Bachelor Thesis

Department of Statistics
Ludwig-Maximilians-Universität München

Wavelet Representations for functional data

Sven Martin Lorenz

Supervisor:
Dr. Fabian Scheipl
Date: 19. December 2019

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit selbständig verfasst und dabei keine anderen als die angegebenen Hilfsmittel benutzt habe. Sämtliche Stellen der Arbeit, die im Wortlaut oder dem Sinn nach Publikationen oder Vorträgen anderer Autoren entnommen sind, habe ich als solche kenntlich gemacht. Die Arbeit wurde bisher weder gesamt noch in Teilen einer anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Sven Martin Lorenz	Ort, Datum

Abstract

We introduce the function `tfb_wavelet()` for the `tidyfun`-package in R[1]. It uses Debauchies extremal phase wavelets to fit wavelets for functional data. Because we implement the function into an existing framework, which bases many of its functions on matrices, we also use wavelet matrices to estimate our coefficients and not the DWT. Further, we use these matrices to combat the boundary issues wavelets have by introducing a trend column. Additionally we solve the limitations of wavelets that they need an equispaced and dyadic sequence as their inputs by interpolating our input support to such an equispaced and dyadic sequence, next estimating the wavelet matrix and then linearly interpolating the matrix back to the original support. We conclude that it delivers similar results to `tfb_spline` and that some work can be done to optimise `tfb_wavelet` in regards to performance and finding correct default inputs.

[1] R Core Team (2019)

Contents

1 Introduction

In this work, we are trying to describe our approach and the outcome of implementing wavelets for functional data into the `tidyfun` package in R. A lot of our implementation is based on the methods in Wand and Ormerod (2011), wherein they are not using the normal discrete wavelet transform or pyramid algorithm, but are using wavelets in a semiparametric regression more akin to the rest of statistical literature. This means that we use wavelet matrices to compute our coefficients. We also use several adaptions for dealing with real world data, described in section 2.2.

1.1 Functional Data

We need two definitions from (Ferraty and Vieu 2006):

1. Definition: A random variable X is called functional variable (f.v.) if it takes values in an infinite dimensional space (or functional space). An observation x of X is called a functional data.

2. Definition: A functional dataset $x_1, ..., x_n$ is the observation of n functional variables $X_1, ..., X_n$ identically distributed as X.

Note that these definitions theoretically work for infinite dimensional space, but we are only using 2d data. So we have value and support pairs like this (y_i, x_i) with $i = 1, ..., N$, which build one observation x.

Another note the support does not need to be the same for each observation, we will call this case *irregular* functional data.

1.2 Wavelets

1.2.1 Haar wavelets

Since we are only using Debauchies extremal phase wavelets, we will only be describing them. Specifically we will explain Haar wavelets, the simplest of the Debauchies extremal phase wavelets. We also explain wavelets in the context of the discrete wavelet transform (DWT), a tree like structure to compute the coefficients. The explanation and the example is based on (Nason 2006, 15ff).

The DWT is only defined for an input sequence of dyadic ($2^J, J \in \mathbb{N}$) length and equal spacing. It essentially tries to analyse a given sequence y by evaluating the sequence at different levels by

adding and subtracting neighboured y_i. For the first level, we have following formulas:

$$d_k = y_{2k} - y_{2k-1} \text{ with } k = 1, ..., 2^{J-1} \tag{1}$$

$$c_k = y_{2k} + y_{2k-1} \text{ with } k = 1, ..., 2^{J-1} \tag{2}$$

As you can see the d_k are the subtraction coefficients and the c_k are the addition coefficients. Both d_k and c_k are also only half as long as the original sequence.

We continue this computation by replacing the y_i with the c_k for level = 2 and adding an additional subscript of difference $J - \text{level}$ to our coefficients. So for the second level we get:

$$d_{J-2,k} = c_{J-1,2k} - c_{J-1,2k-1} \text{ with } k = 1, ..., 2^{J-2} \tag{3}$$

$$c_{J-2,k} = c_{J-1,k} + c_{J-1,k} \text{ with } k = 1, ..., 2^{J-2} \tag{4}$$

These coefficients can be computed till $J - \text{level} = 0$, resulting in $c_{0,1} = \sum_{i=1}^{n} y_i$. In the next step, we introduce a so-called filter, which is different for every wavelet. This is needed because the energy of y is smaller than that of d (Nason 2006, 20f), with the definition of energy being: $\|y\|^2 = \Sigma_{i=1}^{n} y_i^2$. For Haar wavelets, this filter is relatively simple: $\alpha = \frac{1}{\sqrt{2}}$. We multiply this α to every coefficient at every level like this:

$$d_{j,k} = \alpha(c_{j+1,2k} - c_{j+1,2k-1}) \text{ with } j = J - \text{level} \tag{5}$$

$$c_{j,k} = \alpha(c_{j+1,2k} + c_{j+1,2k-1}) \text{ with } j = J - \text{level} \tag{6}$$

Since we add the Multiplication with α at every level, we can rewrite our formulas (example for second level).

$$d_{J-2,1} = \alpha(c_{J-1,2} - c_{J-1,1}) = \alpha(\alpha(y_4 + y_3) - \alpha(y_2 + y_1)) = \alpha^2(y_4 + y_3 - y_2 - y_1) \tag{7}$$

We can also graphically view this process as a tree. We use the sequence (1, 1, 7, 9, 2, 8, 8, 6), so $J = 3$:

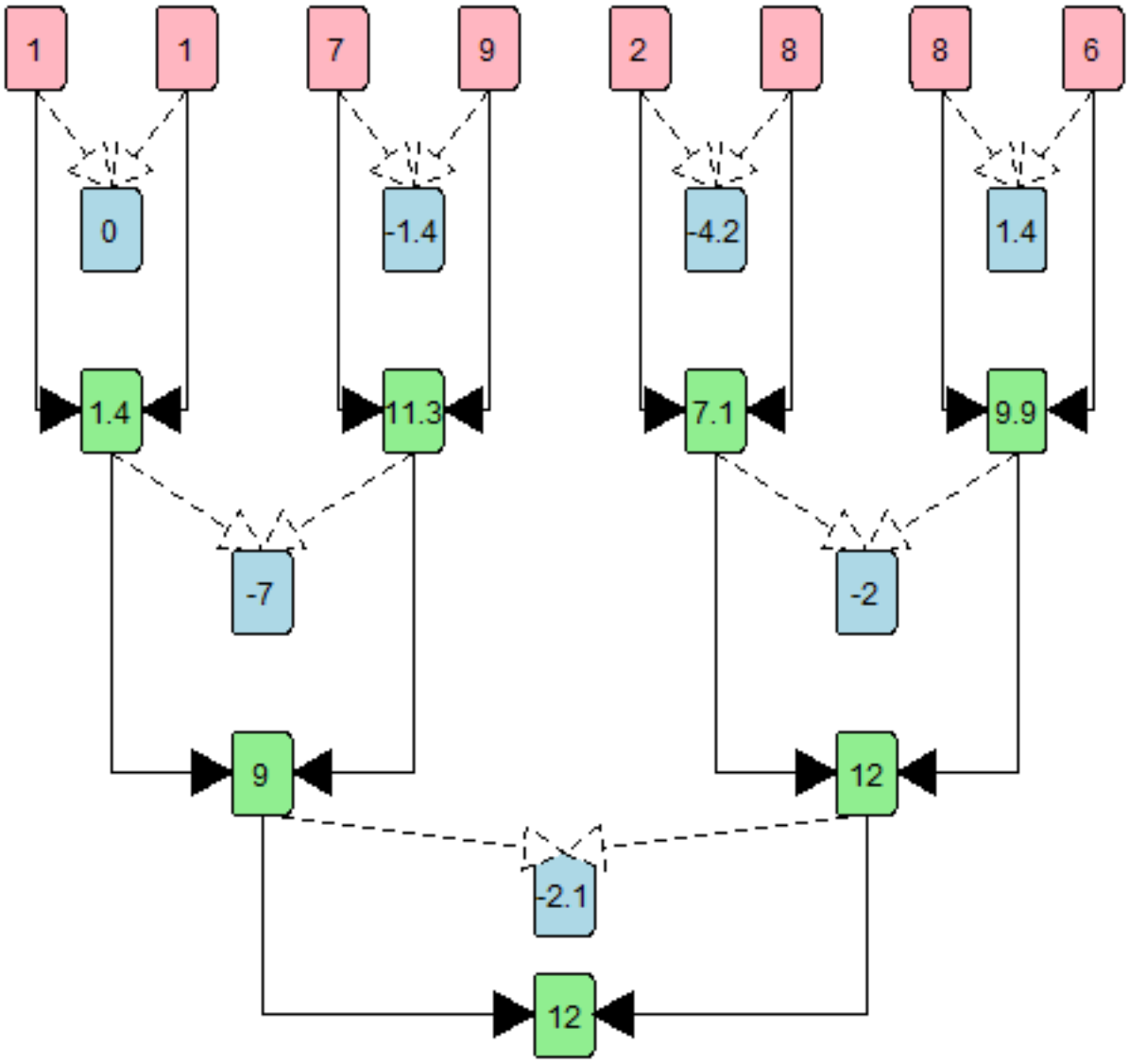

Figure 1: DWT example with (1, 1, 7, 9, 2, 8, 8, 6). Pink is y, green is c and blue is d. Dotted lines mean its subtractions and drawn through lines are addition.

The matrix notation for this process is as follows: $d = Zy$, but d includes as its first entry $c_{0,1}$ and the rest is d ordered from lowest to highest level.

For the above sequence, this matrix would look like this:

$$Z = \begin{bmatrix} \frac{1}{2^{\frac{3}{2}}} & \frac{1}{2^{\frac{3}{2}}} & \frac{1}{2^{\frac{3}{2}}} & \frac{1}{2^{\frac{3}{2}}} & \frac{1}{2^{\frac{3}{2}}} & \frac{1}{2^{\frac{3}{2}}} & \frac{1}{2^{\frac{3}{2}}} & \frac{1}{2^{\frac{3}{2}}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \\ \frac{1}{2^{\frac{3}{2}}} & \frac{1}{2^{\frac{3}{2}}} & \frac{1}{2^{\frac{3}{2}}} & \frac{1}{2^{\frac{3}{2}}} & -\frac{1}{2^{\frac{3}{2}}} & -\frac{1}{2^{\frac{3}{2}}} & -\frac{1}{2^{\frac{3}{2}}} & -\frac{1}{2^{\frac{3}{2}}} \end{bmatrix}$$

The first row computes $c_{0,1}$, rows 2-5 compute $d_{J-1,k}$ rows 6 and 7 $d_{J-2,k}$ and the last $d_{J-3,k}$. This can be obviously continued for longer sequences. Z is also orthogonal.

More interesting to us is Z^T, since with $d = Zy$ we compute a perfect fit, we need to get our d differently, more on that in 1.2.6. If we transpose Z every row can be thought of as an observation, which in the context of wavelets means a point of the input sequence. The columns then can be thought of as weights for different levels.

Lastly, we draw the common form of the Haar wavelet without the scaling of α:

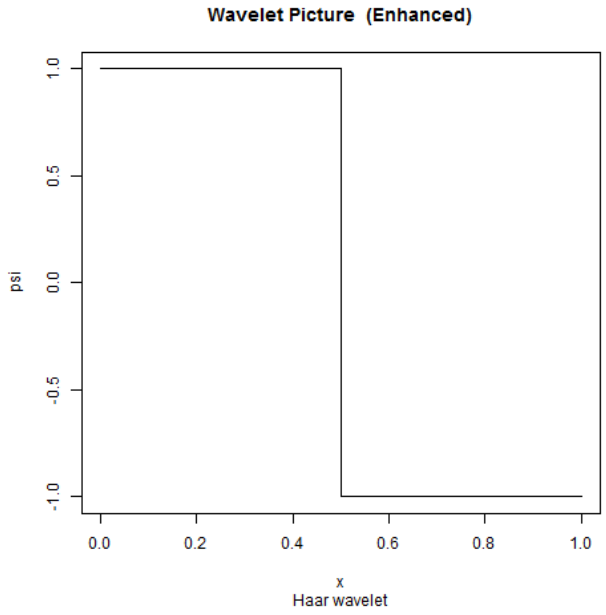

Figure 2: Haar wavelet filter

For a given wavelet coefficient, we know the y_i that factor into that computation. If we scale the support of y_i to $[0, 1]$ and overlay these onto this curve, we can know the value (without α) which we need to scale them with. This seems banal for Haar wavelets, but can be valuable for more complicated Debauchies extremal phase wavelets.

1.2.2 Debauchies extremal phase wavelets

As we mentioned the Haar wavelet from the previous chapter is a special case of the Debauchies extremal phase wavelet. Specifically the Haar wavelet has one vanishing moment. Meanwhile Debauchies extremal phase wavelets can have up to ten vanishing moments. This means that the filter and addition- and subtraction-structure will be a lot more complicated and with that the wavelet matrix. We show this with the example of three vanishing moments[2]:

The filter:

(0.33267055, 0.80689151, 0.45987750, -0.13501102, -0.08544127, 0.03522629)

The corresponding matrix:

$$Z = \begin{bmatrix} \frac{1}{2^{\frac{3}{2}}} & \frac{1}{2^{\frac{3}{2}}} & \frac{1}{2^{\frac{3}{2}}} & \frac{1}{2^{\frac{3}{2}}} & \frac{1}{2^{\frac{3}{2}}} & \frac{1}{2^{\frac{3}{2}}} & \frac{1}{2^{\frac{3}{2}}} & \frac{1}{2^{\frac{3}{2}}} \\ 0.807 & -0.333 & 0 & 0 & 0.035 & 0.085 & -0.135 & -0.460 \\ -0.135 & -0.460 & 0.807 & -0.333 & 0 & 0 & 0.035 & 0.085 \\ 0.035 & 0.085 & -0.135 & -0.460 & 0.807 & -0.333 & 0 & 0 \\ 0 & 0 & 0.035 & 0.085 & -0.135 & -0.460 & 0.807 & -0.333 \\ 0.080 & 0.737 & 0.344 & -0.329 & -0.231 & -0.046 & -0.194 & -0.362 \\ -0.231 & -0.046 & -0.194 & -0.362 & 0.080 & 0.737 & 0.344 & -0.329 \\ -0.381 & -0.023 & 0.220 & 0.553 & 0.381 & 0.023 & -0.220 & -0.553 \end{bmatrix}$$

And the filter drawn:

[2] Nason (2006)

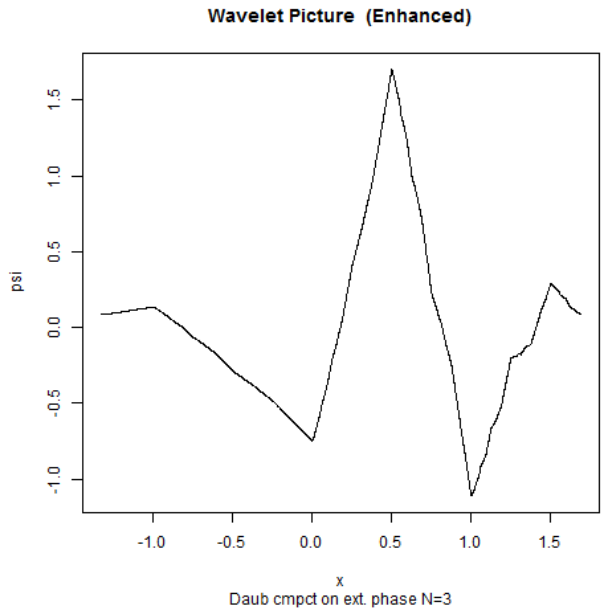

Figure 3: Filter for Debauchies Extremal Phase wavelets with three vanishing moments

The general formulas for these wavelets are in the appendix (4).

1.2.3 Inverse Transform

After having a set of parameters d and c, the inverse transform gets y back. Since this gets more and more complicated the higher the vanishing moments and in section 1.2.5 we use Haar wavelets as an example, only the inverse transform for Haar wavelets are shown:

$$c_{j,2k} = 2^{-1/2}(c_{j-1,k} - d_{j-1,k})$$
$$c_{j,2k-1} = 2^{-1/2}(c_{j-1,k} + d_{j-1,k}) \tag{8}$$

The general formula for this is in the appendix(4).

1.2.4 Advantages of wavelets in our context

Wavelets have several advantages. First, one can derive the structure of a sequence by looking at the values of d. $d_{j,k}$ gets larger if there is a large discrepancy between neighboured points. From this we only need to look at d to know the oscillations of our sequence.

Second, these oscillations are a locale phenomenon in our d, since $d_{j,k}$ are only influenced by a

subset of points. So even by one oscillation on a low level, only the $d_{j,k}$ down the line are impacted.

Thirdly, we have relatively sparse coefficient vectors as we only evaluate to a certain level (more on that in the next chapter) and we also use lasso regression, which pushes coefficients to zero if so wished.

We can also analyse our function on different scales. Meaning that changing the level to which we evaluate results in getting different estimated functions. This can be useful to extract either big changes in our data or very tiny changes and can be adjusted relatively easily[3].

1.2.5 Wavelet matrix creation

The normal way to compute the wavelet matrix is `wavethresh::GenW()`, that computes the whole wavelet matrix, but this is very inefficient for our case, because we only want the matrix up to a certain level. Therefore, for large n computing the matrix like that takes a lot of time and then takes up a lot of space. We use the alternative function `ZDaub()` which only computes up to the tenth level and thus is a lot faster[4].

Input : x: support with dyadic length and equal spacing

level: level to which to evaluate

filter_number: filter number or vanishing moments

resolution: a dyadic integer

Output: wavelet matrix evaluated to level for filter_number

for $i = 1, ..., 2^{level}$ **do**

1. Create wavelet with $y_i = 0 \; \forall i = 1, ..., \text{resolution}$, so all d and c are zero
2. Set $d_{i-1,1} = 1$
3. Calculate Inverse Transform y_{inv}
4. Scale y^{inv} with $\sqrt{\text{resolution}}$ to get (y_i^{inv}, x_i^{inv}) with $x_i^{inv} \in [1, \text{resolution}]$
5. Scale x_i to $[1, \text{resolution}]$, then use linear interpolation to get the correct values

end

Algorithm 1: Fast Algorithm for creating a wavelet matrix

We use the Haar wavelets as an example again. Set $d_{1,1} = 1$ and resolution $= 8$. The resolution

[3] Nason (2006)

[4] Wand and Ormerod (2011)

should definitely be higher in a user setting, but would make things way more complicated for our example. Since we have $d_{1,1} = 1$ we get a second level column.

The 1. Step:

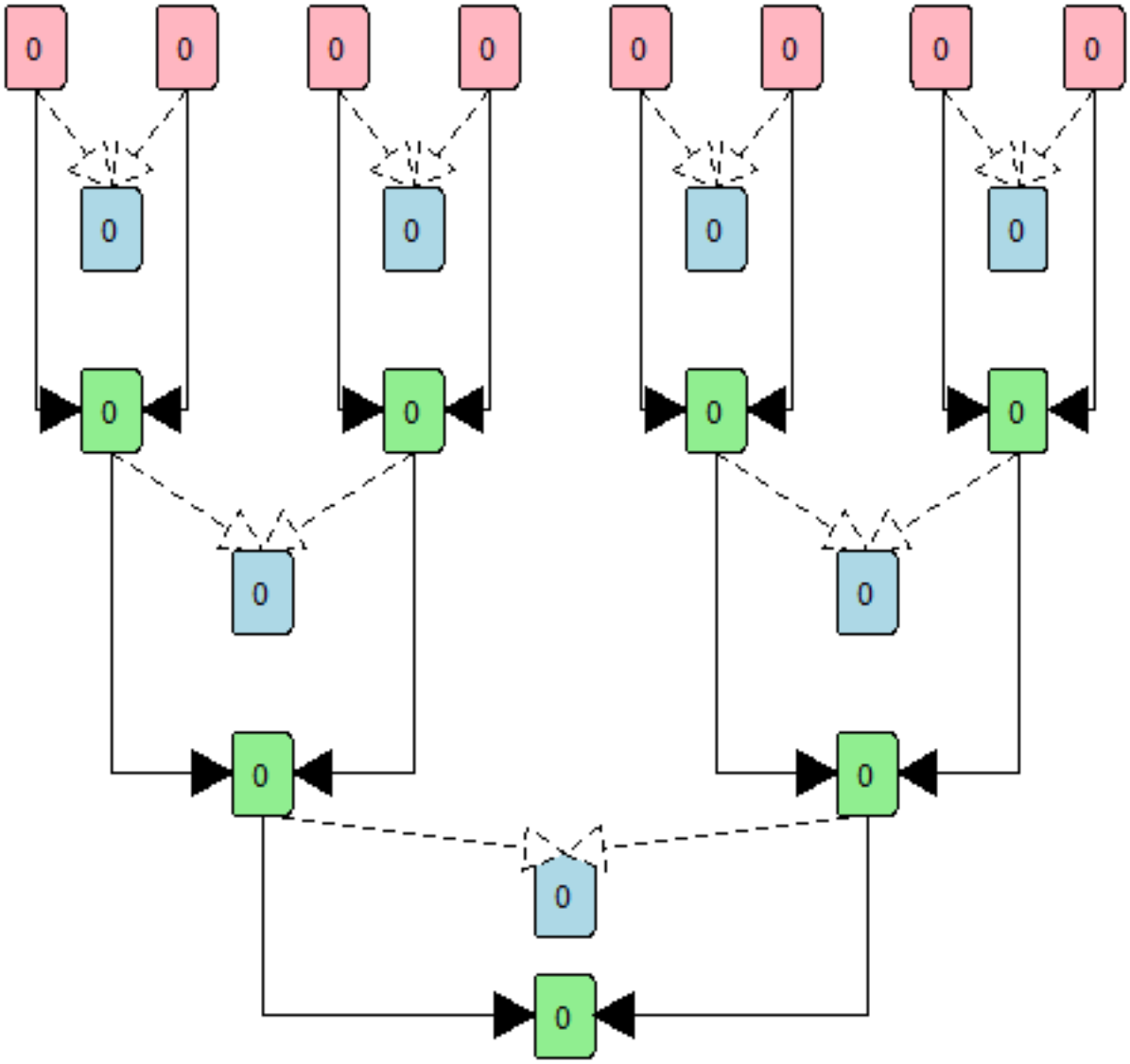

Figure 4: DWT with everything set to zero

The 2. Step: Now, we set $d_{1,1} = 1$:

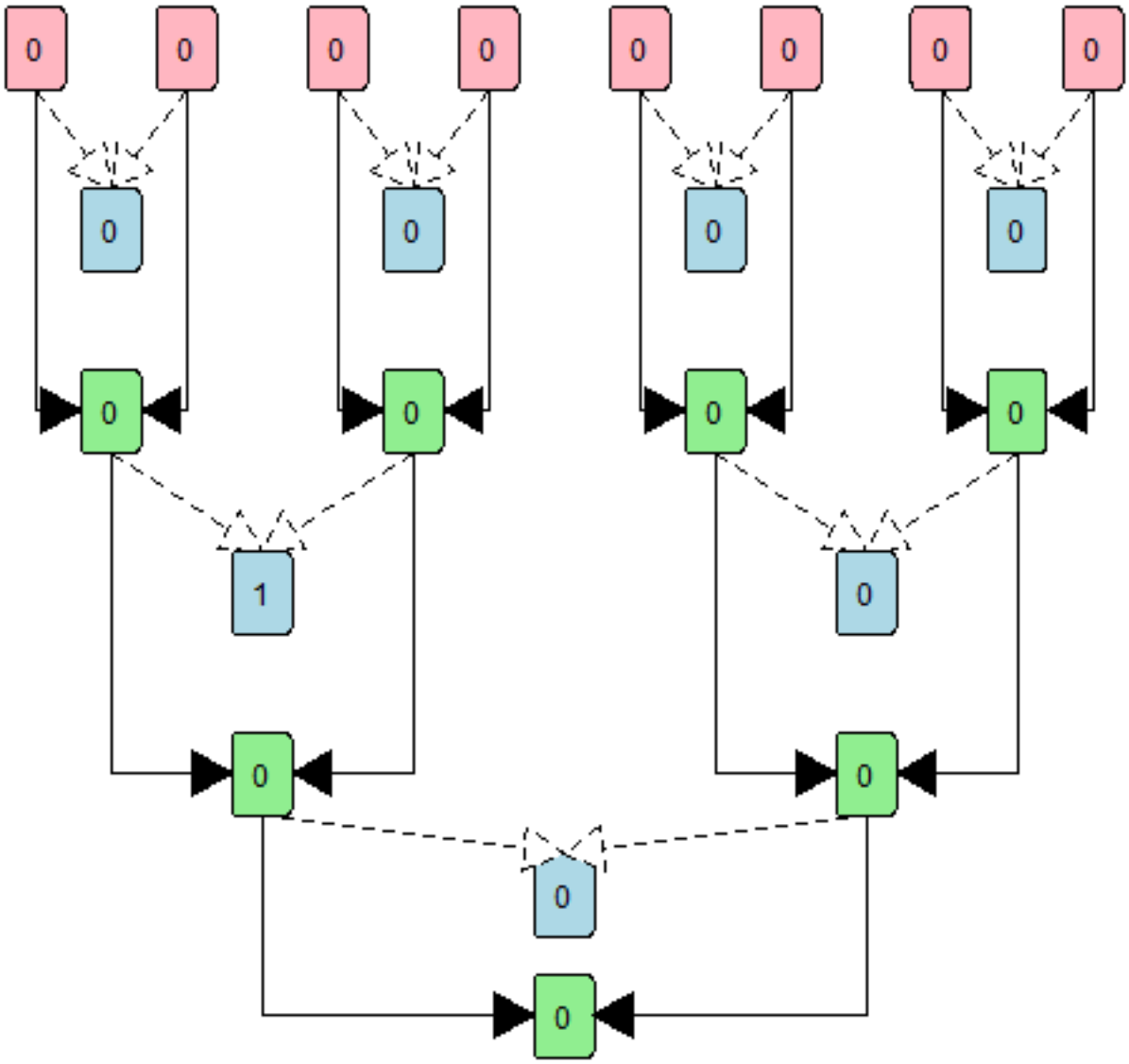

Figure 5: Set one d to one.

The 3. Step is computing the inverse with formula 8.

$$c_{2,1} = 2^{-1/2}(c_{1,1} + d_{1,1}) = 2^{-1/2}(0 + 1) = 2^{-1/2}$$

$$c_{2,2} = 2^{-1/2}(c_{1,1} - d_{1,1}) = 2^{-1/2}(0 - 1) = -2^{-1/2}$$

$$y_1 = c_{3,1} = 2^{-1/2}(c_{2,1} + d_{2,1}) = 2^{-1/2}(2^{-1/2} + 0) = 1/2$$

$$y_2 = c_{3,2} = 2^{-1/2}(c_{2,1} + d_{2,1}) = 2^{-1/2}(2^{-1/2} + 0) = 1/2$$

$$y_3 = c_{3,3} = 2^{-1/2}(c_{2,2} + d_{2,1}) = 2^{-1/2}(-2^{-1/2} + 0) = -1/2$$

$$y_4 = c_{3,4} = 2^{-1/2}(c_{2,2} + d_{2,1}) = 2^{-1/2}(-2^{-1/2} + 0) = -1/2$$

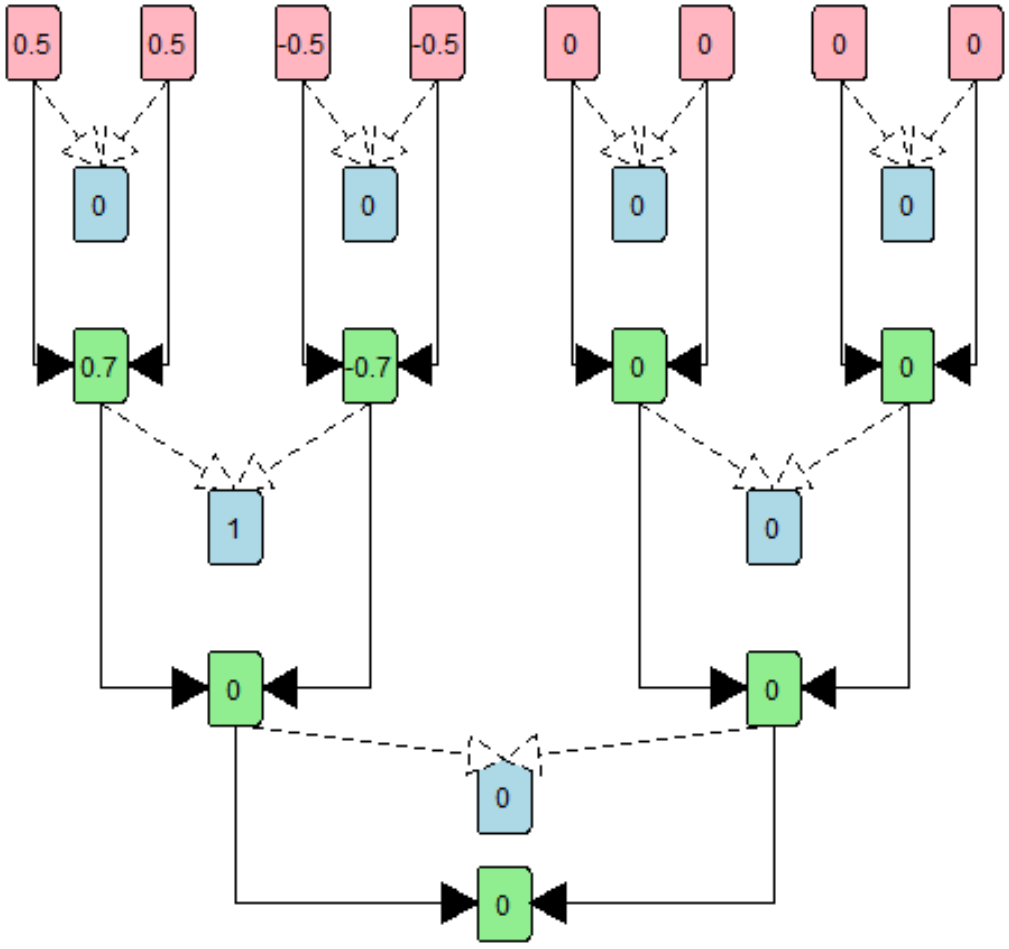

Figure 6: Compute the inverse transform

The 4. Step is scaling it with $\sqrt{\text{resolution}}$ and plotting the second level Haar wavelet. Note that we do not include scaling with α in this!

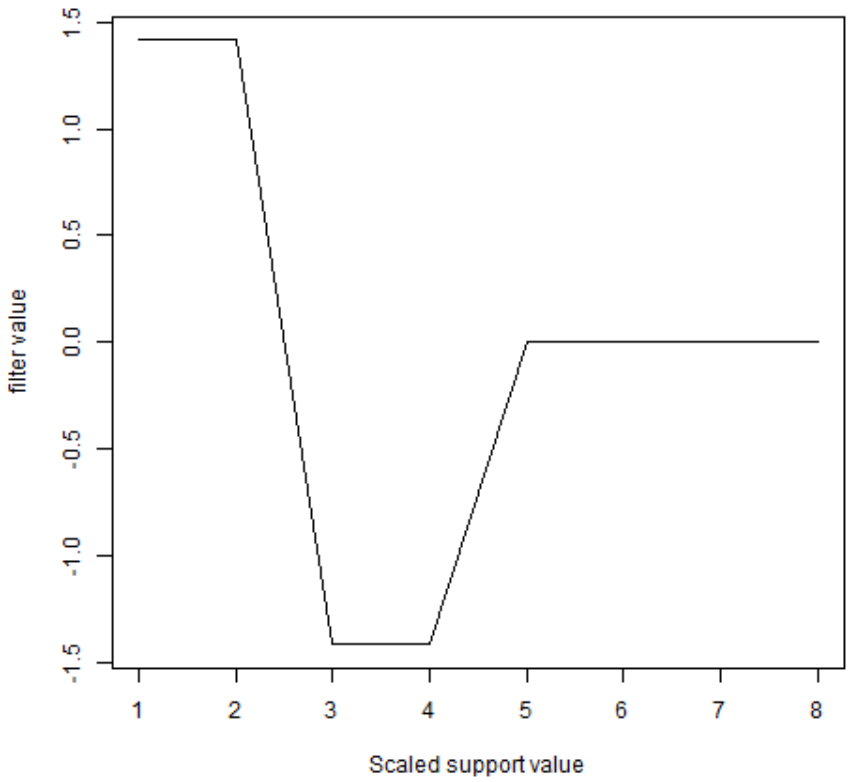

Figure 7: The resulting second level column plotted

Out of brevity we exclude the 5. Step.

We can do this for several low level j to get more wavelet matrix columns and with that we can fill our wavelet matrix. This plot also shows us why we need a high resolution, since we would get the wrong values if our scaled support would be between two and three. A higher resolution makes this less likely.

The last parameter not yet mentioned is level. Note that this level parameter is different to the one in the DWT, as it counts (in the context of the tree) from bottom to top ($j = \text{level}$) and the DWT from top to bottom ($j = J - \text{level}$. The parameter controls how big our matrix will be and which level should be included. Normal wavelet matrices are $Z \in \mathbb{R}^{nxn}$, but the highest levels are normally not necessary to get a good estimate and it takes a lot of time to compute more than up to ten levels. Ten levels are already bigger than it seems, because $2^{10} = 1024$ columns will be computed. Included will be every level from $j = 0$ to $j = \text{level}$.

1.2.6 Wavelet Regression

Instead of using the fast discrete wavelet transform, we are defining our wavelet fitting as a regression problem. For that, we get this common form:

$$y = \beta_0 + \sum_{i=1}^{K} d_i z_i + \epsilon_i \tag{9}$$

with β_0 as intercept. d_i are the wavelet coefficients, but not computed through the DWT, but rather through some regression method. $z_i(x)$ are the rows of the wavelet matrix. ϵ_i are the error terms.

1.2.6.1 Least squares regression

We now, for simplicity, add an intercept column to our wavelet matrix and correspondingly add β_0 to our d. We can then write our formula from before in matrix notation:

$$y = Zd + \epsilon \tag{10}$$

To fit we first compute the QR decomposition ("9. The Qr Decomposition," n.d.) and then use the Newton-Raphson method to get our coefficients (Gil, Segura, and Temme 2007).

This is the quick and possibly dirty approach in our function. Since we do not use the DWT and lasso regression with cross-validation is relatively time expensive, least squares regression fills that niche. It performs relatively well, but does not produce any zeroes in the coefficient vector, so every level is factored in no matter how irrelevant.

1.2.6.2 Lasso regression

We have the same base formula as before, but we now use a different estimation process:

$$\hat{d}^{Lasso} = \underset{d}{\operatorname{argmin}} \frac{1}{2} \sum_{i=1}^{n} (y_i - d_0 - \sum_{j=1}^{\text{level}} x_{i,j} d_j)^2 + \lambda \sum_{j=1}^{\text{level}} |d_j| \tag{11}$$

The best λ is estimated through cross-validation[5], while every model is fitted via penalized maximum likelihood[6].

[5] Hastie, Tibshirani, and Friedman (2001)
[6] Friedman, Hastie, and Tibshirani (2009)

2 Implemented Function

2.1 Documentation

The new function `tfb_wavelet()` which converts functional data in different formats to a `tfb`-object including but not limited to the wavelet matrix and the fitted coefficients for every curve. Important parameters are wavelet matrix controlling parameters `level` and `filter_number`. `penalized=TRUE` means we use lasso regression otherwise we use least squares regression. `...` controls the parameters for the lasso regression in `glmnet::cv.glmnet` and `glmnet::glmnet`. For more information use `?tfb_wavelet` in R.

2.2 Theory

We are using several adjustments to deal with real world data and to circumvent the restrictions wavelets have. These adjustments are also made in part to more fluently incorporate `tfb_wavelet()` into existing code of the `tidyfun` package.

2.2.1 Why matrix representation of wavelets?

The `tidyfun` package relies heavily on the `tf_evaluate(x, arg)` function for its `tfb`-class. The function evaluates a `tfb`-objects at certain points defined by `arg`. For that it uses the design matrix and the coefficients saved in the `tfb`-object and linearly interpolates the design matrix if necessary. So right from the get-go we needed to compute the wavelet matrix, otherwise we would need to reimplement `tf_evaluate()`, which would have been very inefficient.

The second reason is simplicity for our target user base. Least squares and lasso regression are well known for statisticians and the `glmnet` package is widely used, so for most statisticians the only new things are using the `level` and `filter_number` parameters. This goes hand in hand with tidyfun's goal "to provide accessible and well-documented software that makes functional data analysis in R easy"[7].

Next there were several problems with the preferred wavelet package `wavethresh` that implements the DWT. The `level` parameter for `ZDaub()` was clashing with a similar but not equal parameter in `threshold.wd(levels)`. This parameter also needs to either be handpicked through experience or

[7] Scheipl and Goldsmith (2019)

cross-validated. This cross-validation is also not implemented in `wavethresh`, making this package rather incovenient.

The last point has to do with the next chapter where we solve several problems, which would not have been possible without the wavelet matrix.

2.2.2 Non wavelet conforming supports

Wavelets as we know only handle data that is of dyadic length and equal spacing, because real life data almost never conforms to this standard one needs to clean up their supports. Therefore, we would need to incorporate a bunch of different cases, which makes code complicated. A different solution includes our wavelet matrix and is relativly easy to implement.

The problems we are solving:

1. The supports between the curves are different.
2. The support is not of length 2^J and regularly spaced as required by wavelets.

The same algorithm solves both of these:

Input : A functional dataset

Output: A wavelet matrix

begin

1. Get all unique xvalues of input dataset
2. Linearly interpolate between $\min(x)$ and $\max(x)$ so that $n = 2^J$, get x_{interp}
3. Estimate wavelet matrix with x_{interp} and algorithm 1
4. Linearly interpolate between columns of wavelet matrix back to the original xvalues

end

Algorithm 2: Wavelet correction

The wavelet matrix, if we remember section 1.2, has the points as observations in the rows. Since we need points on a different support set, we need to interpolate between these observations. Therefore, we take each column and transform them onto the original grid.

2.2.3 Symmetry constraints

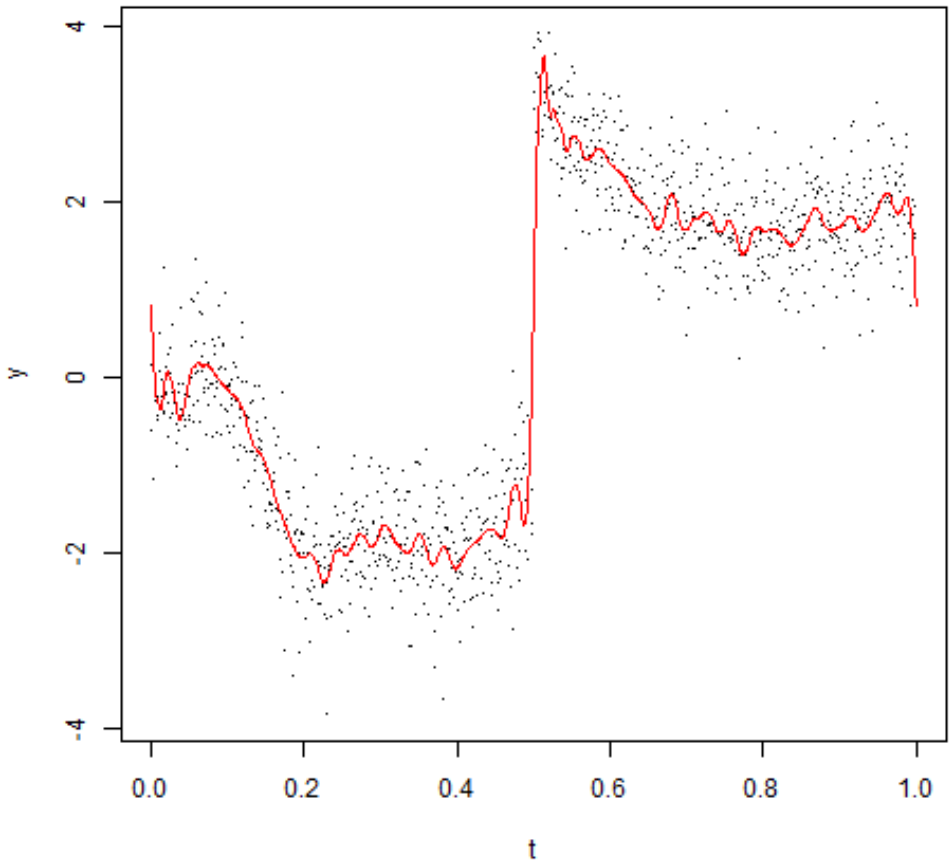

Figure 8: Boundary artifacts from wavelets

Wavelets have symmetry constraints as shown by figure 8, the end points always try to be on the same height. This is normally solved by extending the sequence with some method on each side. Since we do not use such extension methods, we need a different solution.

What we came up with was introducing a trend variable. So $Z_{complete} = (1 \text{ support } Z_{wavelet})$. With that we can compute the wavelet fit from figure 8 again:

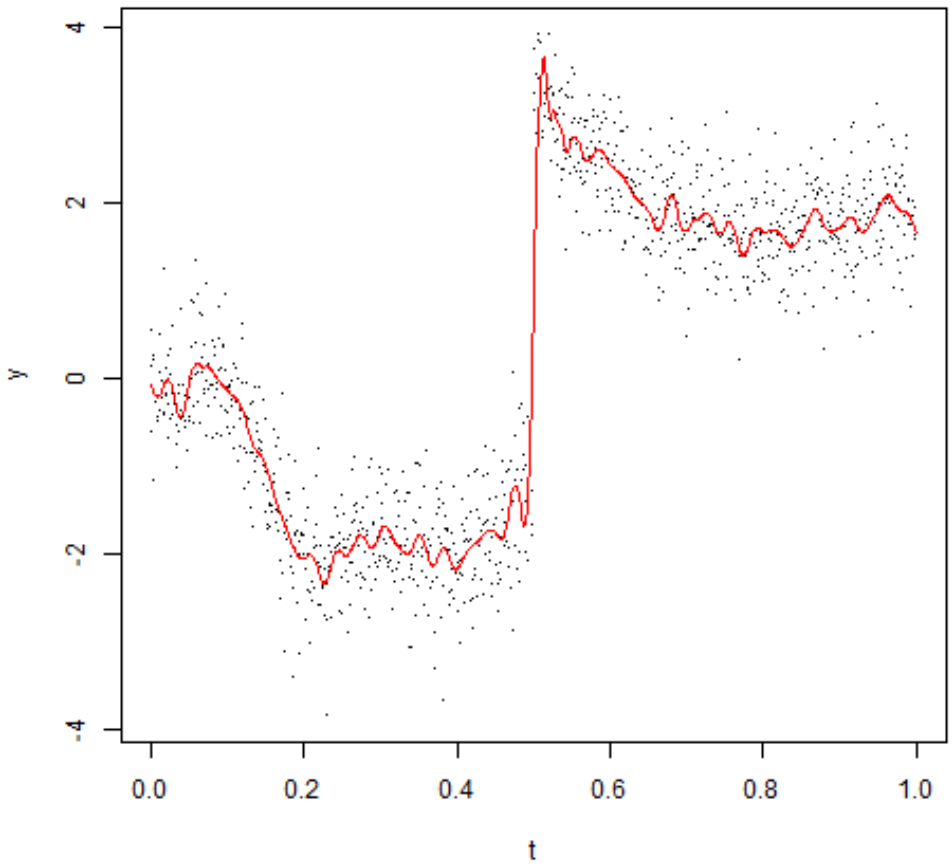

Figure 9: Boundary artifacts from wavelets remedied through trend column

And thus we see that the artefacts at the end have almost completely disappeared.

2.3 Implementation

In this section we will go over some examples for `tfb_wavelet()`.

2.3.1 Dataset

Our dataset describes activity data from a study of congestive heart failure. And since using all the observations is a little bit confusing, we aggregate for the `gender` column. So in the end we have two curves for which we are going to fit different wavelets. One for `Female` and one for `Male`.

2.3.2 Different Parameters

First let us use the least squares fit, evaluated to the second level and the Haar wavelet:

```
wavelet <- tfb_wavelet(activity$mean_act,
                       level = 2,
                       filter_number = 1, #Haar wavelet
                       penalized = FALSE)
```

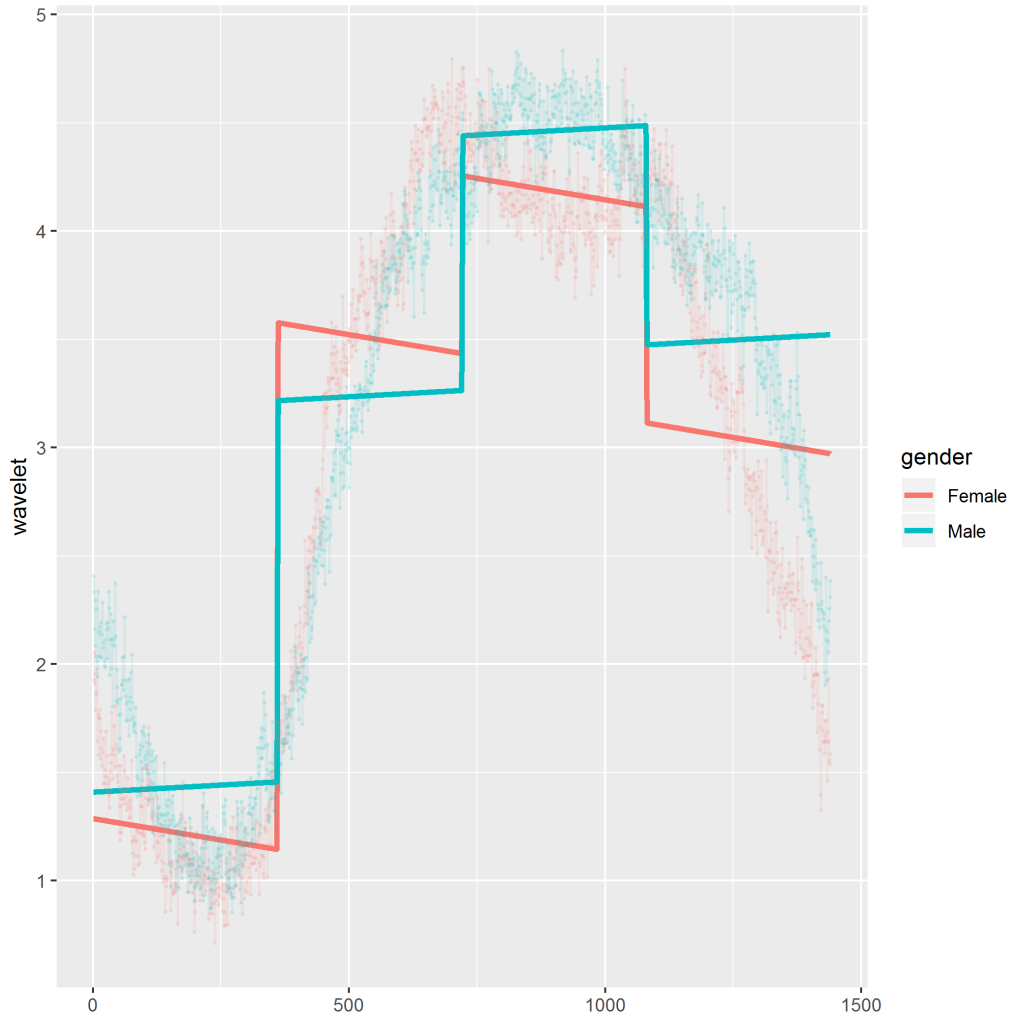

Figure 10: tfb_wavelet() for a level two Haar wavelet

The fitted curves have the typical Haar wavelet form, which is a stepwise function. Since we use a trend these stepwise functions also have a slope.

We can also get more steps by making Z bigger:

```
wavelet <- tfb_wavelet(activity$mean_act,
                       level = 5,
                       filter_number = 1,
                       penalized = FALSE)
```

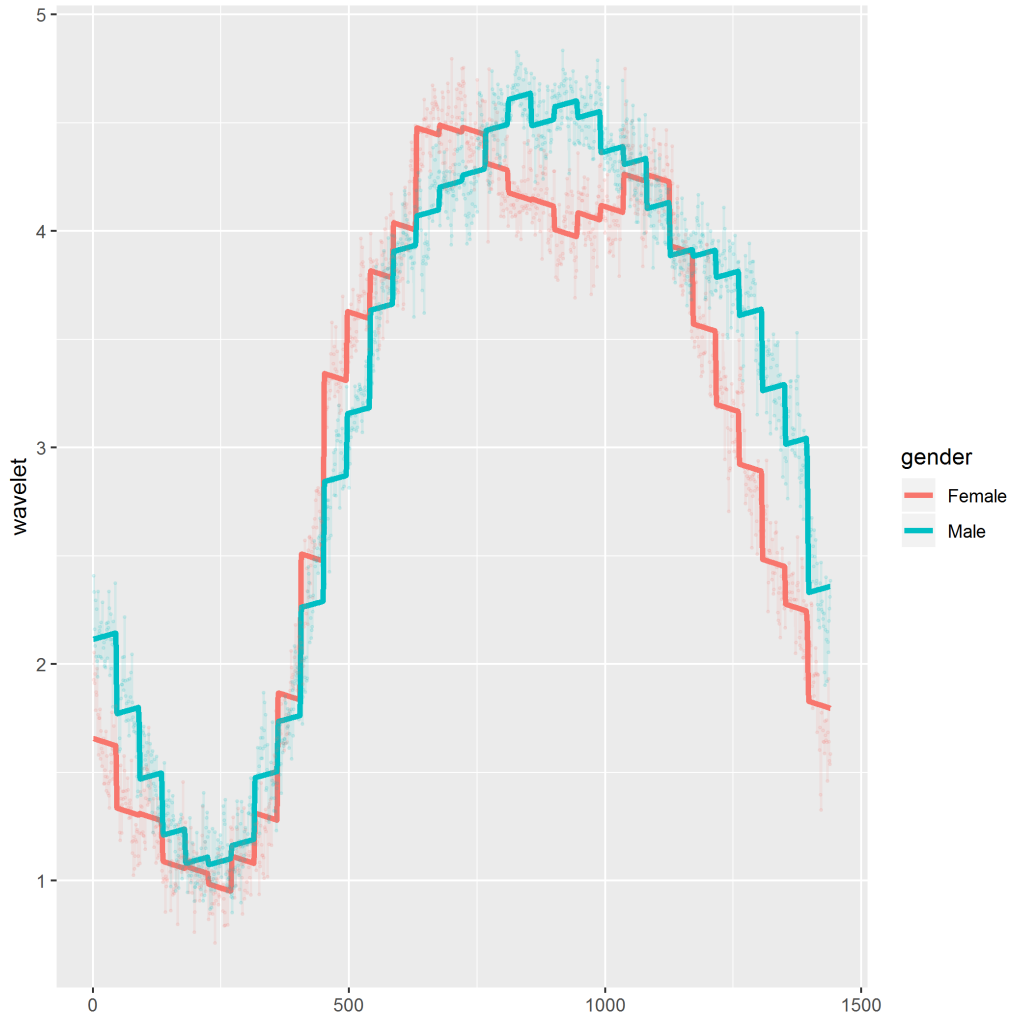

Figure 11: tfb_wavelet() for a level five Haar wavelet

On the other end of the spectrum of smoothness we set both `level` and `filter_number` to ten and watch our fit get squiggly.

```
wavelet <- tfb_wavelet(activity$mean_act,
                       level = 10,
                       filter_number = 10,
                       penalized = FALSE)
```

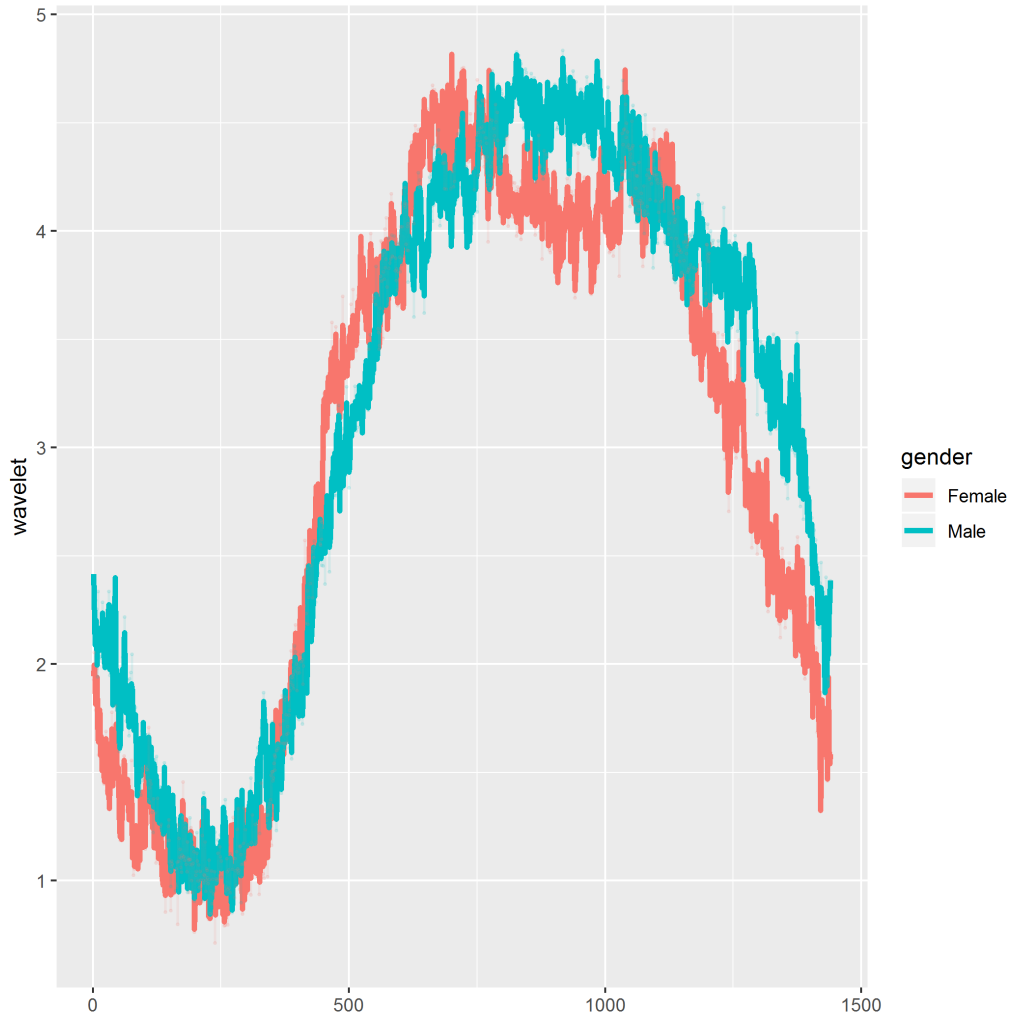

Figure 12: tfb_wavelet() with maximum squigglyness

We overfit our data by a lot. Therefore, this is not too useful here.

These were all least squares fits. So what about Lasso regression. Here we use `cv.glmnet` from the `glmnet` package. Since `cv.glmnet` also interfaces `glmnet` we have a bunch of parameters to choose from. Let us try Lasso regression for the Haar wavelet and `level = 2`.

```
activity_wavelet <- tfb_wavelet(activity$mean_act,
                        level = 2,
                        filter_number = 1,
                        penalized = TRUE # Lasso
)
```

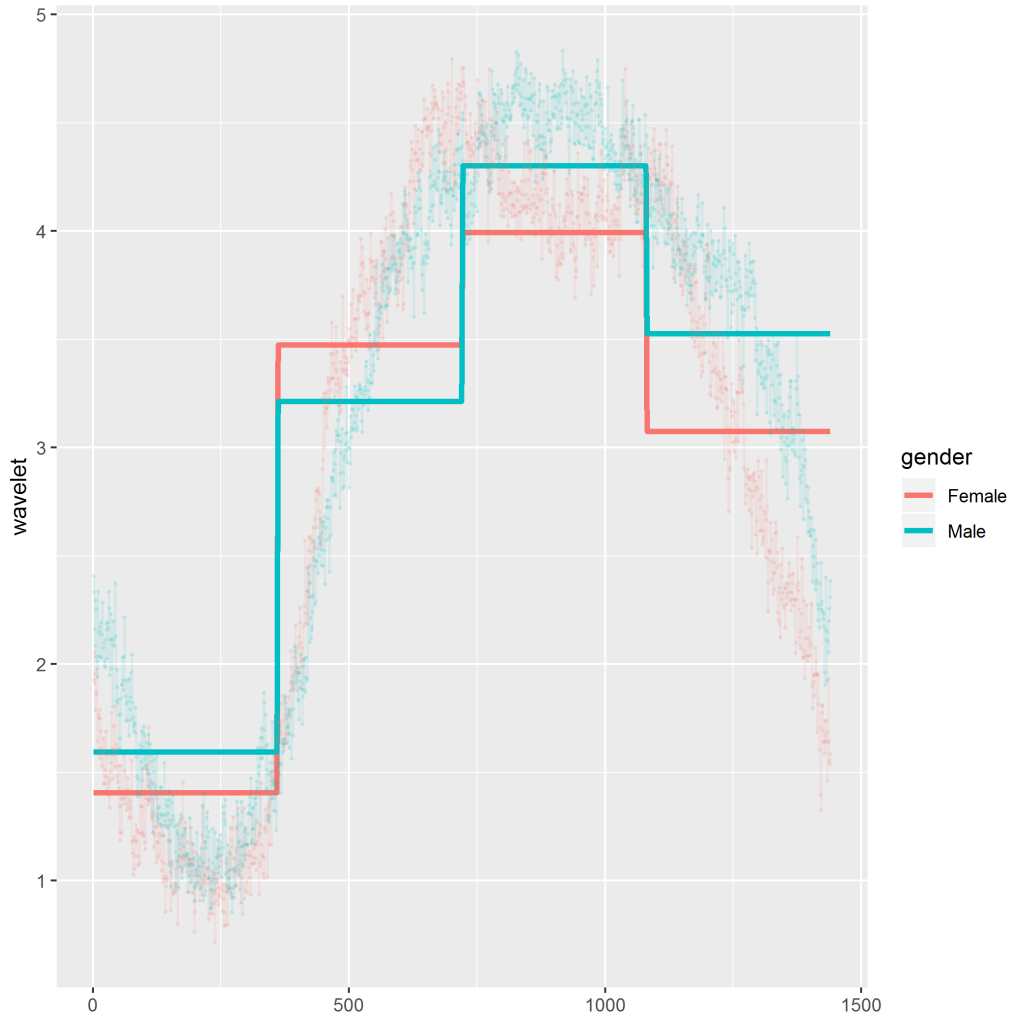

Figure 13: tfb_wavelet() for a level two Haar wavelet with lasso

Apparently the trend coefficients did not make the cut.

```
activity_wavelet[[1]][2] # = 0, for Males

activity_wavelet[[2]][2] # = 0, for Females
```

Indeed the second parameter, which is the trend parameter, is zero for both women and men.

Next we do leave-one-out cross-validation (LOOCV) by setting `nfolds = 1440`:

```
wavelet <- tfb_wavelet(activity$mean_act,
                                level = 3,
                                filter_number = 5,
                                penalized = TRUE, # Lasso
                                nfolds = 1440)
```

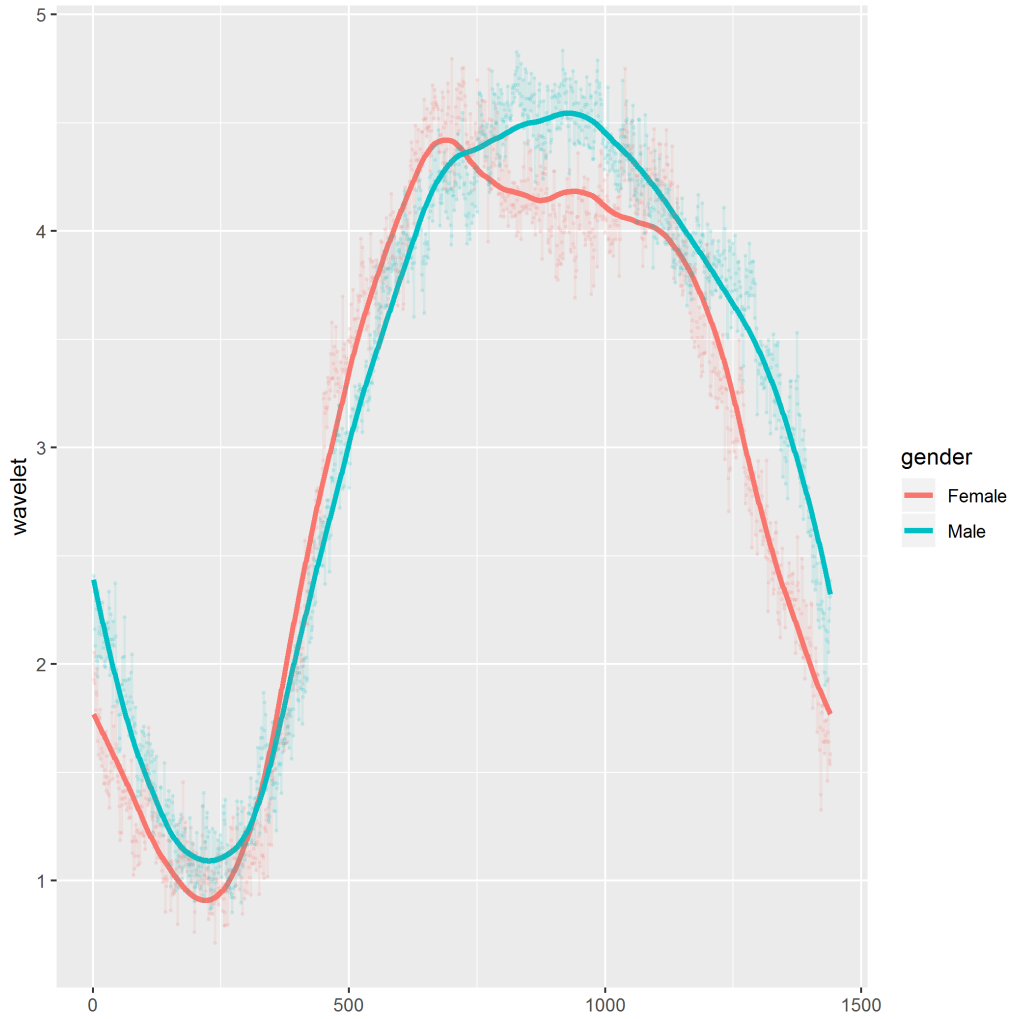

Figure 14: tfb_wavelet() for a level=3 filter_number=5 with lasso with leave-one-out cross-validation

Lastly we examine the output of `tfb_wavelet()` a bit closer. The output is a list with the coefficients as vectors. Everything else is written in the attributes of the object. The attributes:

name	description
domain	The domain for the support
basis_args	Wavelet matrix coefficients
basis_label	Short text for print()
basis	Function to interpolate wavelet matrix
basis_matrix	Wavelet matrix
resolution	Tolerance parameter for the support
arg	Unique support values

2.3.3 Comparison to `tfb_spline()`

For simplicity we aggregate the whole dataset now and compute both a spline and a wavelet. Both matrices used to fit are the same size.

```
wavelet <- tfb_wavelet(activity_sum$mean_act, level = 3)


spline <- tfb_spline(activity_sum$mean_act, k = 9)
```

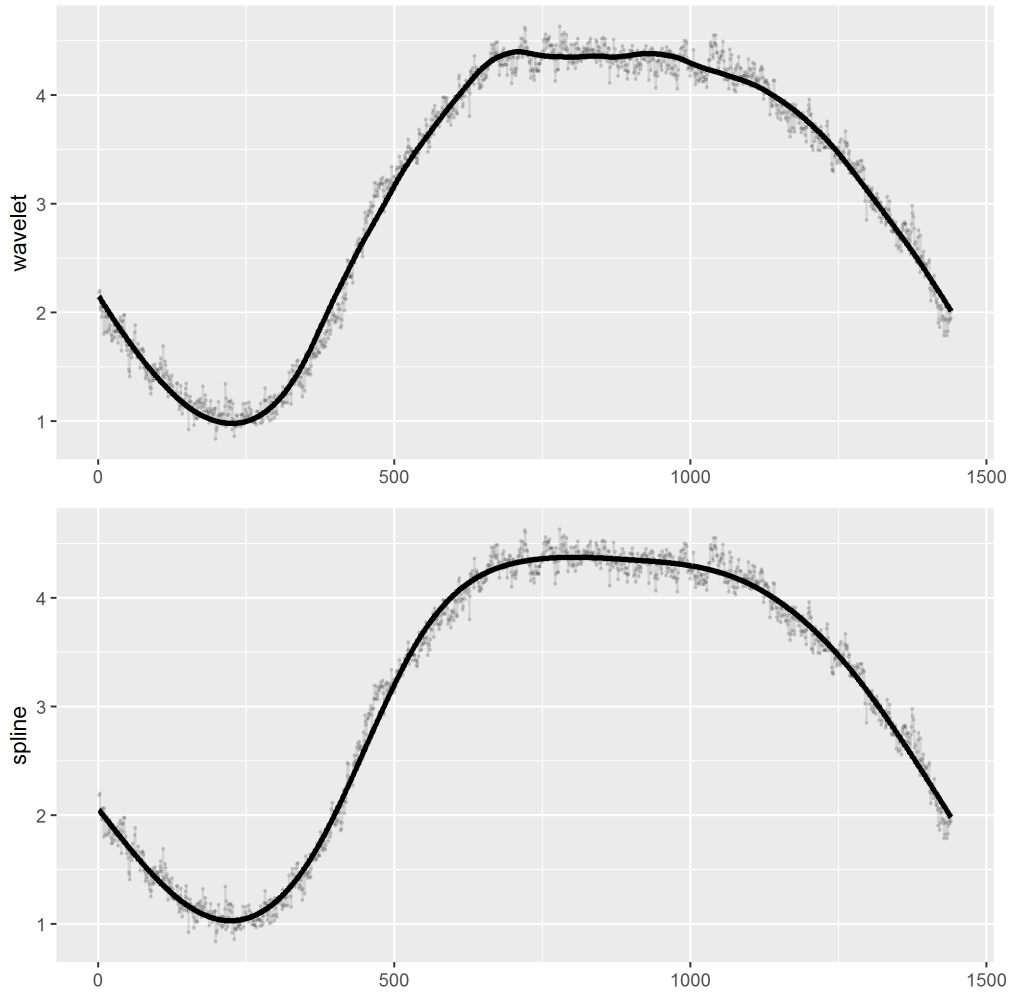

Figure 15: Comparison between tfb_spline() and tfb_wavelet()

We can see that the fits are pretty similiar. Let us compare the time it takes to fit both these curves:

name	min	median	mem_alloc
wavelet	26.5ms	27.9ms	15.23MB
spline	12.9ms	14.6ms	1.51MB

However, this does not give `tfb_wavelet` full credit, since we need to QR decompose our wavelet matrix for least squares we have an overhead and for one curve that is noticeable. So let us try it on all the curves of the original dataset.

name	min	median	mem_alloc
wavelet	268ms	269ms	197MB
spline	847ms	847ms	257MB

Here `tfb_wavelet` is a bit faster than `tfb_spline` for these 329 curves.

2.3.4 Constraints

The time from the benchmarks is relatively small, but how does `tfb_wavelet()` scale? Generally, the biggest influence on time and memory allocation is the `level` parameter since it controls the size of the wavelet matrix. Also the `penalized` parameter is an increase in time and space, depending on how big `level` is.

For a dataset of 100 curves with either a length of 256 or 32768, we get the following time and memory allocation:

name	level	lasso	min	median	mem_alloc
wavelet_256	2	FALSE	21.04ms	24.11ms	12.37MB
wavelet_32768	2	FALSE	3.96s	3.96s	877.56MB
wavelet_32768	6	FALSE	3.33s	3.33s	2.93GB
spline_32768	2	FALSE	5.91s	5.91s	2.63GB
wavelet_256	2	TRUE	6.76s	6.76s	407MB
wavelet_32768	2	TRUE	35.82s	35.82s	39.76GB
wavelet_32768	6	TRUE	2.44m	2.44m	79.67GB

Therefore, for large datasets either trying to have a low `level` and/or no lasso regression is a

must for a quick computation.

Next, we encountered a problem with extreme outliers. In this case the domain for all but one data point is $x \in [0, 1]$ and the last is point at $x_n = 1.5$. It turns out this only gives us reasonable estimates of our data if `level > 2`. This might differ between such datasets, but a general rule of thumb is that you need a higher-level parameter for datasets that are sparse at some point in their domain.

```
wavelet_2 <- tfb_wavelet(woo_out, level = 2)
wavelet_3 <- tfb_wavelet(woo_out, level = 3)


plot(tfd(woo_out))
lines(wavelet_2, col = 3)
lines(wavelet_3, col = 4)
legend("bottom", legend = c("level = 2", "level = 3"), col = 3:4, lty = "solid")
```

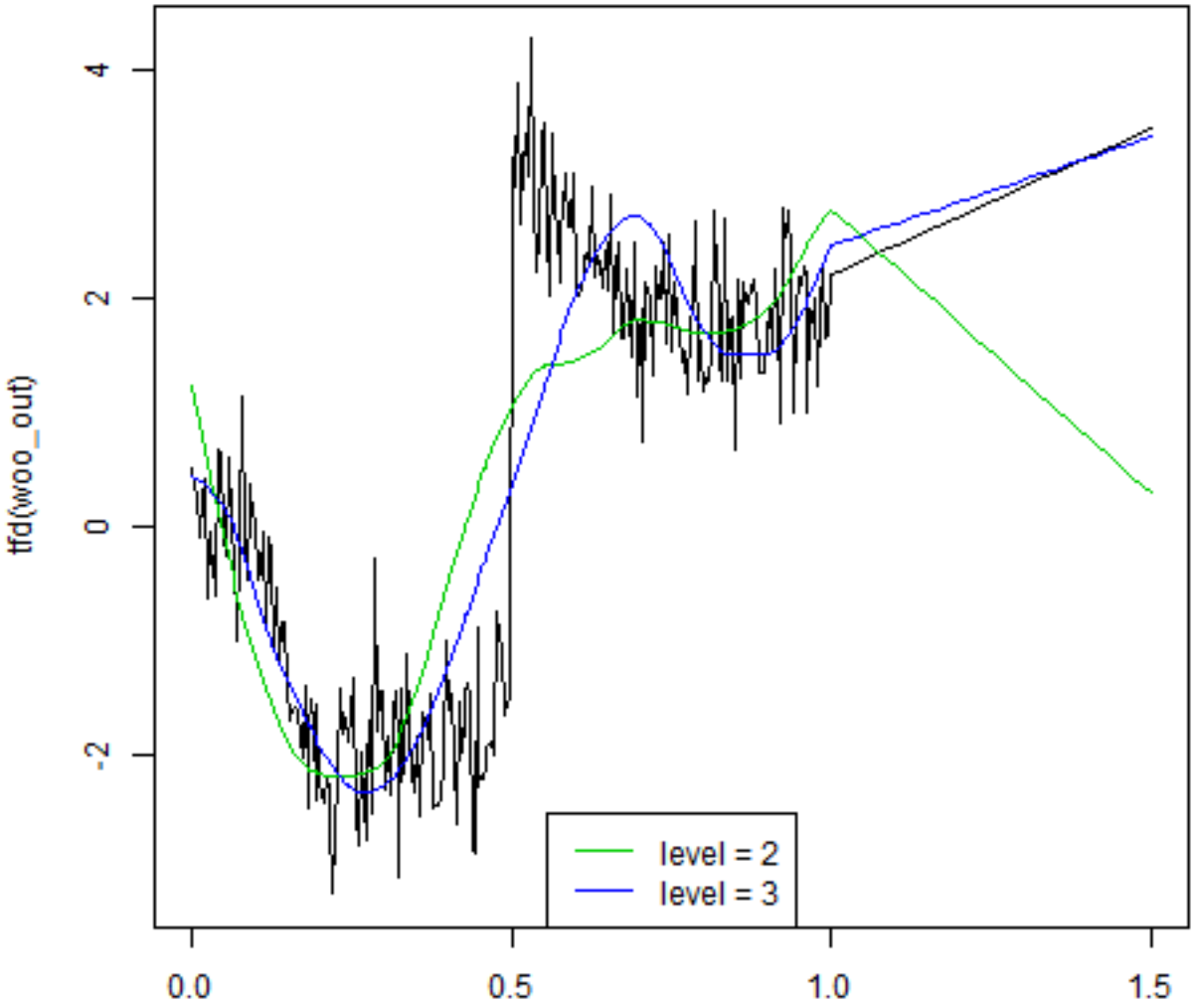

Figure 16: Plot to show the difference between levels two and three with an outlier

The last notable issue is the explosion of our wavelet matrix if our functional data is irregular. Since irregular datasets can potentially have unique support values for every curve, our wavelet matrix can get quite large, because every row corresponds to a unique support value. We combat this by raising the `resolution` parameter, which determines how much tolerance a support value has for being unique. Example from `?tfd`: If an evaluation of $f(t)$ is available at $t = 1$ and a function value is requested at $t = 1.01$, $f(1)$ will be returned if resolution < 0.01[8].

[8]Scheipl and Goldsmith (2019)

3 Conclusion

`tfb_wavelet` works for most data situations and is relativly flexible in its use, as wavelets are very good at estimating fluctuations in the data. Although, especially if lasso regression is used, `tfb_wavelet` is a potentially slow function. We could improve this by introducing a `global` parameter like in the `tfb_spline` function, which samples a percentage of the curves, computes the fit on them and then averages the fits for all curves. This is not tested yet, but could, especially for lasso regression, be a significant performance increase. Another performance increase could be to make the wavelet matrix coarser, by leaving out some rows, for irregular functional data, since right now a potentially big matrix is computed and outputted. In addition to that, the trend column could be optional, also increasing the performance for a few data situations.

Further work can be done on the `tf_derive` function, because this function only works for `tfb_spline` right now and not for `tfb_wavelet`.

The defaults for the wavelet matrix `level = 2` and `filter_number = 5` could be better optimized, but more data needs to be tested to get a conclusion. Right now the defaults are set as to optimize time, but maybe the fit could or should be prioritized.

The last point is that lasso regression right now is a bit of a black box, because the λ's are not included in the output, so this should probably be changed.

4 Appendix

g_l is the filter for d. So for the Haar wavelet this means:

$$g_l = \begin{cases} 2^{-\frac{1}{2}} & l = 0, \\ -2^{-\frac{1}{2}} & l = 1, \\ 0 & \text{otherwise.} \end{cases} \tag{12}$$

h_l is the filter for c. So for the Haar wavelet this means:

$$h_l = \begin{cases} 2^{-\frac{1}{2}} & l = 0, \\ 2^{-\frac{1}{2}} & l = 1, \\ 0 & \text{otherwise.} \end{cases} \tag{13}$$

Common form for equation (5)((Nason 2006), p.21):

$$d_k = \sum_{l=-\infty}^{\infty} g_l y_{2k-l} \tag{14}$$

Common form for equation (6) ((Nason 2006), p.23):

$$c_k = \sum_{l=-\infty}^{\infty} h_l y_{2k-l} \tag{15}$$

Common form for equation (8) ((Nason 2006), p.55):

$$c_{j,n} = \sum_k h_{n-2k} c_{j-1,k} + \sum_k g_{n-2k} d_{j-1,k} \tag{16}$$

References

"9. The Qr Decomposition." n.d. In *LINPACK Users' Guide*, 9.1–9.27. https://doi.org/10.1137/1.9781611971811.ch9.

Ferraty, Frédéric, and Philippe Vieu. 2006. *Nonparametric Functional Data Analysis: Theory and Practice*. Springer-Verlag New York. https://doi.org/10.1007/0-387-36620-2.

Friedman, Jerome, Trevor Hastie, and Rob Tibshirani. 2009. "Regularization Paths for Generalized Linear Models via Coordinate Descent."

Gil, Amparo, Javier Segura, and Nico Temme. 2007. *Numerical Methods for Special Functions*. https://doi.org/10.1137/1.9780898717822.

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman. 2001. *The Elements of Statistical Learning*. Springer Series in Statistics. New York, NY, USA: Springer New York Inc.

Nason, G. P. 2006. *Wavelet Methods in Statistics with R*. https://doi.org/10.1007/978-0-387-75961-6.

Scheipl, Fabian, and Jeff Goldsmith. 2019. *Tidyfun: Tools for Tidy Functional Data*. https://github.com/fabian-s/tidyfun.

Wand, M. P., and J. T. Ormerod. 2011. "Penalized Wavelets: Embedding Wavelets into Semiparametric Regression." *Electron. J. Statist.* 5: 1654–1717. https://doi.org/10.1214/11-EJS652.