The WaveD Transform in R: *performs fast translation-invariant wavelet deconvolution.* [*]

Marc Raimondo and Michael Stewart [†]
University of Sydney

20 February 2007

Abstract

This paper provides an introduction to a software package called `waved` that works within the statistical environment R making available all code necessary for reproducing the figures in the recently published articles on the WaveD method for Wavelet Deconvolution of noisy signals, Johnstone, Kerkyacharian, Picard and Raimondo (2004). 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.

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 $g * f$,

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

where the convolution kernel g is observed with or without noise,

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

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

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

[*]**Key Words and Phrases.** R-packages, WaveD method, deconvolution, Meyer wavelet.
Short Title: The WaveD transform in R.
AMS 2000 Subject Classification: primary 62G05; secondary 62G08

[†]School of Mathematics and Statistics, the University of Sydney, NSW 2006, Australia. marcr@maths.usyd.edu.au

where σ is the noise standard deviation and n is the sample size. We denote $\mathtt{y} = (\mathtt{y_1}, ..., \mathtt{y_n})$ the n observed values $\mathtt{y_i} = Y(t_i), i = 1, \ldots, n$. 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 $\mathtt{g} = (\mathtt{g_1}, ..., \mathtt{g_n})$ the n observed values $\mathtt{g_i} = g_\epsilon(t_i), i = 1, \ldots, n$. An illustration of model (2) in the Fourier domain is given in Figure 8. The WaveD transform as discussed in this paper requires only two input arguments: $\mathtt{y} = (\mathtt{y_1}, ..., \mathtt{y_n})$ and $\mathtt{g} = (\mathtt{g_1}, ..., \mathtt{g_n})$.

1.1 Some references on wavelet deconvolution

Over the last decade many wavelet methods have been developed to recover f from indirect observations: see 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). See also Hall, Ruymgaart, van Gaans and van Rooij (2001); Neelamani, Choi and Baraniuk (2004). The WaveD method of Johnstone, Kerkyacharian, Picard and Raimondo (2004) ([JKPR] in the sequel) specifically addresses the deconvolution problem in the periodic setting (1). The fast implementation of the translation invariant WaveD transform is described in Donoho and Raimondo (2004). The noisy convolution kernel setting (2) and data-driven resolution level tuning of the WaveD method is discussed in Cavalier and Raimondo (2006). Boxcar deconvolution using the WaveD method is discussed in Kerkyacharian, Picard and Raimondo (2006) and Johnstone and Raimondo (2004). Applications of deconvolution models may be found in O'Sullivan (1986) and Bertero and Boccacci (1998).

1.2 What's new?

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 WaveLab802 (`http://www-stat.stanford.edu/wavelab/`).

This paper describes a unified setting where all the WaveD transforms are implemented in the software environment R (R Development Core Team (2006)) via a contributed package named `waved`. The aims of the `waved` package in R 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 et al. (2006): a software to perform wavelet statistics and transforms; the `waveslim` package of Whitcher (2005): basic wavelet routines for one, two and three dimensional signal analysis; the `wavelets` package: 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 using 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, 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,

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

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

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

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 also observed with noise as illustrated on Figure 8.

In this paper ψ (and its periodised version $\Psi(x) = \sum_{k \in \mathtt{Z}} \psi(x + k)$) denotes a Meyer wavelet. Typically ψ is a band limited function whose Fourier transform $F(\psi) := \hat{\psi}$ is smooth, see the formula for the construction of $\hat{\psi}$ page 247 of Mallat (1998). In practice, we use a polynomial function to define the so-called Meyer window, see Mallat (1998) page 248. Throughout this paper we use $\hat{\psi}$ and $\hat{\phi}$ corresponding to a polynomial of degree 3. Let $\Psi_\kappa(x) = 2^{j/2}\Psi(2^j x - k)$ where $\kappa = (j, k)$. The Fourier coefficients satisfy

$$\Psi_\ell^{j,0} = \langle \Psi_{j,0}, e_\ell \rangle = 2^{-j/2}\hat{\psi}(\ell/(2^j \times 2\pi)) \tag{5}$$

and

$$\Psi_\ell^\kappa = \langle \Psi_\kappa, e_\ell \rangle = \exp(2\pi i \ell k / 2^j)\Psi_\ell^{j,0} \tag{6}$$

2.2 The WaveD paradigm

The WaveD paradigm of Johnstone, Kerkyacharian, Picard and Raimondo (2004) [JKPR] 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 whereas $\rightarrow^{\texttt{FWaveD}}, \leftarrow^{\texttt{IWaveD}}$ denotes the Forward WaveD transform and its inverse.

$$\mathrm{FWaveD}(h,g) = (\beta_\kappa)_\kappa, \quad \mathrm{IWaveD}(\beta_\kappa) = \sum_\kappa \beta_\kappa \Psi_\kappa, \tag{7}$$

where $\beta_\kappa = \int f\Psi_\kappa, \quad \kappa = (k,j),\ k = 0,...,2^j-1, j = -1,0,1,...$ with $\Psi_{-1,0} = \Phi$, are the wavelet coefficients of f. Details of the FWaveD and IWaveD transforms in Time, Fourier and Wavelet domain:

$$\begin{tikzcd}
\text{Time domain} & \text{Fourier domain} & & \text{Wavelet domain} \\
h(t) = (f*g)(t) \arrow[r, "F"] & h_\ell = f_\ell \times g_\ell & & \\
\Psi_\kappa(t) & (\Psi^\kappa_\ell) & {} \arrow[r, "\texttt{FWaveD}"] & \int h\bar{\Psi}_\kappa \\
 & & \sum_\ell (h_\ell)\bar{\Psi}^\kappa_\ell & \\
 & & h_\ell \div g_\ell & \textit{(elementwise division)} \\
 & & \sum_\ell f_\ell \bar{\Psi}^\kappa_\ell & \\
 & & & \int f\bar{\Psi}_\kappa := \beta_\kappa \\
f(t) & & {} \arrow[l, "\texttt{IWaveD}"'] &
\end{tikzcd}$$

2.3 Adaptive denoising via non-linear WaveD Transform

A key feature of the WaveD Transform is its ability to deal with noisy data (1), (2). Note that

$$\mathrm{FWaveD}(y,g) = \Big(\sum_\ell (\frac{y_\ell}{g_\ell})\Psi^\kappa_\ell\Big)_\kappa := (\tilde{\beta}_\kappa)_\kappa, \tag{8}$$

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

$$\eta(\tilde{\beta}_\kappa) := \tilde{\beta}_\kappa \times \mathtt{I}(|\tilde{\beta}_\kappa| \geq \lambda) \tag{9}$$

where λ is a threshold parameter. The WaveD estimator [JKPR] (which with a slight abuse of terminology we also call the WaveD transform) is defined as

$$\mathrm{WaveD}(y,g) := \sum_\kappa \eta(\tilde{\beta}_\kappa)\Psi_\kappa(t) := \hat{f}(t), \tag{10}$$

and is illustrated in the diagram below:

$$\begin{tikzcd}
\text{Time domain} & \text{Fourier domain} & \text{Wavelet domain} \\
Y(t) \arrow[r, "F"] & y_\ell = f_\ell\, g_\ell + \varepsilon\xi_\ell \arrow[r, "\texttt{FWaveD}"] & \Big(\sum_\ell(\frac{y_\ell}{g_\ell})\Psi^\kappa_\ell\Big)_\kappa \arrow[d, "\textit{thresholding}"] \\
\hat{f}(t) & {} & (\eta(\tilde{\beta}_\kappa))_\kappa \arrow[l, "\texttt{IWaveD}"']
\end{tikzcd}$$

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 let $T_h f(x) = f(x+h)$ denote the shift operator. For an arbitrary time shift h we define one cycle-spin of WaveD as

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

Let $H_n = \{1/n, 2/n, ..., 1-1/n, 1\}$ be the set of all possible circulant shifts. The Translation Invariant WaveD estimator is defined by

$$\tilde{f}_{TI} = \mathrm{Ave}_{h\in H_n} T_{-h}(\hat{f}_h) = \frac{1}{|H_n|} \sum_{h\in H_n} T_{-h}(\hat{f}_h)\,. \tag{11}$$

2.5 What can WaveD offer?

WaveD is a truly non-linear adaptive algorithm which has near optimal asymptotic properties over a wide range of function classes for a variety of L^p-loss functions, [JKPR]. The translation invariant version of WaveD improves the numerical performances of ordinary WaveD by cycle spinning over all circulant shifts. The fast algorithm which implements the translation invariant version of WaveD takes full advantages of the Fast Fourier Transform and is computed in $O(n(\log n)^2)$ steps only. This makes WaveD an attractive non-iterative deconvolution technique. From the statistical viewpoint WaveD enjoys the benefits of using wavelet expansion such as: being capable of representing function with discontinuities or with non-homogeneous time and frequency behaviour.

3 The `WaveD` Transform in R

3.1 Software access

The `waved` software is provided as an R package obtainable from the Comprehensive R Archive Network (CRAN) 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.

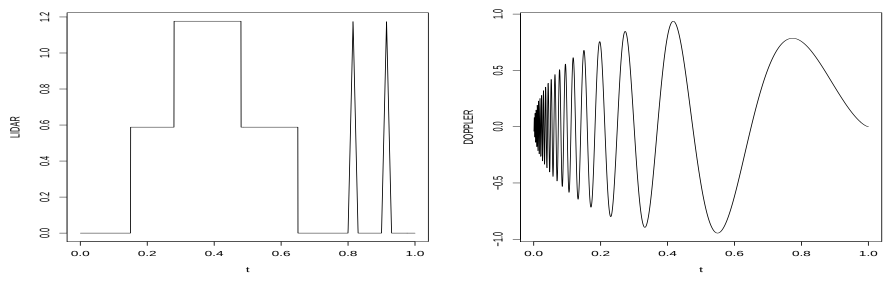

Figure 1: Two signals $t \to f(t)$, $t_i = i/n, i = 1, ..., n = 2048$. Left: LIDAR; right: Doppler.

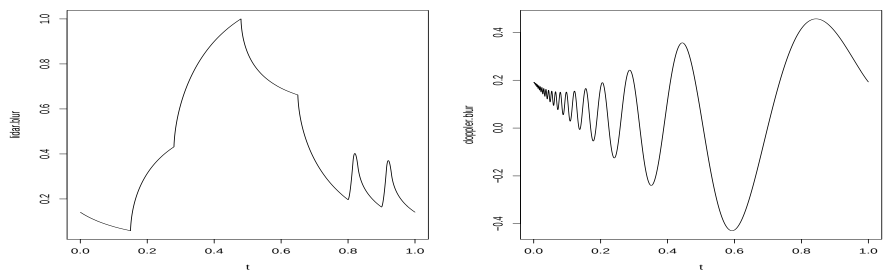

Figure 2: Signals of Figure 1 after smooth blurring with DIP=0.5

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 illustrates 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 `Y` (yes) to get the default setting which produces the following output and ask if you would like to see the Figures of the paper.

```
-------------------------------------------------------
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
-------------------------------------------------------
```

The simulated data are depicted on Figure 1 (Target signals), Figure 2 (blurred signals) and Figure 3 (noisy blurred signals). The noise level default setting (as shown in Figure 3) is `sigma.med` $= 0.05$ 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

$$BSNR_{dB} = 10\ \log_{10}\Big(\frac{||f*g||^2}{\sigma^2}\Big). \qquad (12)$$

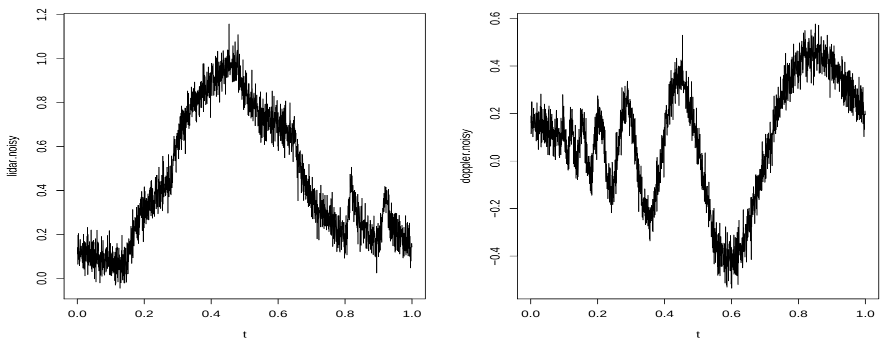

Figure 3: Blurred signals of Figure 2 *plus* noise with s.d $\sigma = 0.05$, in each case the BSNR defined at (12) is approximately $15\,dB$.

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 $|g_\ell| \sim |l|^{-\nu}$ with $\nu = 0.5$. The parameter ν which drives the decay of the eigen-values for high frequencies is often referred as the Degree of Ill-Posedness (DIP) of the convolution problem [JKPR].

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 and Figures with different model parameters: sample size, noise level, Degree of Ill-Posedness, seed and so on...This is done by typing `my.own.simulation=waved.example(F)` and answering questions at the prompt. The list `my.own.simulation` contains the newly simulated data sets: e.g.
`lidar.noisy=my.own.simulation$lidar.noisy`
is a new blurred noisy lidar data set. The list `my.own.simulation` has the following components:

```
> names(my.own.simulation)
 [1] "lidar.noisy"   "lidar.blur"    "doppler.noisy" "doppler.blur"
 [5] "t"             "n"             "g"             "lidar"
 [9] "doppler"       "seed"          "sigma"         "g.noisy"
[13] "dip"           "k.scale"
```

To return to the default setting as used for the figures in this paper type `demo(waved)` and answer `Y` (yes) to get back to the default setting.

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. You can check what `y.wvd` contains by typing `names(y.wvd)`. 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: the blurred data $\mathtt{y} = (\mathtt{y}_1, ..., \mathtt{y}_\mathtt{n})$ as in Figure 2 (or as in Figure 3) and the blurring kernel $\mathtt{g} = (\mathtt{g}_1, ..., \mathtt{g}_\mathtt{n})$. Optional arguments to WaveD include: `F` the finest resolution level j used in the expansion (10) as well as the threshold value λ at (9). The parameter `F` may take any value within the range $\mathtt{L}, ..., (\log_2(n) - 1)$ 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, we set:

```
>lidar.wvd=WaveD(lidar.blur,g,F=6,thr=0)
```

this computes various WaveD transforms of the blurred lidar data of Figure 2 using $j = \mathtt{F} = 6$ as the finest resolution level and threshold $\lambda = \mathtt{thr} = 0$ (no thresholding). Similarly,

```
>lidar.noisy.wvd=WaveD(lidar.noisy,g,F=6,thr=0)
```

computes various WaveD transforms of the noisy-blurred lidar data of Figure 3 using $j = \mathtt{F} = 6$ as the finest resolution level and threshold $\lambda = \mathtt{thr} = 0$ (no thresholding).

The Forward WaveD transform and its inverse are depicted on Figure 4 using the `lidar.wvd` and `lidar.noisy.wvd` objects.

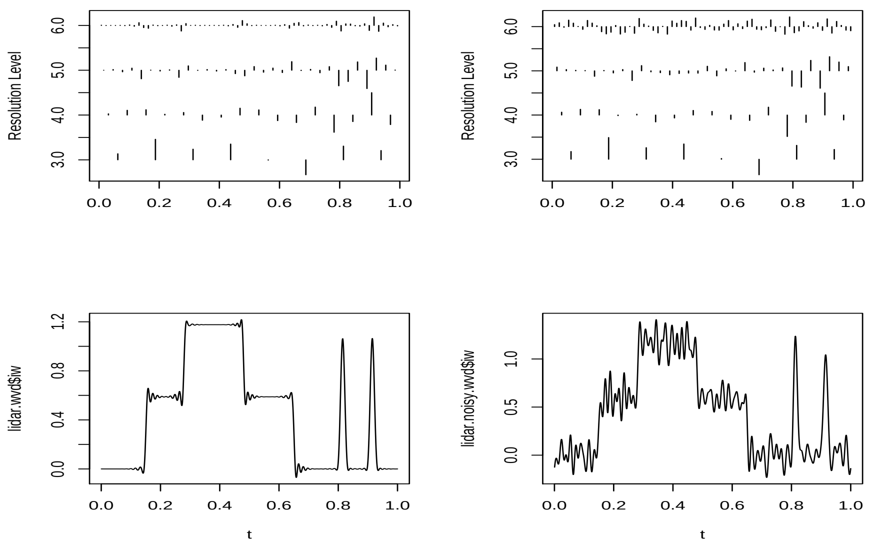

Figure 4: Top plots: Forward WaveD transform, LIDAR wavelet coefficients according to time and resolution (left: `multires(lidar.wvd$w,lo=3,hi=6)`; right: `multires(lidar.noisy.wvd$w,lo=3,hi=6)`). Bottom plots: corresponding Inverse WaveD transforms.

The Forward WaveD transform is obtained from a `wvd` object by typing `lidar.w` = `lidar.wvd$FWaveD` or simply `lidar.w` = `lidar.wvd$w`. There is also an R function called `FWaveD()` which returns the Forward WaveD transform for example `FWaveD(lidar.blur,g,F=6)` returns the same output as `lidar.wvd$w`.

The vector `lidar.w` as defined above 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 get the wavelet coefficients at resolution level `j`:

```
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 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 plot 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 `FWaveD` transform are given in Section 5.

The inverse WaveD transform is obtained from a `wvd` object by typing `lidar.wvd$IWaveD` or simply `lidar.wvd$iw`. The vector `lidar.wvd$iw` returns the inverse WaveD transform (7) computed from `lidar.w` without any thresholding. There is also an R function called `IWaveD` which returns the Inverse WaveD transform computed from a vector of wavelet coefficients. For example `IWaveD(lidar.wvd$w)` returns the same output as `lidar.wvd$iw`. Two illustrations of the inverse WaveD transform are depicted on the bottom plots of 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. The ordinary transform (10) is obtained from a `wvd` object by typing `lidar.wvd$ordinary` or simply `lidar.wvd$ord` which, here, returns approximations to the LIDAR function (as depicted on the RHS bottom plot of 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 whereas the inverse WaveD transform returns the inverse WaveD transform after no 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 in combination with the Translation-Invariant WaveD transform, as described in Section 2.4. This is detailed next.

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 WaveD choose the fine tuning parameters `F` and `thr` in a data-driven fashion in agreement with the optimal choices prescribed in the literature [JKPR], Cavalier and Raimondo (2006). In its simplest form the WaveD function has two input arguments: the blurred data $\mathtt{y} = (\mathtt{y}_1, ..., \mathtt{y}_\mathtt{n})$ as in Figure 2 or in Figure 3 and the blurring kernel $\mathtt{g} = (\mathtt{g}_1, ..., \mathtt{g}_\mathtt{n})$. All other arguments are optional, see section 5 for a complete list. For example,

```
>lidar.maxi.wvd=WaveD(lidar.noisy,g)
```

creates a `wvd` object from the noisy LIDAR data of Figure 3. The fine tuning parameters `F` and `thr` are computed automatically from the data to ensure the best possible performances of the ordinary waved estimate (depicted on the bottom RHS of Figure 6) and of the translation invariant waved estimate (depicted on the RHS of Figure 9).

4.1 Choosing a threshold

The threshold value λ (see (9) or (10)) 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 as shown in Figure 5

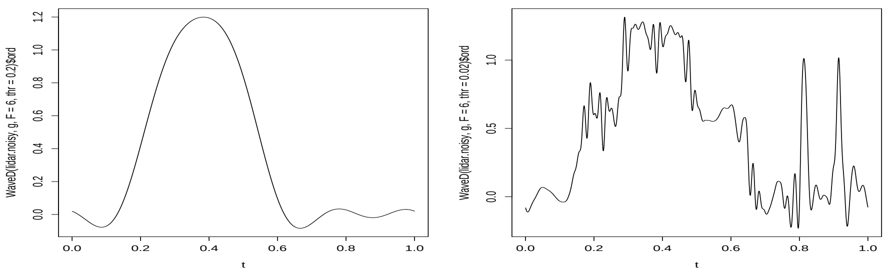

Figure 5: Left, ordinary WaveD with $\lambda = 0.2$ and $F = 6$: `plot(t, WaveD(lidar.noisy, g, F = 6, thr = 0.2)$ord)`. Right, ordinary WaveD with $\lambda = 0.02$ and $F = 6$: `plot(t, WaveD(lidar.noisy, g, F = 6, thr = 0.02)$ord)`

Alternatively a set of level dependent thresholds may be entered as a vector, for example: `WaveD(lidar.noisy,g,C=3,F=6,thr=c(0.01,0.02,0.03,0.04)` will use $\lambda = 0.01$ at resolution level $j = 3$, $\lambda = 0.02$ at resolution level $j = 4$ and so on...

Maxiset threshold: if no threshold parameter is specified the WaveD function will compute and use the so-called "Maxiset threshold". This threshold is derived from the Maxisets Theory [JKPR]. The numerical values of `thr` may be obtained from a `wvd` object by typing `lidar.maxi.wvd$thr` which here returns the following values:

```
>0.014 0.021 0.031 0.048 0.075
```

corresponding to a vector $(\lambda_3, ..., \lambda_7)$ of level dependent thresholds computed as

$$\lambda_j = \hat{\sigma}\gamma\,\sigma_j\,c_n \tag{13}$$

- $\hat{\sigma}$: 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. In `R (WaveD)` type `scale(data)` to get $\hat{\sigma}$.

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

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

$$\sigma_j := \tau_j(x_\ell) = \Big(|C_j|^{-1}\sum_{l\in C_j}|x_\ell|^{-2}\Big)^{1/2} \tag{14}$$

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

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

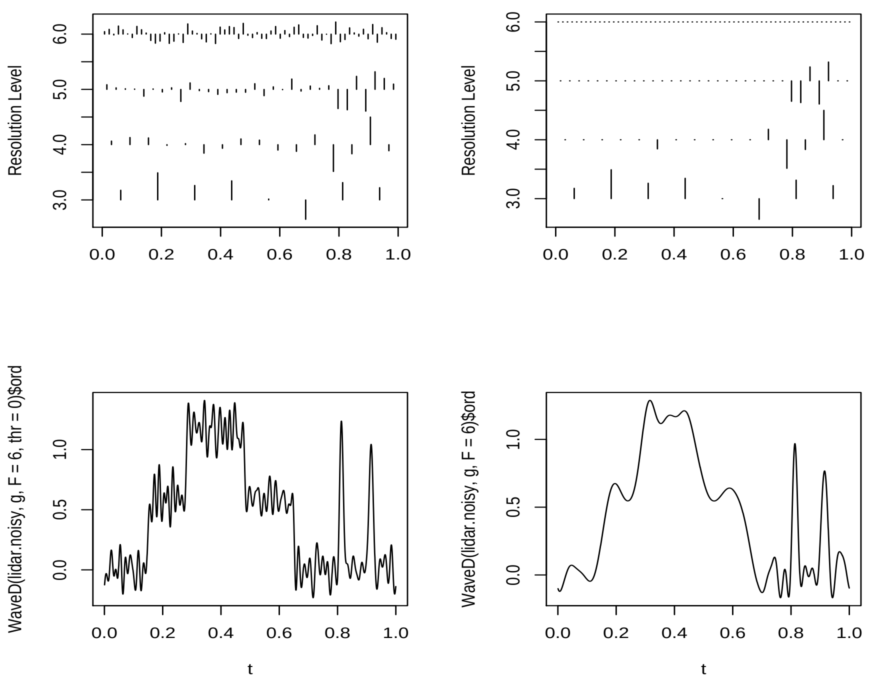

Figure 6: Top left: Forward WaveD transform (un-thresholded), $\texttt{multires(lidar.maxi.wvd\$w, lowest} = 3, \texttt{highest} = 6)$. Top right: Forward WaveD transform after maxisets-thresholding, $\texttt{multires(lidar.maxi.wvd\$w.thr, lo} = 3, \texttt{hi} = 6)$. Bottom left: Ordinary WaveD transforms with no thresholding. Bottom right: Ordinary WaveD transforms with maxisets-thresholding.

The effect of the Maxiset threshold is illustrated on Figure 6 for the LIDAR data. 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 smooths the estimate. The thresholded Forward WaveD transform may be obtained from a `wvd` object: for the LIDAR example `lidar.maxi.wvd$w.thr` returns the Forward WaveD transform after maxiset thresholding as depicted Figure 6.

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 $2^{\texttt{F}} \approx M$. The tuning parameter `F` stipulates the range of resolution levels where the approximations (10) or (11) are used:

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

A numerical value for `F` within the range $\texttt{L} \le \texttt{F} \le \log_2(n) - 1$ may be entered directly in the WaveD function as shown in Figure 5 or as in the example of section 3.5. Here `L` is a low resolution parameter (default is `L=3`). Unlike direct estimation problems e.g. Donoho et al. (1995) where it is customary to keep all resolution levels setting $\texttt{F} = \log_2(n) - 1$, the asymptotic theory for deconvolution [JKPR] shows that one should stop at a fine resolution level $\texttt{F} = j_1$ where j_1 depends on the degree of ill-posedness of the convolution kernel (the faster the eigen values go to zero the sooner the wavelet expansion should stop). In other words 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. In practical terms this means that, even after Maxiset thresholding, the WaveD estimate based on all resolution levels may, sometimes, contain high noise perturbations, in our LIDAR example

```
> lidar.Fmax.wvd=WaveD(lidar.noisy,g,F=10)
```

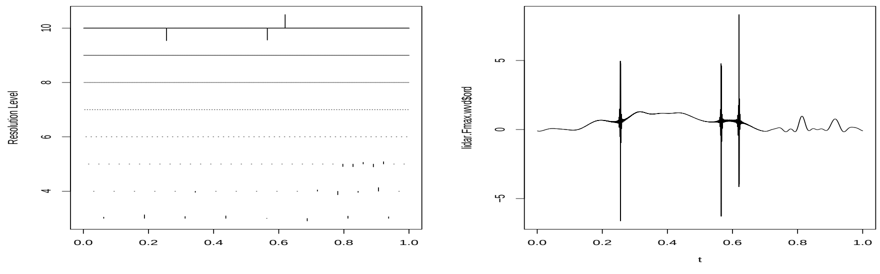

Figure 7: Left: `multires(lidar.wvd.Fmax$w.thr)`. Right: `plot(t, lidar.wvd.Fmax$ord)`.

computes the WaveD approximation with maxiset threshold and largest possible resolution level `F` = 10 (here $n = 2048$). As seen on Figure 7 there are large noise residuals in the WaveD estimate due to large (unthresholded) coefficients at resolution level 10.

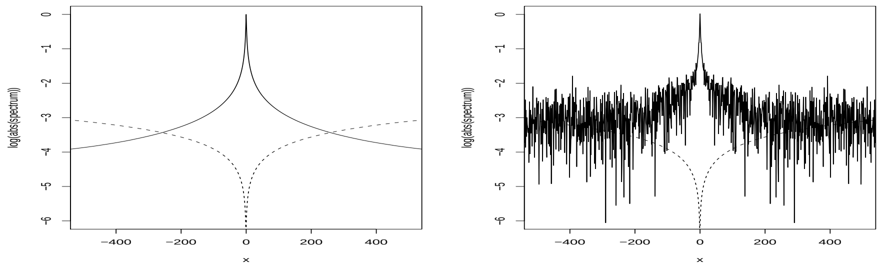

Figure 8: Illustration of the fine level selection in the Fourier domain (15). Left: $\ell \longrightarrow \log|g_\ell|$, where g_ℓ are noise-free eigen-values ($\epsilon = 0$). Here $M = 191$, $\hat{j}_1 = 6$. Right: $\ell \longrightarrow \log|x_\ell|$ where $x_\ell = g_\ell + \epsilon\xi_\ell$ with $\epsilon = \varepsilon = 0.05/\sqrt{2048}$. Here $M = 74$, $\hat{j}_1 = 5$.

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 (2006) 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) the eigen values fall below an appropriate noise level. This is illustrated on Figure 8. Let

$$M = \min\left\{\ell, \ell \geq 0 : |x_\ell| \leq \ell^{1/2}\,\varepsilon\,(\log 1/\varepsilon^2)\right\}, \tag{15}$$

denote the maximum Fourier frequency allowed in the WaveD formula (7). Then we define the maximum wavelet resolution level as

$$\hat{j}_1 = \lfloor \log_2(M) \rfloor - 1, \tag{16}$$

where $\lfloor x \rfloor$ is the largest integer below x. This process is illustrated on Figure 8. To get the numerical value of j_1 or M from a `lidar.wvd` object type `lidar.wvd$j1` or `lidar.wvd$M`.

4.3 Improving the fit using the TI-WaveD transform

While thresholding wavelet coefficients reduces the noise and smooths the WaveD estimate it also introduces Gibbs phenomenon near discontinuities see e.g. RHS bottom plots of Figure 6. Such Gibbs effects can be reduced by cycle spining Donoho and Raimondo (2004).

The translation invariant WaveD transform (11) (which performs a full-cycle spin) is obtained from a `wvd` object by typing `lidar.maxi.wvd$ti` or `lidar.maxi.wvd$waved`.

In any case where some thresholding is performed we recommend using the `TI-WaveD` transform as it reduces visual artifacts in WaveD estimate. This is illustrated on Figure 9.

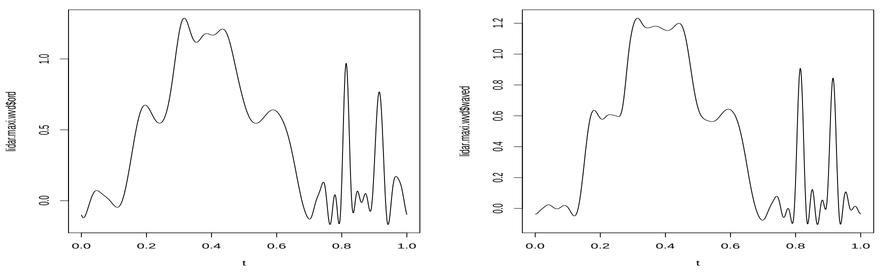

Figure 9: Left, ordinary WaveD: `plot(t,lidar.maxi.wvd$ord)`. Right, TI-WaveD: `plot(t,lidar.maxi.wvd$waved)`

The MC-option. The algorithm which implement the `TI-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 implement 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 heavy simulations and Monte-Carlo approximations, it is possible to set `MC=TRUE`, in this case the WaveD function will only return the translation invariant WaveD estimate.

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 (9) or the `Soft` thresholding policy:

$$\eta_S(\tilde{\beta}_\kappa) := \mathrm{sign}(\tilde{\beta}_\kappa)(|\tilde{\beta}_\kappa| - \lambda) \times \mathtt{I}(|\tilde{\beta}_\kappa| \geq \lambda) \tag{17}$$

where λ is a threshold parameter. The statistical theory for WaveD estimation [JKPR], Kerkyacharian et al. (2006), Cavalier and Raimondo (2006) is established for the `Hard` threshold policy (9) which is the default setting in WaveD. However, for data analysis purposes and experimental study we provide a `Soft` thresholding option (17) in the WaveD function, this is illustrated on Figure 10. As seen on Figure 10 `Soft` thresholding tends to further smooth the WaveD estimator but the general appearance does not appear as sharp as the TI-WaveD estimate of Figure 9.

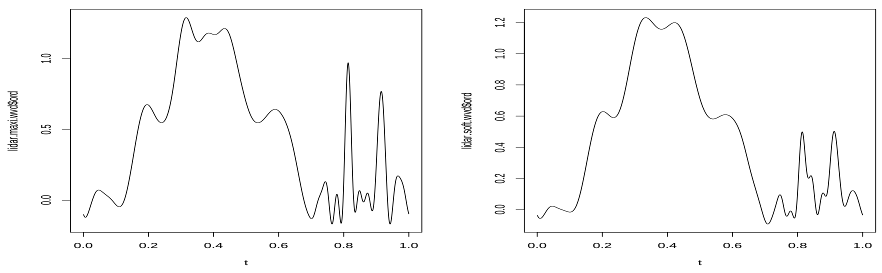

Figure 10: Left, ordinary WaveD with `Hard` thresholding:`plot(t, lidar.wvd$ord)`. Right, ordinary `WaveD` with `soft` thresholding : `plot(t,WaveD(lidar.noisy,g,SOFT=TRUE))` .

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

For convenience we provide a `summary` and `plot` function specifically for objects of class `wvd`. We illustrate these functions using the noisy Doppler example of Figure 3. First we create a `wvd` object for the `doppler.noisy` data: `doppler.wvd=WaveD(doppler.noisy,g)`, then `summary(doppler.wvd)` gives

```
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
level 4     0.167     0.020             0.625
level 5     0.128     0.030             0.906
level 6     0.078     0.046             0.969
Noise-proxy statistics:
Estimated standard deviation=  0.049
Shapiro test for normality, P= 0.88634
```

In addition to providing the tuning parameters `F`, `thr`, M, γ and 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.

4.6 Assessing the WaveD fit

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, see e.g. Donoho et al. (1995) or Raimondo and Tajvidi (2004). Here we call the vector of wavelet coefficients at the largest resolution level: `noise.proxy`. This vector may be obtained from a `wvd` object by typing `lidar.wvd$noise`.

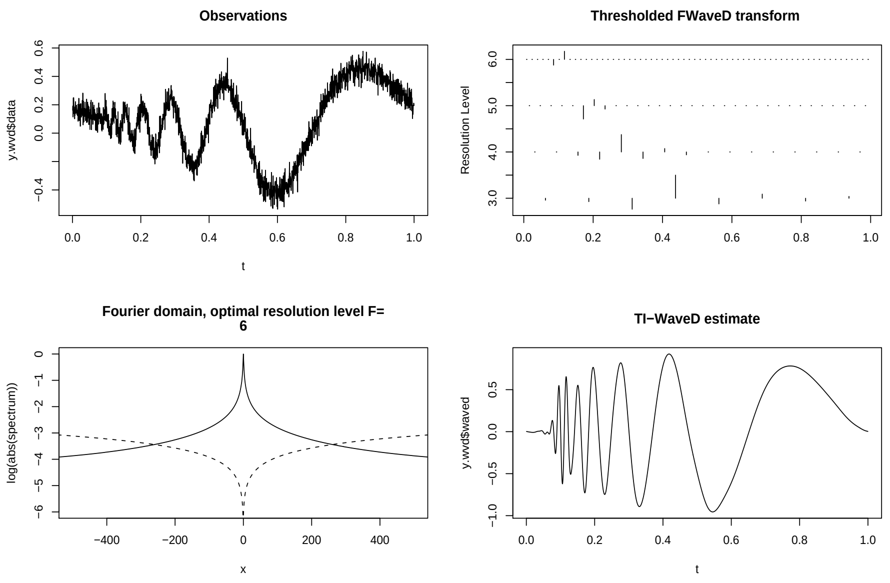

Figure 11: A typical plot of an object of class `wvd`, `plot(doppler.wvd)`

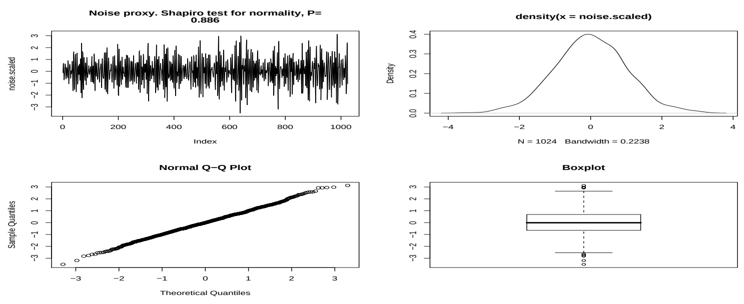

Figure 12: Assessing the WaveD, `plot(doppler.wvd)` (2nd plot).

The `summary` and `plot` function use the `noise.proxy` vector to perform some elementary data analysis see Figure 12.

WaveD-fit assessment. The asymptotic theory of [JKPR] and Cavalier and Raimondo (2006) is based on the white noise model (1) in which the error terms follow a normal distribution. A close inspection to the proof of [JKPR] 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 result 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.

4.7 WaveD estimation with noisy eigen values

We finish this section by illustrating further adaptive properties of WaveD estimates. Depicted on Figure 13 is the WaveD LIDAR estimate constructed from noisy-blurred data as in Figure 3 and noisy eigen-values as in the RHS plot of Figure 8. By comparing with

Figure 9 we see that the quality of the WaveD approximation is not much affected if one use noisy eigen values instead of the true eigen values. This is consistent with the asymptotic theory of Cavalier and Raimondo (2006).

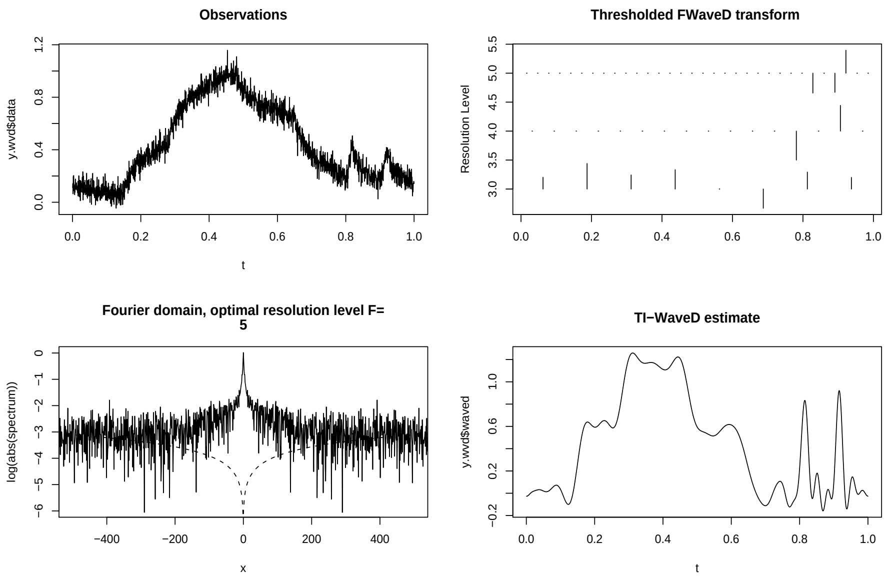

Figure 13: Lidar WaveD estimate when eigen-values are noisy.

5 R commands for WaveD transform

5.1 The WaveD command

$\texttt{WaveD} < -\texttt{function}(\texttt{yobs},\texttt{g})\{...\}$

```
# Performs wavelet deconvolution using Meyer wavelet.
# If g is not specified WaveD performs  a  wavelet transform.

Inputs (REQUIRED)
    yobs: Sample of f*g+Noise
Inputs (OPTIONAL)
    g:    Sample of g or g+Noise (default is direct mass at 0)--same length as yobs
    L:    Lowest resolution level (default=3)
    F:    Finest resolution level (default=data driven choice of j1)
    deg:   deg  of the Meyer Wavelet deg=1,2, or 3 (default=3)
    eta: smoothing parameter (default=conservative sqrt(6))
    MC: if Monte Carlo (MC=TRUE) WaveD returns only the TI-WaveD (default=FALSE)
    --note if MC=TRUE 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 (default is Maxiset Threshold)--length=1 or length=F-L+2

Outputs: object of class wvd, list with following components
   j1: estimate of  optimal resolution level (for Maxiset Threshold)
   F: Fine resolution level used (may be different than j1)
   M:  estimate of optimal Fourier frequency (for Maxiset Threshold)
```

```
thr: threshold (default is Maxiset Threshold)
w or FWaveD: Forward WaveD Transform (before thresholding)
w.thr:  Forward WaveD Transform (after thresholding)
iw or IWaveD: Inverse WaveD Transform (based on w)
ordinary: ordinary WaveD transform
waved or ti: translation invariant WaveD transform
percent: percent of thresholding per resolution level
s: estimate of noise sd
noise: noise proxy
ps: P-value of Shapiro normality test for noise proxy
residuals: wavelet coefficients that have been removed before fine level F
```

Example: `y.wvd=WaveD(lidar.noisy,g)`

References Johnstone, Kerkyacharian, Picard and Raimondo (2004); Donoho and Raimondo (2005); Kerkyacharian, Picard and Raimondo (2006); Cavalier and Raimondo (2006)

5.2 Other useful command

`FWaveD` has the same inputs as WaveD. The command `lidar.w=FWaveD(lidar.blur,g)` returns the same output as `WaveD(lidar.blur,g)$w` i.e. a vector of wavelet coefficients for the `LIDAR` function. 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 `dyad` below for how to access wavelet coefficient at a given resolution level.

`dyad(j)` returns integers $2^j+1, ..., 2^{j+1}$, hence the command the command `lidar.w[dyad(7)]` returns the LIDAR wavelet coefficients at resolution level 7.

`multires(lidar.w,lo=3,hi=7)` depicts wavelet coefficients according to time and resolution level 3,4,..7. See Figure 4

`maxithresh(lidar.noisy,g,L=3,F=7)` returns the maxiset thresholds.

`scale(lidar.noisy)` returns an estimate of the noise standard deviation.

`find.j1(g,scale(lidar.noisy))` returns the optimal Fourier frequency and optimal resolution level for using the maxiset threshold with the `lidar.noisy` data. See Figure 8.

`IWaveD(lidar.w)` returns the inverse `WaveD` transform based on the vector of wavelet coefficients `lidar.w`. The `IWaveD` function can be used to construct/plot wavelets $\Psi_{j,k}$. One just need to create a zero vector and put a one in the appropriate index ($ind = 2^j + k + 1$), as given by the function `dyadjk(j,k)`). For example

```
>wL=rep(0,2048); wR=rep(0,2048); wL[dyadjk(4,3)]=1; wR[dyadjk(6,40)]=1;
>plot(t,IWaveD(wL,3),type='l'); plot(t,IWaveD(wR,3),type='l')
```

plot of the $\Psi_{4,3}, \Psi_{6,40}$ Meyer wavelets, $n = 2048$.

References

Abramovich, F. and Silverman, B. (1998), 'Wavelet decomposition approaches to statistical inverse problems', *Biometrika* **85**(1), 115–129.

Aldrich, E. (2005), *wavelets: A package of funtions for computing wavelet filters, wavelet transforms* , R Contributed pakckage.
URL: *http://cran.au.r-project.org/src/contrib/Descriptions/wavelets.html*

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

Cavalier, L. and Koo, J.-Y. (2002), 'Poisson intensity estimation for tomographic data using a wavelet shrinkage approach', *IEEE Trans. Inform. Theory* **48**, 2794–2802.

Cavalier, L. and Raimondo, M. (2006), 'Wavelet deconvolution with noisy eigen-values', *IEEE Trans. Signal Process*, 20 pages, to appear.

Donoho, D. (1995), 'Nonlinear solution of linear inverse problems by wavelet-vaguelette decomposition', *Applied Computational and Harmonic Analysis* **2**, 101–126.

Donoho, D. L., Johnstone, I. M., Kerkyacharian, G. and Picard, D. (1995), 'Wavelet shrinkage: Asymptopia?', *Journal of the Royal Statistical Society, Series B* **57**, 301–369. With Discussion.

Donoho, D. L. and Raimondo, M. E. (2005), 'A fast wavelet algorithm for image deblurring', *ANZIAM J.* **46**, C29–C46. http://anziamj.austms.org.au/V46/CTAC2004/Dono.

Donoho, D. and Raimondo, M. (2004), 'Translation invariant deconvolution in a periodic setting', *The International Journal of Wavelets, Multiresolution and Information Processing* **14**(1), 415–423.

Fan, J. and Koo, J. (2002), 'Wavelet deconvolution', *IEEE Transactions on Information Theory* **48**(3), 734–747.

Hall, P., Ruymgaart, F., van Gaans, O. and van Rooij, A. (2001), Inverting noisy integral equations using wavelet expansions: A class of irregular convolutions, *in* 'State of the Art in Probability and Statistics: Festschrift for Willem R. van Zwet', Vol. 36 of *Lecture Notes-Monograph Series*, Institute of Mathematical Statistics, pp. 533–546.

Johnstone, I., Kerkyacharian, G., Picard, D. and Raimondo, M. (2004), 'Wavelet deconvolution in a periodic setting', *Journal of the Royal Statistical Society, Series B* **66**(3), 547–573. with discussion pp.627-652.

Johnstone, I. M. (1999), 'Wavelet shrinkage for correlated data and inverse problems: adaptivity results', *Statistica Sinica* **9**(1), 51–83.

Johnstone, I. M. and Raimondo, M. (2004), 'Periodic boxcar deconvolution and diophantine approximation', *Annals of Statistics* **32**(5), 1781–1804.

Kalifa, J. and Mallat, S. (2003), 'Thresholding estimators for linear inverse problems and deconvolutions', *Annals of Statistics* **31**, 58–109.

Kerkyacharian, G., Picard, D. and Raimondo, M. (2006), 'Adaptive boxcar deconvolution on full lebesgue measure sets', *Statistica Sinica* p. 21 pages. To appear.

Kolaczyk, E. (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 (2nd Edition)*, Academic Press Inc., San Diego, CA.

Nason, G. and Kovac, A. and Maechler, M. (2006), *wavethresh: Software to perform wavelet statistics and transforms.*, R Contributed pakckage.
URL: *http://cran.au.r-project.org/src/contrib/Descriptions/wavethresh.html*

Neelamani, R., Choi, H. and Baraniuk, R. (2004), 'Forward: Fourier-wavelet regularized deconvolution for ill-conditioned systems', *IEEE Transactions on signal processing* **52**, 418–433.

O'Sullivan, F. (1986), 'A statistical perspective on ill-posed inverse problems', *Statistical Science* **1**, 502–527.

Pensky, M. and Vidakovic, B. (1999), 'Adaptive wavelet estimator for nonparametric density deconvolution', *Annals of Statistics* **27**, 2033–2053.

R Development Core Team (2006), *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*

Raimondo, M. and Tajvidi, N. (2004), 'A peaks over threshold model for change-points detection by wavelets', *Statistica Sinica* **14**(1), 395–412.

Walter, G. and Shen, X. (1999), 'Deconvolution using the meyer wavelet', *Journal of Integral Equations and Applications* **11**, 515–534.

Whitcher, B. (2005), *waveslim: Basic wavelet routines for one-, two- and three-dimensional signals*, R Contributed pakckage.
URL: *http://cran.au.r-project.org/src/contrib/Descriptions/waveslim.html*