Lab 9: An Introduction to Wavelets

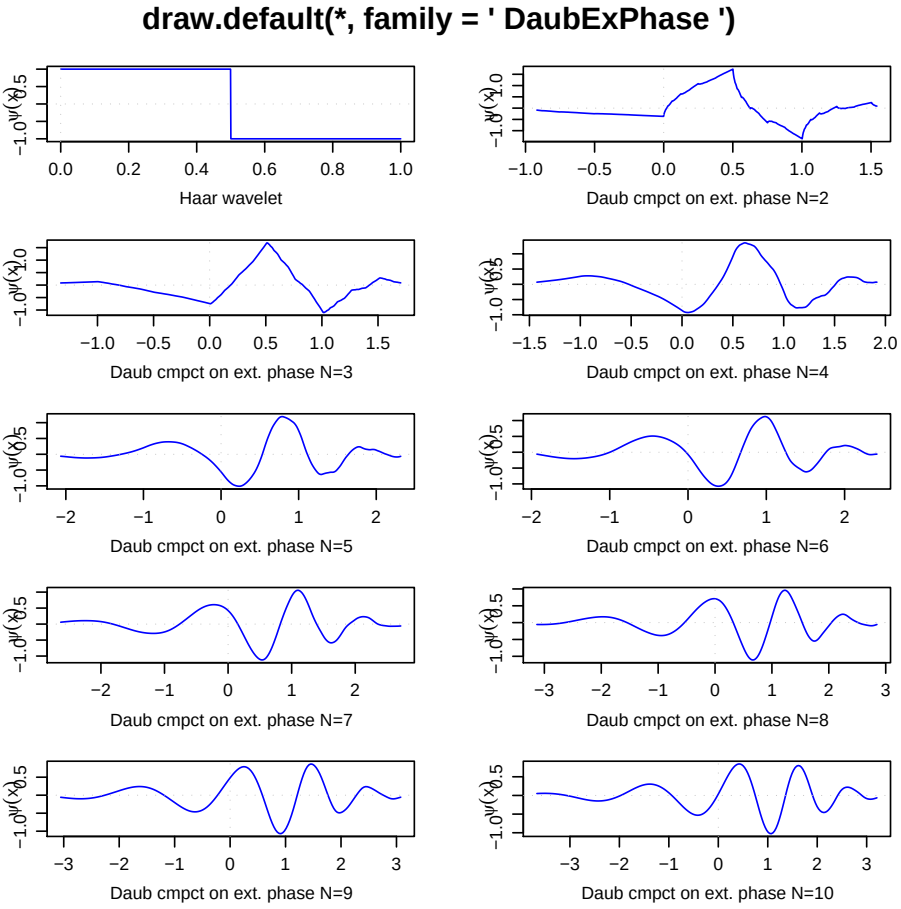
June 5, 2003

In this lab, we introduce some basic concepts of wavelet theory and present the essentials required for possible applications of wavelet decompositions for the analysis of microarray data. We first explore some wavelet families that have been proved useful in practice using the `wavethresh` package and then present basic wavelet methods for approximation, compression and smoothing.

```
> library(wavethresh)
> library(Rwave)
> library(ts)
> library(Milan)
```

Any discussion of wavelets begins with Haar, the simplest. Ingrid Daubechies developed the family of compactly supported wavelets – thus making discrete wavelet analysis practicable. What follows gives describes graphically several of the most useful wavelets from the Daubechies compactly supported family. The names of the Daubechies family wavelets are named by their filters N. The $N = 1$ wavelet is the same as Haar.The following commands display the first 10 members of this family.

```
> op = par(mfrow = c(5, 2), oma = c(0, 0, 4, 0), mgp = c(1.2,
+     0.8, 0), mar = 0.1 + c(4, 4, 0.5, 1))
> for (fn in 1:10) {
+     draw.default(filter.number = fn, col = "blue", main = NULL,
+         xlab = "")
+     abline(h = 0, v = 0, lty = 3, lwd = 0.5, col = "gray")
+ }
> mtext(paste("draw.default(*, family = '", formals(draw.default)$family[[2]],
+     "')"), side = 3, line = 1, outer = TRUE, cex = par("cex.main"),
+     font = par("font.main"))
> par(op)
```

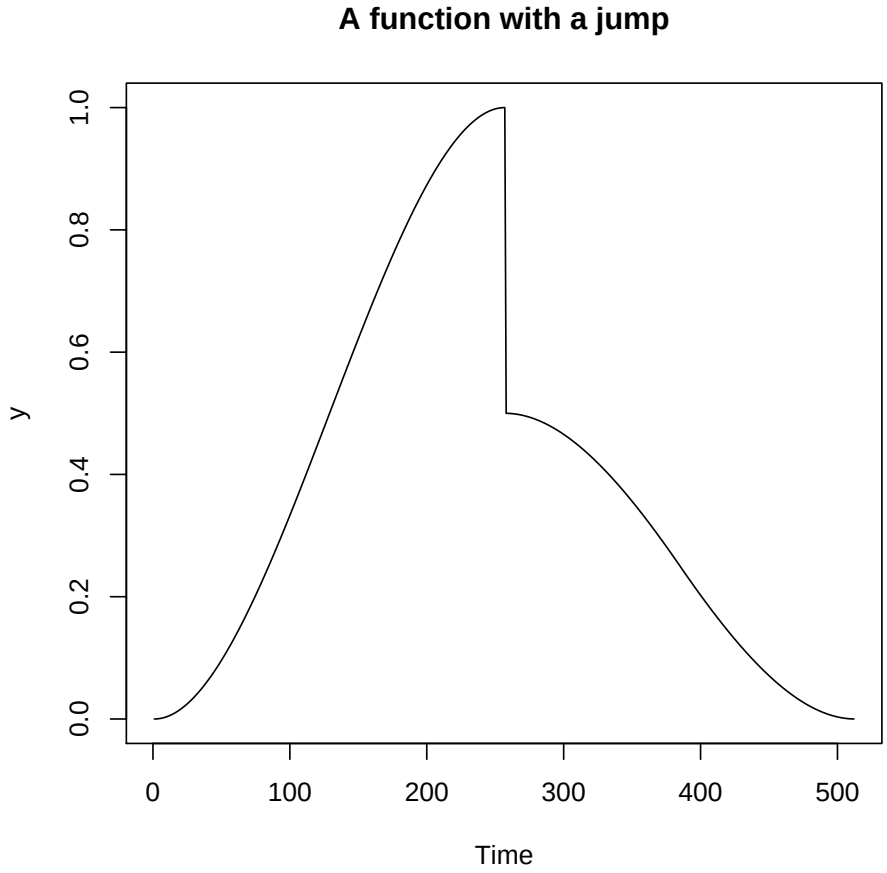


In practical applications, the choice of wavelet family and index N is not terribly important. Notice how the support of the functions increases with their index.

In order to appreciate the application of wavelets to various function estimation problems in statistics, it is first useful to examine the basic principles of wavelet analysis by means of the discrete wavelet transform and the concept of multiresolution analysis.

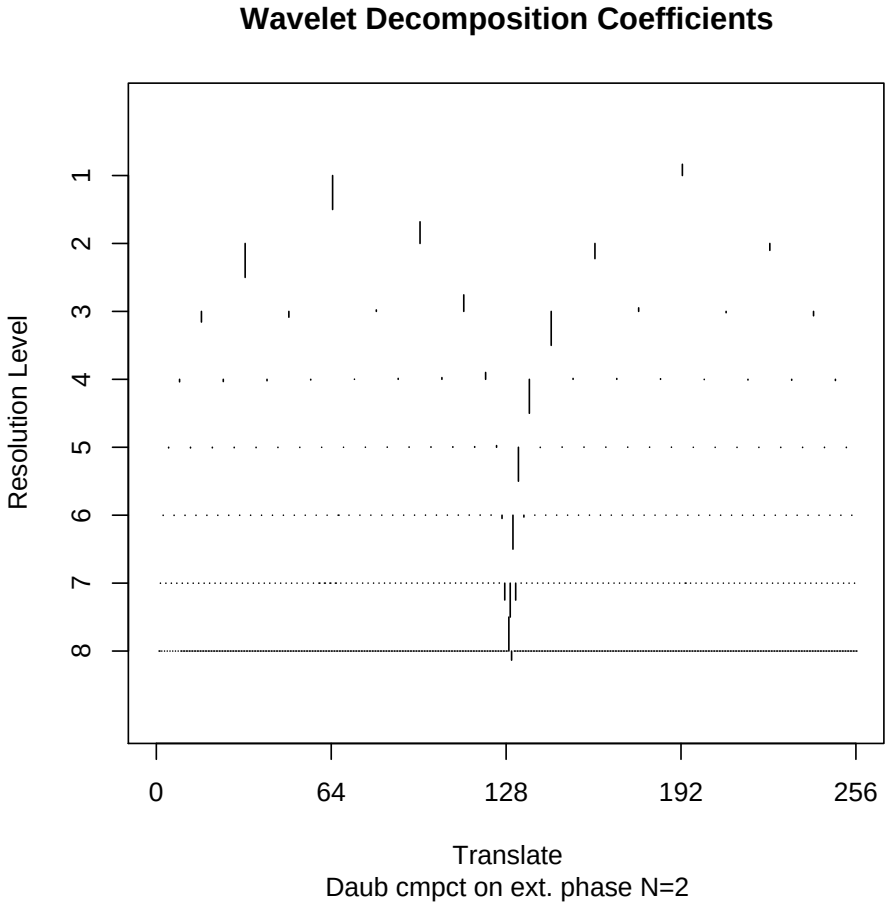
Let us create a signal with some singularities and display it.

```
> test.data1 = example.1()$y
> ts.plot(test.data1, ylab = "y")
> title("A function with a jump")
```



The following commands produce a *Wavelet decomposition-plot* of the wavelet coefficients at different resolution levels (scales). The plot results from applying the *discrete wavelet transform* with a Daubechies $N = 2$ wavelet to a set of 512 equally spaced values from the function displayed previously. The number of coefficients at each level is exactly half that of the next higher level, so there are 256 coefficients at level 8, 128 at level 7, and so on.

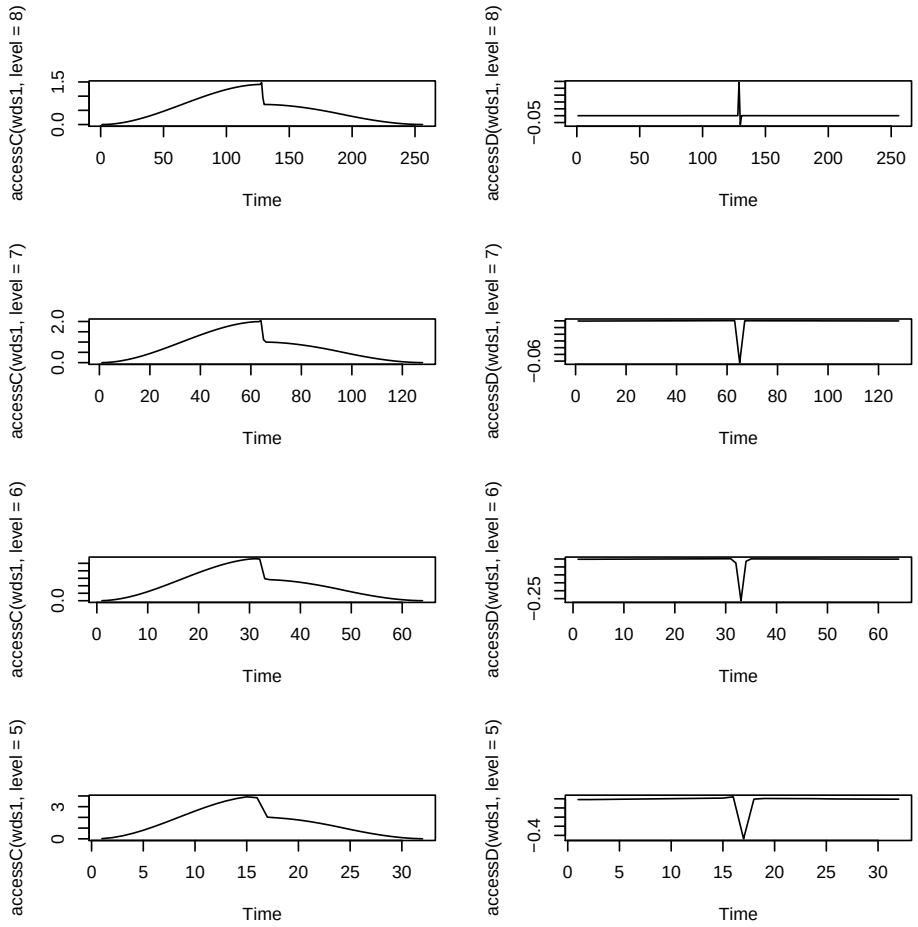
```
> wds1 <- wd(test.data1)
> plot(wds1)
```



Such a plot provides a good description of where significant changes are taking place in the function. The location of the abrupt jump can be spotted by looking for vertical clusterings of relatively large coefficients.

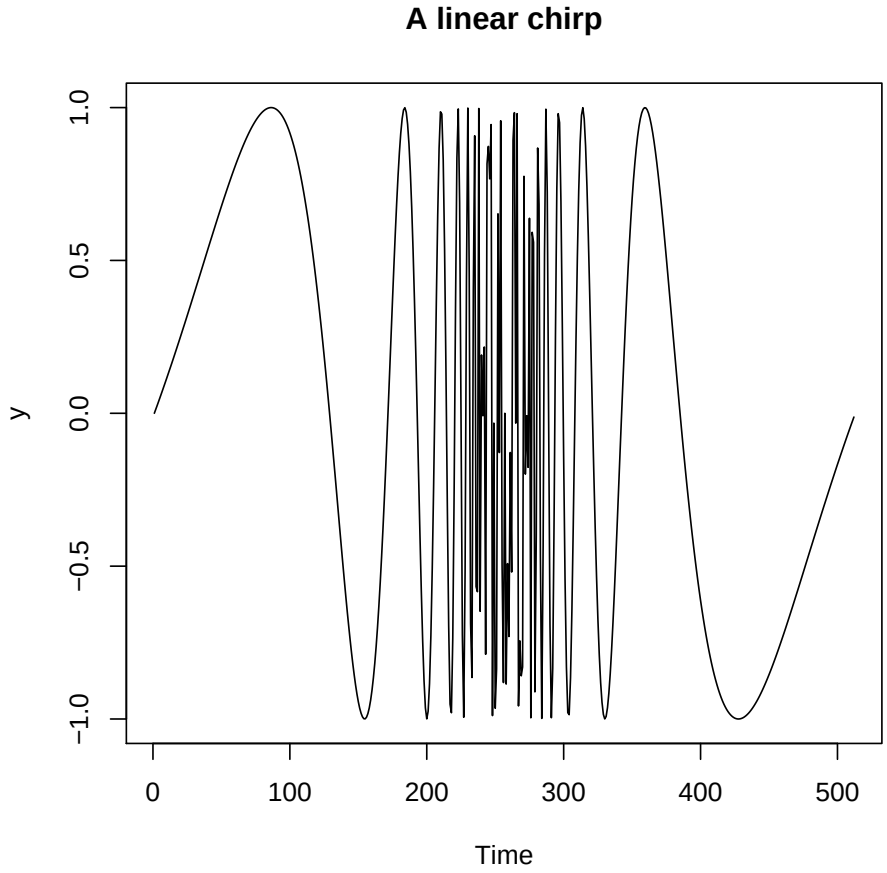
The next plot gives a schematic diagram of the decomposition process underlying a multiresolution analysis. The original signal is broken down into successive approximations (plots on the left) and details (plots on the right). Adding the details to the approximations at a given row produces a better approximation at the next higher level. Such a plot is called a *wavelet decomposition tree*.

```
> par(mfrow = c(4, 2))
> plot.ts(accessC(wds1, level = 8))
> plot.ts(accessD(wds1, level = 8))
> plot.ts(accessC(wds1, level = 7))
> plot.ts(accessD(wds1, level = 7))
> plot.ts(accessC(wds1, level = 6))
> plot.ts(accessD(wds1, level = 6))
> plot.ts(accessC(wds1, level = 5))
> plot.ts(accessD(wds1, level = 5))
```



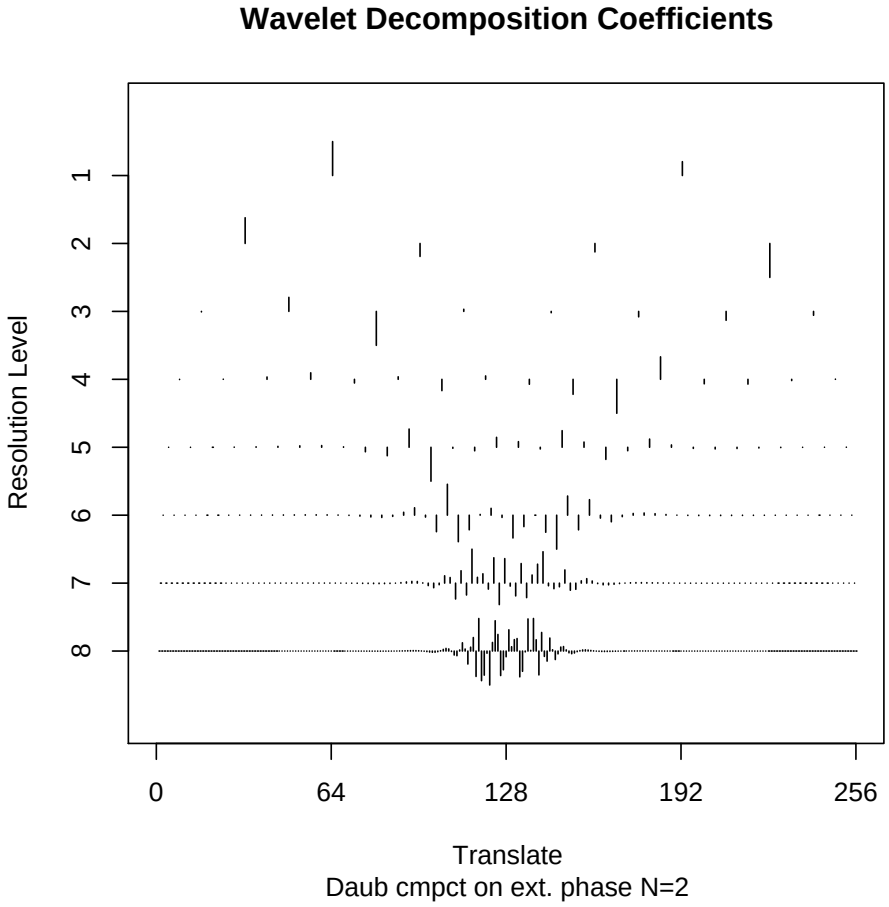
We illustrate the above by treating another example. The analyzed signal is a chirp.

```
> par(mfrow = c(1, 1))
> test.data2 <- achirp(512)$y
> ts.plot(test.data2, ylab = "y")
> title("A linear chirp")
```



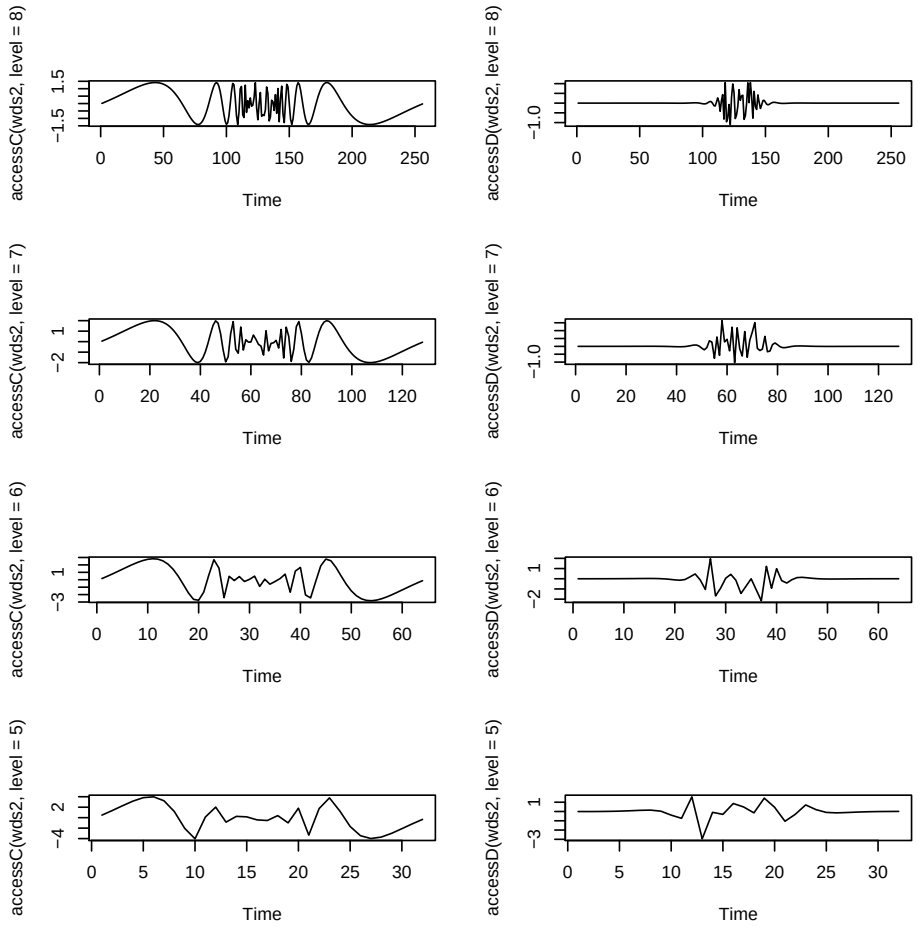
Its discrete wavelet transform

```
> wds2 <- wd(test.data2)
> plot(wds2)
```



and the corresponding wavelet decomposition tree

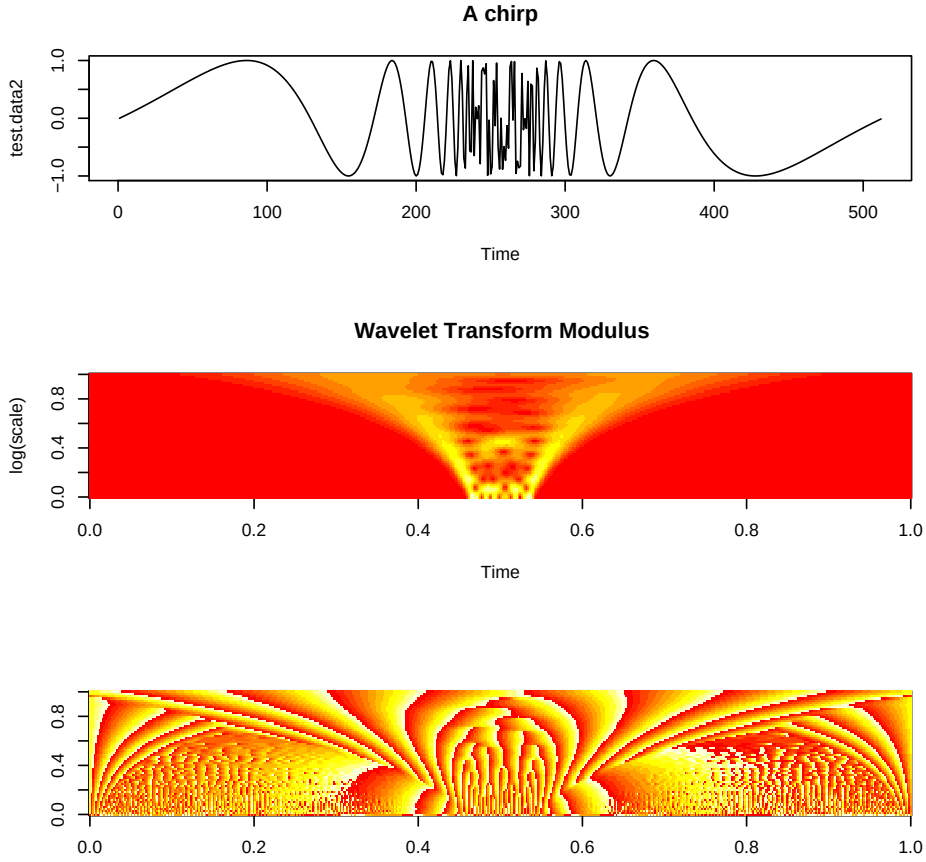
```
> par(mfrow = c(4, 2))
> plot.ts(accessC(wds2, level = 8))
> plot.ts(accessD(wds2, level = 8))
> plot.ts(accessC(wds2, level = 7))
> plot.ts(accessD(wds2, level = 7))
> plot.ts(accessC(wds2, level = 6))
> plot.ts(accessD(wds2, level = 6))
> plot.ts(accessC(wds2, level = 5))
> plot.ts(accessD(wds2, level = 5))
```



To get a clearer picture of the frequency contents of the chirp signal we produce a time-scale view of it using instead the *continuous wavelet transform* available in the *Rwave* package.

The corresponding plot displays the original signal, the image plot of the modulus of the (complex valued) continuous wavelets coeffcients and their phase (bottom figure). How to make sense of all these coefficients? The x-axis represents positions along the signal (time), the y-axis represents scale (frequency), and the color at each x-y point represents the magnitude of the wavelet coefficient.

```
> par(mfrow = c(3, 1))
> plot.ts(test.data2)
> title("A chirp")
> cwtchirp <- cwt(test.data2, 5, 12)
> image(Arg(cwtchirp))
```



The next chunk illustrates the good compression properties of the wavelet transform. One needs to use the `test.data1` example again. Any wavelet coefficients whose magnitude, in absolute value, is smaller than 0.06, is thresholded to zero. There are in total 491 coefficients from the original 512 that are set to 0.

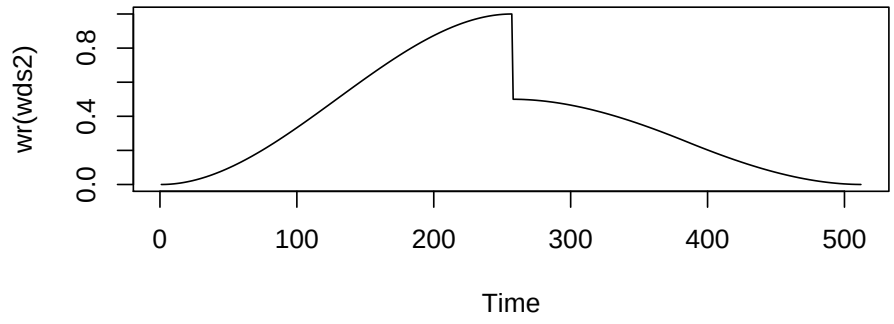
```
> levels <- 3:(wds1$nlevels - 1)
> nthresh <- length(levels)
> d <- NULL
> dz <- 0
> for (i in 1:nthresh) {
+     d <- accessD(wds1, level = levels[i])
+     d[abs(d) <= 0.06] = 0
+     dz = dz + sum(d == 0)
+     cat("Level: ", levels[i], " there are ", sum(d == 0), " zeroes\n")
+     wds2 <- putD(wds1, level = levels[i], v = d)
+ }
```

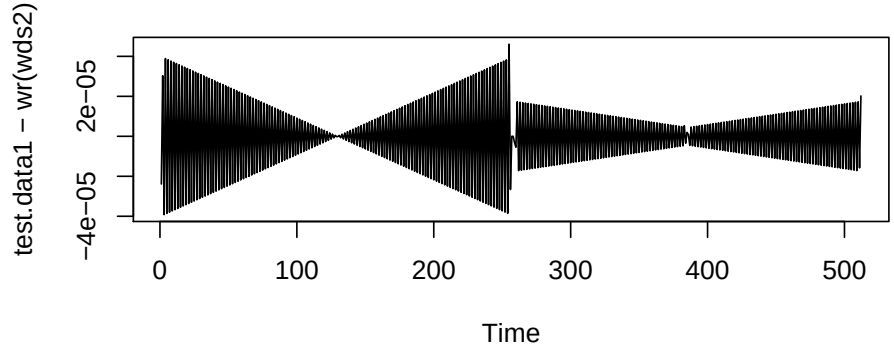
```
Level:  3  there are  2  zeroes
Level:  4  there are  14  zeroes
```

```
Level:  5  there are  31  zeroes
Level:  6  there are  63  zeroes
Level:  7  there are  127  zeroes
Level:  8  there are  254  zeroes

> cat("there are in total", dz, " zeroes\n")

there are in total 491  zeroes
```

The figure displays the resulting approximation obtained by inverting the wavelet transform of the thresholded signal (using the function `wr`) and its difference with the original signal.

```
> par(mfrow = c(2, 1))
> ts.plot(wr(wds2))
> ts.plot(test.data1 - wr(wds2))
```

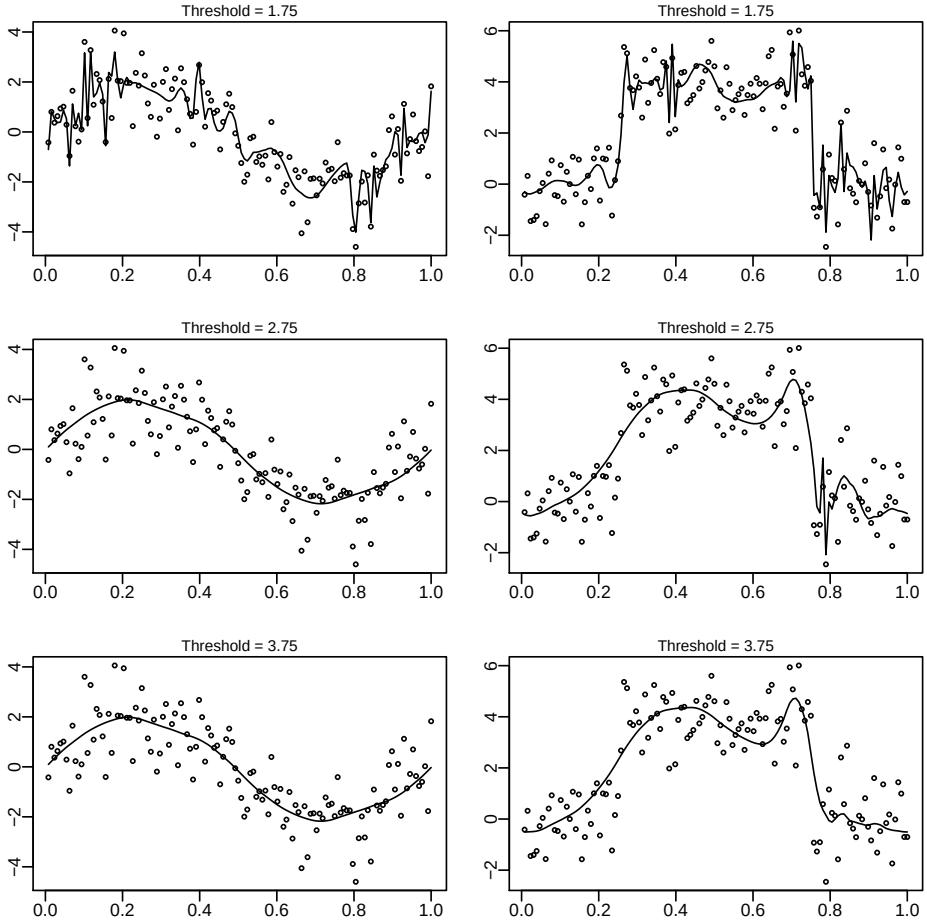




The next series of plots illustrates the use of denoising by means of selective wavelet reconstructions. Such procedures are nonlinear and are designed to distinguish coefficients that belong to the noise from true coefficients which contribute significant signal.

We prepare the graphic device, simulate some noisy data and display wavelet hard thresholding estimates with varying values of the threshold.

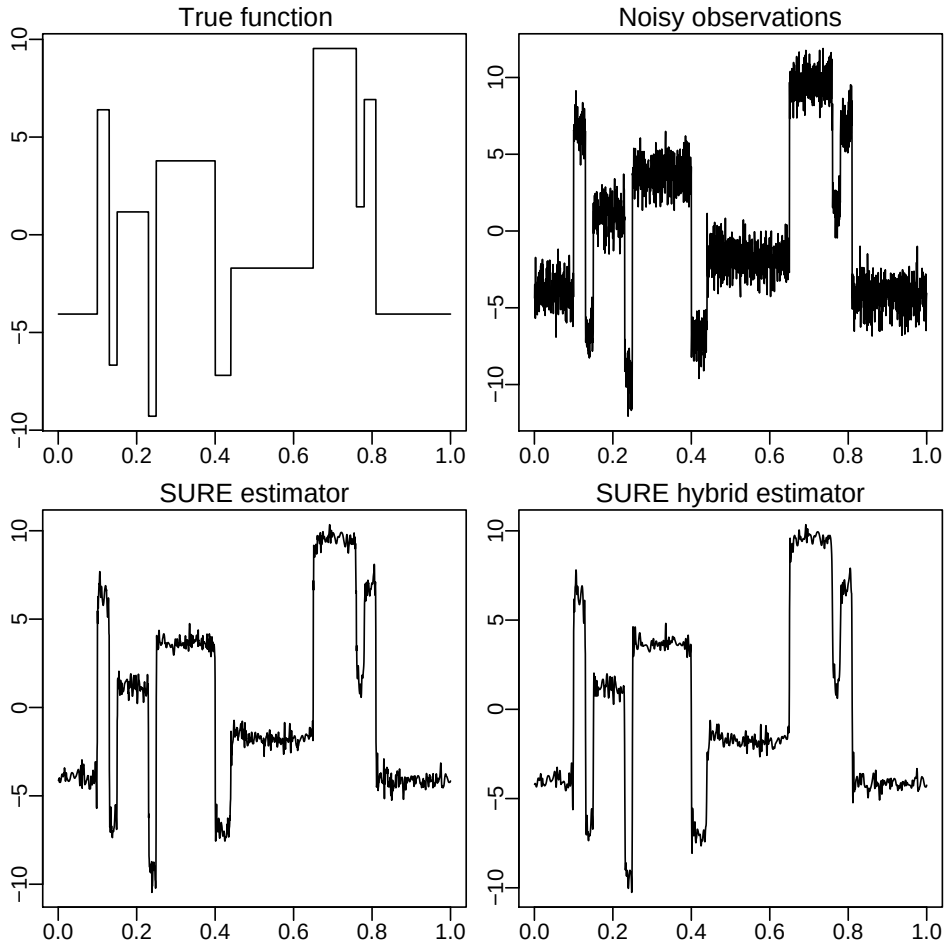
```
> par(mfcol = c(3, 2), mar = c(2, 2, 2, 1), mgp = c(5, 0.4, 0))
> x <- (1:128)/128
> f1 <- 2 * sin(2 * pi * x)
> f2 <- c(rep(0, 32), rep(4, 64), rep(0, 32))
> y1 <- f1 + rnorm(128)
> y1wd <- wd(y1, filter.number = 5)
> plot(x, y1, cex = 0.55)
> lines(x, wr(threshold(y1wd, policy = "manual", type = "hard",
+     levels = 0:6, value = 1.75)))
> mtext("Threshold = 1.75", side = 3, line = 0.1, cex = 0.6)
> plot(x, y1, cex = 0.55)
> lines(x, wr(threshold(y1wd, policy = "manual", type = "hard",
+     levels = 0:6, value = 2.75)))
> mtext("Threshold = 2.75", side = 3, line = 0.1, cex = 0.6)
> plot(x, y1, cex = 0.55)
> lines(x, wr(threshold(y1wd, policy = "manual", type = "hard",
+     levels = 0:6, value = 3.75)))
> mtext("Threshold = 3.75", side = 3, line = 0.1, cex = 0.6)
> y2 <- f2 + rnorm(128)
> y2wd <- wd(y2, filter.number = 5)
> plot(x, y2, cex = 0.55)
> lines(x, wr(threshold(y2wd, policy = "manual", type = "hard",
+     levels = 0:6, value = 1.75)))
> mtext("Threshold = 1.75", side = 3, line = 0.1, cex = 0.6)
> plot(x, y2, cex = 0.55)
> lines(x, wr(threshold(y2wd, policy = "manual", type = "hard",
+     levels = 0:6, value = 2.75)))
> mtext("Threshold = 2.75", side = 3, line = 0.1, cex = 0.6)
> plot(x, y2, cex = 0.55)
> lines(x, wr(threshold(y2wd, policy = "manual", type = "hard",
+     levels = 0:6, value = 3.75)))
> mtext("Threshold = 3.75", side = 3, line = 0.1, cex = 0.6)
```



Let's see another example using other types of thresholding procedures (SURE and hybrid SURE). The thresholds are determined by an optimization procedure and we therefore may wait some time to get the result.

```
> par(mfrow = c(2, 2), mar = c(1.5, 1.5, 1.5, 0.5), mgp = c(5,
+     0.4, 0))
> x <- (1:2048)/2048
> f <- fblocks(x)
> ssig <- sqrt(var(f))
> f <- ((f - mean(f)) * 5)/ssig
> fnoise <- f + rnorm(2048)
> fwd <- wd(fnoise, filter.number = 5)
> plot(x, f, type = "l")
> mtext(side = 3, line = 0.1, "True function")
> plot(x, fnoise, type = "l")
> mtext(side = 3, line = 0.1, "Noisy observations")
> fwdsure <- threshsure(fwd)
> plot(x, wr(fwdsure), type = "l")
```

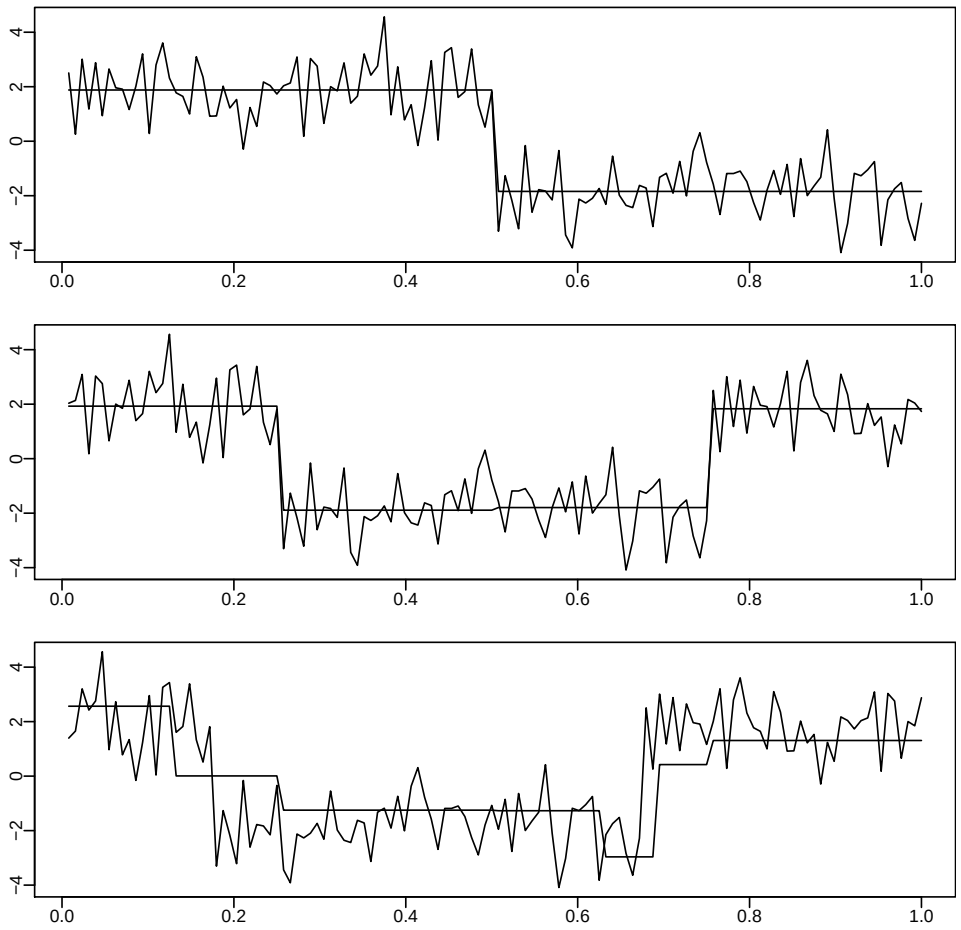
```
> mtext(side = 3, line = 0.1, "SURE estimator")
> fwdhyb <- threshhyb(fwd, lowlev = 2, seed = 0)
> plot(x, wr(fwdhyb), type = "l")
> mtext(side = 3, line = 0.1, "SURE hybrid estimator")
```



To end this lab we want to discuss of a practical issue that arises in using the discrete wavelet transform, namely its lack of translation invariance. To illustrate this point consider a simulated data set consisting of 64 $N(2,1)$ random variables followed by 64 $N(-2,1)$ observations. The other two data set in the plots that follow are just translated versions of the first (corresponding to a shift of $1/3$ and $1/4$ respectively). For all three data sets, universal thresholding smoothing has been applied. The first estimate is quite good, but the third estimate fails considerable, due do the lack of translation invariance.

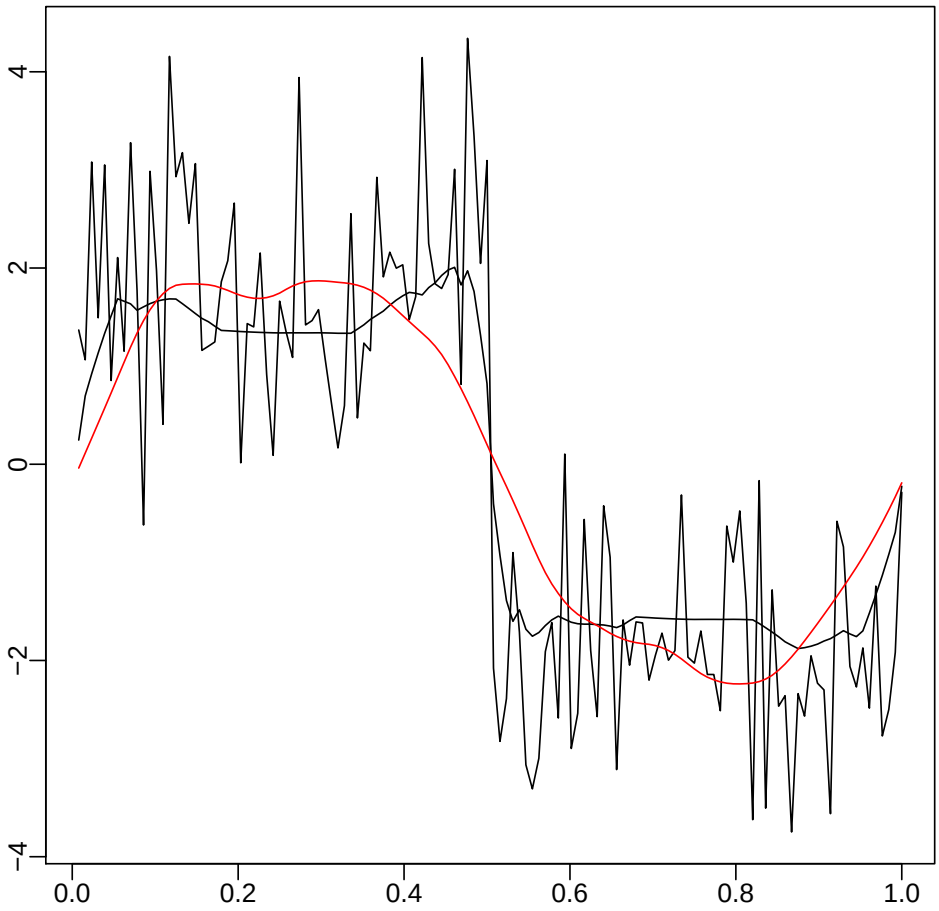
```
> k1 <- 32
> k2 <- 42
> par(mfrow = c(3, 1), mar = c(1.5, 1.5, 1.5, 0), mgp = c(5, 0.4,
+     0))
> x <- (1:128)/128
```

```
> unshft <- c(rep(1, 64), rep(-1, 64))
> noise <- rnorm(128)
> y <- 2 * unshft + noise
> plot(x, y, type = "l")
> yest <- wr(threshold(wd(y, filter.number = 1), levels = 0:6))
> lines(x, yest)
> if (k1 == 0) y1 <- y else y1 <- c(y[(1 + k1):128], y[1:k1])
> if (k1 == 0) shft1 <- unshft else shft2 <- c(unshft[(1 + k1):128],
+     unshft[1:k1])
> plot(x, y1, type = "l")
> y1est <- wr(threshold(wd(y1, filter.number = 1), levels = 0:6))
> lines(x, y1est)
> if (k2 == 0) y2 <- y else y2 <- c(y[(1 + k2):128], y[1:k2])
> if (k2 == 0) shft2 <- unshft else shft2 <- c(unshft[(1 + k2):128],
+     unshft[1:k2])
> plot(x, y2, type = "l")
> y2est <- wr(threshold(wd(y2, filter.number = 1), levels = 0:6))
> lines(x, y2est)
```



There is of course a solution to this problem. One could compute the wavelet estimator for all possible shifts, then inverse shift them and take as a final estimate the average of the estimates resulting from all shift values. This is done by the function `spincyclen` as illustrated below. The red smooth curve is the one obtained without spin cycling.

```
> par(mfrow = c(1, 1))
> par(mar = c(1.5, 1.5, 1.5, 0.5), mgp = c(5, 0.4, 0))
> x <- (1:128)/128
> y <- 2 * c(rep(1, 64), rep(-1, 64)) + rnorm(128)
> plot(x, y, type = "l")
> yest <- spincyclen(y, filter.number = 1)
> lines(x, yest)
> yest1 <- spincyclen(y, filter.number = 3)
> lines(x, yest1, col = "red")
```



One can easily repeat the illustration procedures given above with signals and data of its own choice.