulations and

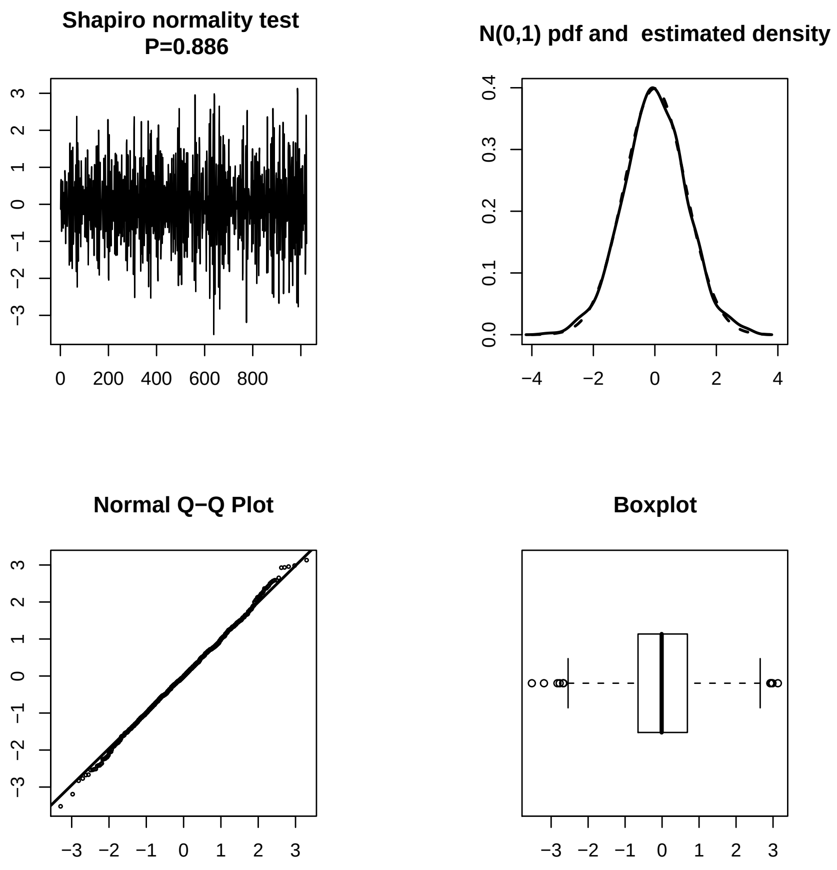

Figure 12: `plot(doppler.wvd)`. These residual plots suggest a good WaveD fit.

Monte-Carlo approximations, it is possible to set `MC=TRUE`, in this case the `WaveD` function will only return the translation invariant WaveD estimate,

```
R> lidar.ti.fast.waved <- WaveD(lidar.noisy, g, MC=TRUE)
```

4.4. Thresholding policy

There are many ways to threshold wavelet coefficients and different strategies may be used (Donoho *et al.* 1995). The two main thresholding policies studied in the literature are the `HARD` thresholding policy as in (6) or the `SOFT` thresholding policy,

$$\eta_S(\tilde{\beta}_\kappa) := \text{sign}(\tilde{\beta}_\kappa)(|\tilde{\beta}_\kappa| - \lambda) \times \mathbb{I}(|\tilde{\beta}_\kappa| \geq \lambda), \tag{15}$$

where λ is a threshold parameter. The statistical theory for WaveD estimation (Johnstone *et al.* 2004) is established for the `HARD` threshold policy (6) which is the default setting in `WaveD`. However, for data analysis purposes and experimental study we provide a `SOFT` thresholding option (15) in the `WaveD` function,

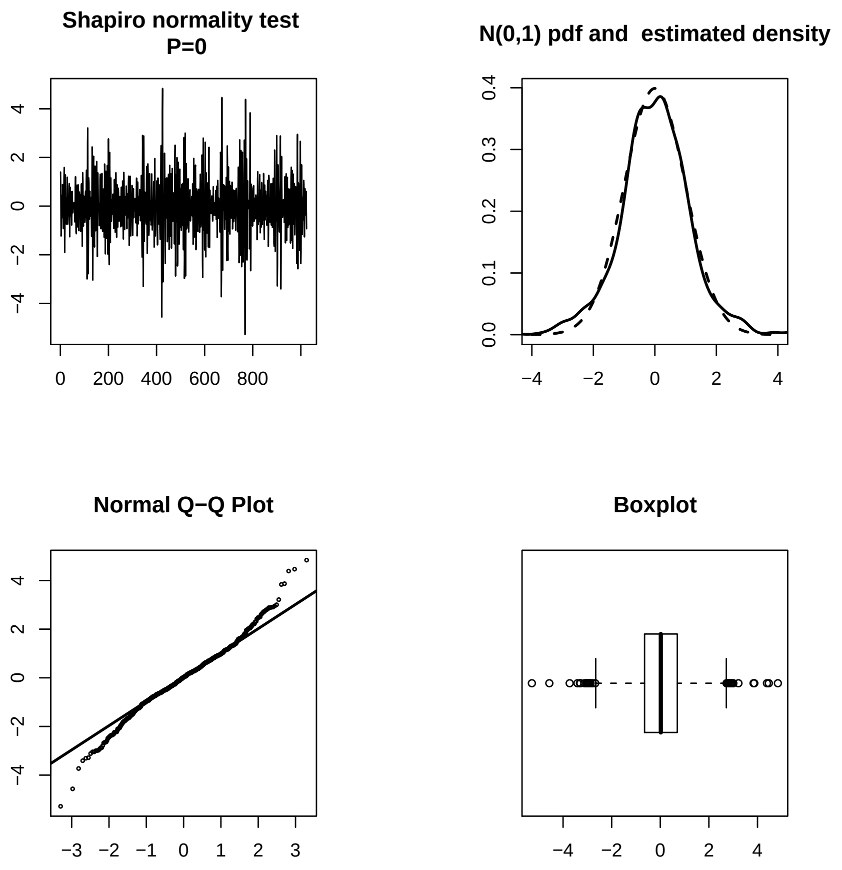

Figure 13: `plot(lidarT.wvd)`. These residual plots suggest a poor WaveD fit.

```
R> plot(t, WaveD(lidar.noisy, g, SOFT=FALSE)$ord, type="l")
R> plot(t, WaveD(lidar.noisy, g, SOFT=TRUE)$ord, type="l")
```

As seen in Figure 10, `SOFT` thresholding tends to further smooth the WaveD estimate but the general appearance does not appear as sharp as the translation invariant WaveD estimate of Figure 9.

4.5. The `summary` and `plot` functions for `wvd` objects

For convenience we provide a `summary` and a `plot` function specifically for objects of class `wvd`. These functions can be used to assess the quality of WaveD fits. We illustrate this using the `doppler.noisy` data set (`doppler` in Gaussian noise, see Figure 11) and the `lidar.noisyT` data set (`lidar` in Student-t_2 noise, see Figure 14).

```
R> doppler.wvd <- WaveD(doppler.noisy, g)
R> lidarT.wvd <- WaveD(lidar.noisyT, g)
R> plot(doppler.wvd)
```

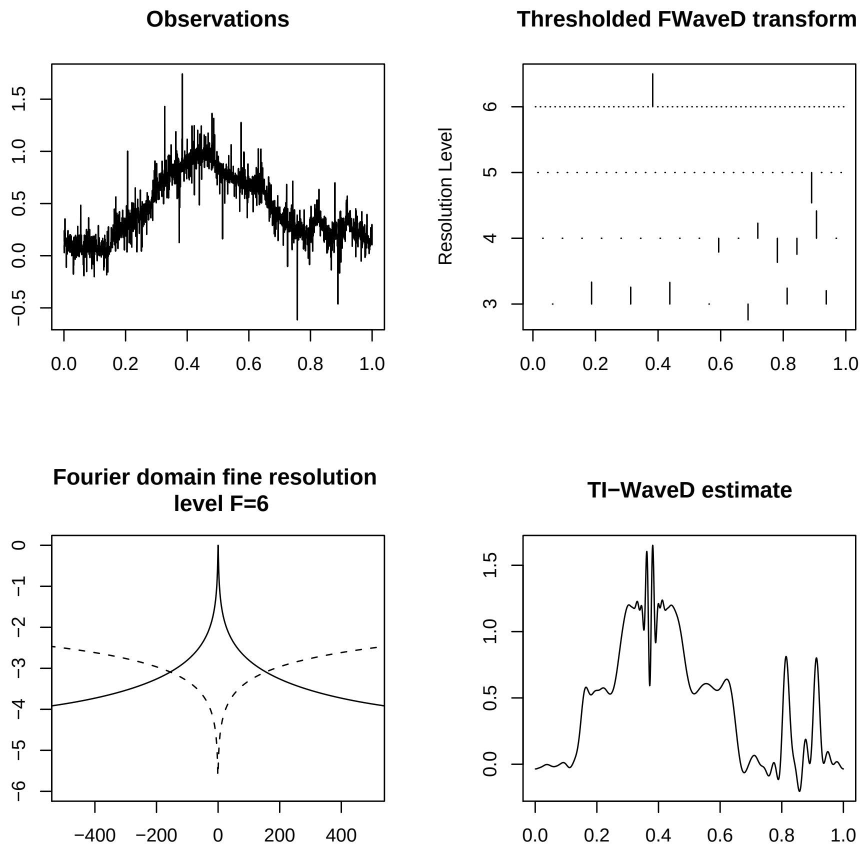

Figure 14: `lidar` WaveD fit in non-Gaussian noise (default setting). The data set `lidar.noisyT` (depicted on the top LHS) is the blurred-lidar of Figure 2 *plus* a Student-t_2 noise scaled so that the BSNR (9) is approximately 15 dB. This plot was produced by `plot(lidarT.wvd)`.

The `plot` function for `wvd` objects is illustrated in Figures 10,12 and 14.

```
R> summary(doppler.wvd)

Call:
WaveD(yobs = doppler.noisy, g = g)
Degree of Meyer wavelet = 3  , Coarse resolution level= 3
Sample size = 2048  , Maximum resolution level= 10 .
WaveD optimal Fourier freq= 196 ; WaveD optimal fine resolution level j1= 6
The choice of the threshold is: Maxiset threshold
Thresholding policy= Hard .   Threshold constant gamma= 2.449

           Max|w| Threshold        % of thresholding
level 3     0.301      0.009             0.125
level 3     0.222      0.013             0.000
```

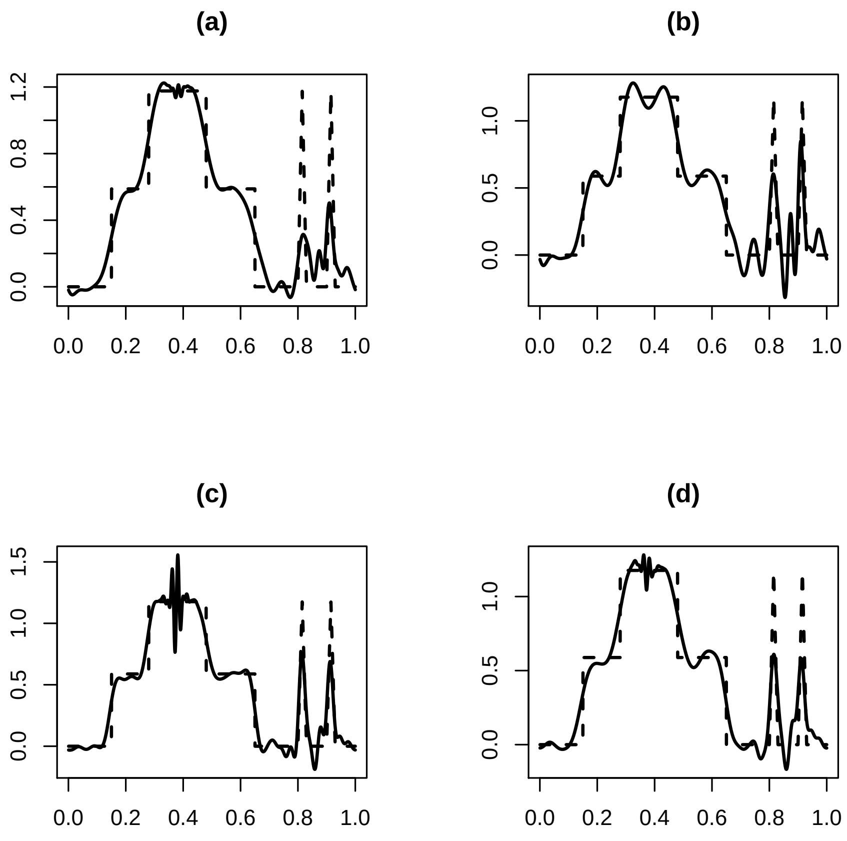

Figure 15: Various `lidar` WaveD fits in the Student-t_2 noise scenario.

```
level 4      0.167      0.020                  0.625
level 5      0.128      0.030                  0.906
level 6      0.078      0.046                  0.969
```

In addition to providing the tuning parameters `F` $= j_1$, `thr` = threshold, `M` = maximum Fourier frequency, γ = maxiset threshold noise constant as well as thresholding policy, the `summary` function gives some additional statistics such as the percentage of thresholding at a given resolution level as well as the maximum (in absolute value) of the wavelet coefficients at a given resolution level. It also gives the result of a test for normality based on the estimated noise in the data. This can be used to assess the WaveD fit as discussed next.

Estimating noise contribution. In statistical application of wavelet methods it is customary to estimate noise feature such as variance or tail index using the wavelet coefficients of the raw data at the largest resolution level (Raimondo and Tajvidi 2004). Here we shall call the vector of wavelet coefficients at the largest resolution level: `noise.proxy`. This vector may be obtained from a `wvd` object: `noise.proxy <- lidar.maxi.wvd$noise`. The `summary` and `plot` functions use the `noise.proxy` vector to perform some elementary data analysis,

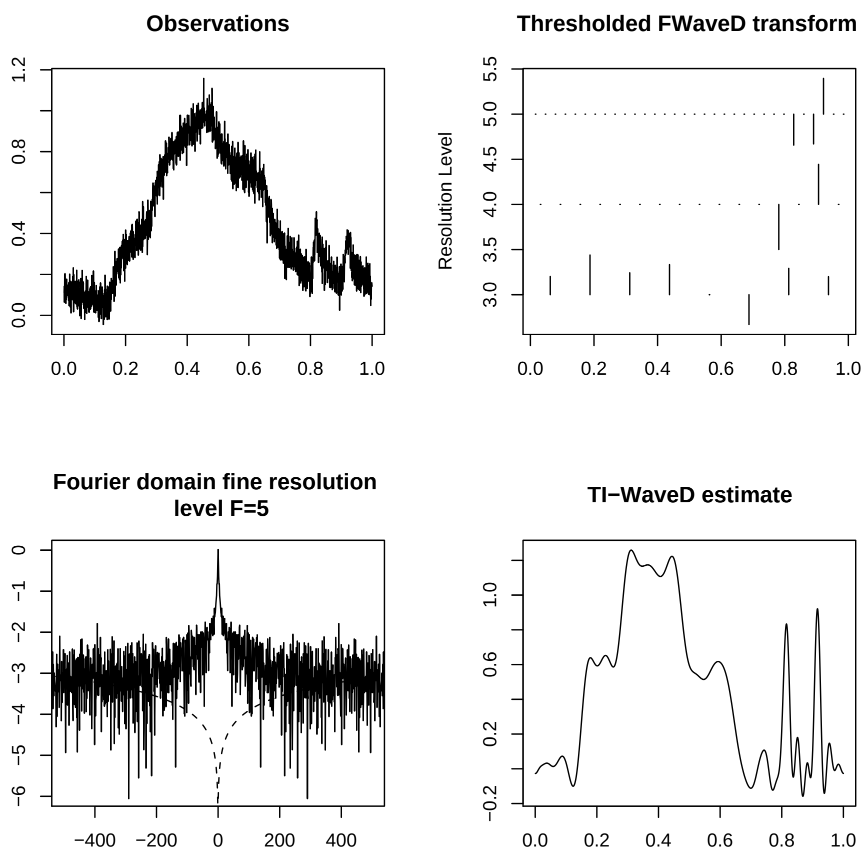

Figure 16: `lidar` WaveD fit when the eigen-values are noisy.

compare Figure 12 with Figure 13.

WaveD-fit assessment. The asymptotic theory and fine tuning of the WaveD parameters (Cavalier and Raimondo 2007) is based on the Gaussian white noise model (1) in which the error terms follow a normal distribution. A close inspection to the proof shows that the constant γ used in the maxiset threshold depends on the tail of the noise. For Gaussian noise the value $\gamma = \sqrt{6}$ gives good results in simulation. However, in other scenarios a larger value may be needed as this would be the case for heavy tailed noise. To assess the appropriateness of the WaveD fit and of the maxiset threshold, the `summary` function gives the result of a (Shapiro) test for normality based on the estimated noise in the data. For example,

```
R> plot(lidarT.wvd)
R> summary(lidarT.wvd)
```

```
...Estimated standard deviation=  0.094
Shapiro test for normality, P= 1.490849e-12
```

we see that for the Student-t_2 noise scenario, the WaveD fit residuals fail the normality-test with a Shapiro-test P-value close to zero. The corresponding WaveD estimate exhibits a large noise residual even after thresholding as seen on the bottom-RHS plot of Figure 14. This combined with the residual plots of Figure 11 suggest a poor WaveD fit.

Improving WaveD-fit in non-Gaussian noise. We suggest some heuristic approaches to improve WaveD fit in non-Gaussian scenarios,

```
R> plot(WaveD(lidar.noisyT, g, SOFT=TRUE)$ord, type="l")
R> plot(WaveD(lidar.noisyT, g, SOFT=TRUE, eta=sqrt(8))$ord, type="l")
R> plot(WaveD(lidar.noisyT, g, SOFT=FALSE, eta=sqrt(8))$waved, type="l")
R> plot(WaveD(lidar.noisyT, g, SOFT=FALSE, eta=sqrt(12))$waved, type="l")
```

(a) Using a soft threshold tends to reduce noise contributions and is more robust against non-normal noise; (b) using an ordinary WaveD estimate with a slightly larger γ tends to reduce noise contributions and is more robust against non-normal noise (in the summary function check `max|w|` against the threshold and increase γ accordingly); (c) using a TI-WaveD estimate with a slightly larger γ tends to reduce noise contributions but may not remove residuals contributions as effectively as in Ordinary WaveD; (d) using a TI-WaveD estimate with a bigger γ tends to reduce noise contributions and Gibbs phenomena. These four approaches are illustrated on Figure 15 using the `lidar.noisyT` data. On this occasion the TI-WaveD estimate with $\gamma = 2\sqrt{3}$ yields a better estimate.

4.6. WaveD estimation with noisy eigen-values

We finish this section by illustrating further adaptive properties of WaveD estimates. Depicted in Figure 16 is a WaveD `lidar` fit constructed from the noisy-blurred data of Figure 3 and the noisy eigen-values in the RHS plot of Figure 8.

```
R> lidar.NEV.wvd <- WaveD(lidar.noisy, g.noisy)
R> plot(lidar.NEV.wvd)
```

By comparing Figure 16 with Figure 9 we see that the quality of the WaveD approximation is not affected much if one uses noisy eigen values instead of the true eigen values. This is consistent with the asymptotic theory and numerical results of Cavalier and Raimondo (2007).

5. R commands

5.1. The `WaveD` command

The command `WaveD(y,g)` performs wavelet deconvolution using the data `y` and the convolution kernel `g`. If `g` is not specified `WaveD(y)` performs a (direct) wavelet transform.

- Required arguments
  - `y`: a vector with elements $(y_1, ..., y_n)$ where $y_i = Y(t_i)$, $i = 1, \ldots, n$, as in (1)
  - `g`: a vector $(g_1, ..., g_n)$ with elements $g_i = g_\epsilon(t_i)$, $i = 1, \ldots, n$, as in (2).
- Optional arguments

  - `L`: lowest resolution level (default=3).
  - `F`: finest resolution level (default=data driven choice (14)).
  - `deg`: deg of the Meyer Wavelet deg=1,2, or 3 (default=3).
  - `eta`: threshold parameter (default=$\sqrt{6}$).
  - `MC`: if Monte Carlo (`MC=TRUE`) WaveD returns only the TI-WaveD (default=`FALSE`). Note if `MC=TRUE` the WaveD output is a simple vector not a list.
  - `SOFT`: if `SOFT=TRUE` WaveD uses the soft-thresholding policy else hard (default=`FALSE`).
  - `thr`: threshold length=1 or length=`F-L`+2 (default is maxiset threshold (11)).

- Value: in the case that `MC=TRUE`, `WaveD` returns a vector consisting of the translation invariant WaveD estimate (8). In the case that `MC=FALSE` (the default), `WaveD` returns an object of class `wvd`, list with following components

  - `j1`: estimate of optimal resolution level (14).
  - `F`: fine resolution level used (may be different than `j1`).
  - `M`: estimate of optimal Fourier frequency (13).
  - `thr`: threshold (6).
  - `w`: Forward WaveD Transform (before thresholding).
  - `FWaveD`: same as `w`.
  - `w.thr`: Forward WaveD Transform (after thresholding).
  - `iw`: Inverse WaveD Transform (based on `w`).
  - `ordinary`: ordinary WaveD transform (7).
  - `waved`: translation invariant WaveD transform (8).
  - `percent`: percent of thresholding per resolution level.
  - `noise`: noise proxy, wavelet coefficients at the largest resolution level.
  - `p`: P-value of the Shapiro normality test based on `noise`.
  - `residuals`: wavelet coefficients that have been removed before fine level `F`.

5.2. Other useful commands

We give a list of other **waved** commands which can be used independently of the `WaveD()` function. In the examples below it is assumed that `y` is a vector with elements $(y_1, ..., y_n)$ where $y_i = Y(t_i)$, $i = 1, \ldots, n$, as in (1) and that `g`: a vector $(g_1, ..., g_n)$ with elements $g_i = g_\epsilon(t_i)$, $i = 1, \ldots, n$, as in (2).

- `FWaveD(y, g)`: the command `lidar.w=FWaveD(y, g)` returns a vector of wavelet coefficients as in `WaveD(y, g)$w`. This vector has length n, the last $n/2$ entries are wavelet coefficients at resolution level $(J-1)$ where $J = \log_2(n)$; the $n/4$ entries before that are wavelet coefficients at resolution level $(J-2)$, and so on until level L. In addition, the first 2^L entries are scaling coefficients at coarse resolution level $C = L$. See the `dyad()` function below for how to access wavelet coefficients at a given resolution level.

- `dyad(j)` returns integers $2^j+1, ..., 2^{j+1}$, hence the command `WaveD(y,g)$w[dyad(7)]` returns the wavelet coefficients at resolution level 7.
- `multires(WaveD(y, g)$w, lo=3, hi=7)` depicts wavelet coefficients according to time and resolution level 3,4,..7. In a fashion similar to the top plots of Figure 4.
- `maxithresh(y, g, L=3, F=7)` returns the maxiset threshold (11).
- `scale(y)` returns an estimate of the noise standard deviation.
- `find.j1(g, scale(y))` returns the optimal Fourier frequency (13) and optimal resolution level (14).
- `IWaveD(WaveD(y,g)$w)` returns the inverse WaveD transform. The `IWaveD` function can be used to construct/plot wavelets $\Psi_{j,k}$. First create a vector with n entries all equal to zero and then set its entry with index $ind = 2^j + k + 1$, as given by the function `dyadjk(j,k)`, to one. Then use the `IWaveD()` function. For example,

  ```
  R> wL <- rep(0,2048)
  R> wR <- rep(0,2048)
  R> wL[dyadjk(4,3)] <- 1
  R> wR[dyadjk(6,40)] <- 1
  R> plot(t, IWaveD(wL,3), type="l")
  R> plot(t, IWaveD(wR,3), type="l")
  ```

  returns plots of the $\Psi_{4,3}$ and $\Psi_{6,40}$ Meyer wavelets.

Acknowledgments

Bothauthors are grateful for the comments of an editor, an associate editor and a referee which have improved the original version of the paper. Both authors would like to aknowledge the contributions on the **waved** project from Laurent Cavalier, David Donoho, Iain Johnstone, Gérard Kerkyacharian and Dominique Picard.

References

Abramovich F, Silverman BW (1998). "Wavelet Decomposition Approaches to Statistical Inverse Problems." *Biometrika*, **85**(1), 115–129.

Aldrich E (2007). **wavelets***: A Package of Funtions for Computing Wavelet Filters, Wavelet Transforms and Multiresolution Analyses.* R package version 0.2-2, URL `http://www.atmos.washington.edu/~ealdrich/wavelets/`.

Bertero M, Boccacci P (1998). *Introduction to Inverse Problems in Imaging.* Institute of Physics, Bristol and Philadelphia.

Buckheit JB, Donoho D, Johnstone IM, Sargle JD (1995). **WaveLab** *Reference Manual.* Stanford University, Stanford, USA. URL `http://www-stat.stanford.edu/~wavelab/`.

Cavalier L, Koo JY (2002). "Poisson Intensity Estimation for Tomographic Data Using a Wavelet Shrinkage Approach." *IEEE Transactions on Information Theory*, **48**, 2794–2802.

Cavalier L, Raimondo M (2006). "On Choosing Wavelet Resolution in Image Deblurring." In E Banissi, M Sarfraz, M Hunag, Q Wu (eds.), "Computer Graphics, Imaging and Visualisation 2006, Proceedings," pp. 177–181. IEEE Computer Society. ISBN 0-7695-2606-3.

Cavalier L, Raimondo M (2007). "Wavelet Deconvolution With Noisy Eigen-Values." *IEEE Transactions on Signal Processing*, **55**(6), 2414–2424.

Donoho D (1995). "Nonlinear Solution of Linear Inverse Problems by Wavelet-Vaguelette Decomposition." *Applied Computational and Harmonic Analysis*, **2**, 101–126.

Donoho D, Johnstone IM, Kerkyacharian G, Picard D (1995). "Wavelet Shrinkage: Asymptopia?" *Journal of the Royal Statistical Society B*, **57**, 301–369. With discussion.

Donoho D, Raimondo M (2004). "Translation Invariant Deconvolution in a Periodic Setting." *The International Journal of Wavelets, Multiresolution and Information Processing*, **14**(1), 415–423.

Donoho D, Raimondo M (2005). "A Fast Wavelet Algorithm for Image Deblurring." *The Australian & New Zealand Industrial and Applied Mathematics Journal*, **46**, C29–C46. URL http://anziamj.austms.org.au/V46/CTAC2004/Dono.

Fan J, Koo JK (2002). "Wavelet Deconvolution." *IEEE Transactions on Information Theory*, **48**(3), 734–747.

Johnstone IM (1999). "Wavelet Shrinkage for Correlated Data and Inverse Problems: Adaptivity Results." *Statistica Sinica*, **9**(1), 51–83.

Johnstone IM, Kerkyacharian G, Picard D, Raimondo M (2004). "Wavelet Deconvolution in a Periodic Setting." *Journal of the Royal Statistical Society B*, **66**(3), 547–573. With discussion.

Johnstone IM, Raimondo M (2004). "Periodic Boxcar Deconvolution and Diophantine Approximation." *The Annals of Statistics*, **32**(5), 1781–1804.

Kalifa J, Mallat S (2003). "Thresholding Estimators for Linear Inverse Problems and Deconvolutions." *The Annals of Statistics*, **31**, 58–109.

Kerkyacharian G, Picard D, Raimondo M (2007). "Adaptive Boxcar Deconvolution on Full Lebesgue Measure Sets." *Statistica Sinica*, **17**, 317–340.

Kolaczyk ED (1994). "Wavelet Methods for the Inversion of Certain Homogeneous Linear Operators in the Presence of Noisy Data." PhD dissertation. Department of Statistics, Stanford University, Stanford.

Mallat S (1998). *A Wavelet Tour of Signal Processing.* Academic Press Inc., San Diego, CA, 2nd edition. ISBN 0-12-466605-1.

Nason G, Kovac A, Mächler M (2006). **wavethresh***: Software to Perform Wavelet Statistics and Transforms.* R package version 2.2-9.

O'Sullivan F (1986). "A Statistical Perspective on Ill-posed Inverse Problems." *Statistical Science*, **1**, 502–527.

Pensky M, Vidakovic B (1999). "Adaptive Wavelet Estimator for Nonparametric Density Deconvolution." *The Annals of Statistics*, **27**, 2033–2053.

Raimondo M, Stewart M (2006). **waved***: Software to Perform Wavelet Deconvolution.* R package version 1.0, URL http://www.maths.usyd.edu.au/~marcr/.

Raimondo M, Tajvidi N (2004). "A Peaks Over Threshold Model for Change-Points Detection by Wavelets." *Statistica Sinica*, **14**(1), 395–412.

R Development Core Team (2007). *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org/.

Walter GG, Shen X (1999). "Deconvolution Using the Meyer Wavelet." *Journal of Integral Equations and Applications*, **11**, 515–534.

Whitcher B (2006). **waveslim***: Basic Wavelet Routines for One, Two and Three-Dimensional Signal Processing.* R package version 1.6, URL http://www.image.ucar.edu/~whitcher/.

Affiliation:

Marc Raimondo and Michael Stewart
School of Mathematics and Statistics
The University of Sydney
NSW 2006, Austrialia
E-mail: marcr@maths.usyd.edu.au
URL: http://www.maths.usyd.edu.au/~marcr/

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 21, Issue 2 *Submitted:* 2007-03-06
July 2007 *Accepted:* 2007-04-20