Technical report for the paper "On the unification of schemes and software for wavelets on the interval"

Vegard Antun* Øyvind Ryan †

May 27, 2021

Abstract

This technical report fills in the necessary details omitted in the paper [1], but needed in the software implementation found at `https://github.com/oyvindry/wl`. It also gives a short introduction to the software implementation, and some code examples. Only the MatLab interface of the software implementation is used. Note that this document is only available online at ..., and will be frequently updated in parallel with the software implementation.

1 Staggered supports

The matrix C in [1] is obtained from evaluations of polynomials at the integers Let us see how to apply a QR factorization on this, in order to obtain staggered supports, as used by the software. Let A be a matrix, and denote by $A^{\updownarrow}$ and $A^{\leftrightarrow}$ the matrices where the rows or columns of A are reversed, respectively, and by $A^{rev} = (A^{\updownarrow})^{\leftrightarrow}$ (i.e., both rows and columns are reversed). If $(C^{\updownarrow})^T = QR$ is a QR-decomposition of $(C^{\updownarrow})^T$, one has that $C^{\updownarrow} = R^T Q^T$, so that $C = (R^T)^{\updownarrow} Q^T = (R^T)^{rev}(Q^T)^{\updownarrow}$. If a change of coordinates is then applied for the ϕ^b-functions with $P = ((Q^T)^{\updownarrow})^{-1} = Q^{\leftrightarrow}$, C will be transformed to the upper triangular matrix $(R^T)^{rev}$. Note that the final occurence of a nonzero in any column can't occur at both even and odd indices, otherwise the support of a $\phi^b_{0,k}$ would not be at the integers.

2 Initial choice of polynomials

Lemma 4.1 requires an initial choice for a polynomial basis, and the computation of a generalized inverse. Some remarks on the choice of initial polymial basis are in order. The monomial basis is known to be badly conditioned, so it should be avoided. The software implementation currently uses *Gram polynomials* [6]. These are orthogonal polynomials with respect to an inner product which secures that the columns of C are orthogonal, so that we have $C^{\dagger}$ for free. That the vectors $(c_k(-R+1), \ldots, c_k(N-1))$ are orthogonal is the same as the polynomials c_k being orthogonal w.r.t. the inner product

$$\langle f, g \rangle = \sum_{i=-R+1}^{K-1} f(i)g(i).$$

*Department of Mathematics, University of Oslo, Norway
†Department of Mathematics, University of Oslo, Norway

The Gram polynomials are obtained by applying the Gram-Schmidt process to $1, t, t^2, \ldots$ with this inner product, and they satisfy the recursion formula [3]

$$\begin{aligned} P_{-1}(t) &= 0 \\ P_0(t) &= 1 \\ P_{n+1}(t) &= L(t)P_n(t) - \beta_n P_{n-1}(t) \end{aligned}$$

where $L(t)$ is the linear function which maps the interval $[-R+1, K-1]$ onto the interval $[-1 + 1/(R+K-1), 1-1/(R+K-1)]$, and where

$$\beta_n = \frac{n^2}{4n^2-1}\left(1 - \frac{n^2}{(R+K-1)^2}\right).$$

(m in [6] has been replaced with $R+K-1$). Using the recursion formulas for the Gram polynomials, a symbolic expression for C as a matrix with fractional entries is easily obtained. Symbolic computations can thus be addressed.

[7] shows empirically that Bernstein polynomials with a carefully chosen parameter greatly reduce the condition numbers of the matrices we encounter. Future versions of the software implementation will experiment with this.

3 Lower limits for input to the DWT

The condition from Theorem 6.2 in [1] is not sufficient for an m-level DWT to be computable: It may be that the smallest resolution spaces do not have room for all the boundary functions needed in the construction, i.e., $\{\phi^b_{0,k}\}_{k=0}^{N'-1}$ and $\{\psi^b_{0,k}\}_{k=N-K_L}^{N_{0,L}-1}$ at the left edge, and similarly at the right edge. The largest index in $\mathcal{C}^b_1$ for these is $\max(2(N'-1), 2N_{0,L}-1)$, so that $[N-K_L, 2\max(N', N_{0,L})-1]$ contain all these. Defining

$$s_L = K_L - N + 2\max(N', N_{0,L}) \qquad\qquad s_R = K_R - N + 2\max(N', N_{0,R}), \tag{1}$$

it is seen that the requirement $s_L + s_R \leq \dim(\boldsymbol{\phi}^b_1)$ secures that there is room for all constructed boundary functions from resolution 0.

Another requirement is that none of the right edge boundary functions should be needed in the expressions for the left edge functions[1]. Recall that $\phi^b_{0,k}$ can be expressed in terms of $\{\phi^b_{1,s}\}_{s\leq 2k+K-N+R}$, and that $\psi^b_{0,k}$ can be expressed in terms of $\{\phi^b_{1,s}\}_{s\leq 2k+K-N-\tilde{L}+1}$ (it is easily verified that the latter formula is fulfilled for $k = N_0$ as well as for $k \geq N_0$). In particular,

- $\{\phi^b_{0,k}\}_{k=0}^{\max(N',N_0)-1}$ can be expressed in terms of $\{\phi^b_{1,s}\}_{s\leq s_L+R-2}$,
- $\{\psi^b_{0,k}\}_{k=N-K}^{\max(N',N_0)-1}$ can be expressed in terms of $\{\phi^b_{1,s}\}_{s\leq s_L-\tilde{L}-1}$.

The largest index in $\boldsymbol{\phi}^b_1$ here is $s_L + \max(R-2, -\tilde{L}-1)$, so that at most the $t_L = s_L + \max(R-1, -\tilde{L})$ first functions in this basis are needed. These must not intersect the last N' constructed boundary functions in $\boldsymbol{\phi}^b_1$, so that one arrives at the requirement $t_L + N' \leq \dim(\boldsymbol{\phi}^b_1)$. On the right hand side and for the dual wavelet similar computations can be done, so that one ends up with the following result.

[1] This would make the construction even more complicated.

Theorem 3.1. Assume that $\boldsymbol{x}$ and m satisfy (31), and that

$$t_L = s_L + \max(R-1, -\tilde{L}) \qquad t_R = s_R + \max(R-1, -\tilde{L})$$
$$\tilde{t}_L = s_L + \max(\tilde{R}-1, -L) \qquad \tilde{t}_R = s_R + \max(\tilde{R}-1, -L).$$

Then the following hold.

1. An m-level DWT of $\boldsymbol{x}$ is possible if

$$s_L + s_R \leq \dim(\boldsymbol{\phi}_1^b) \qquad t_L + N' \leq \dim(\boldsymbol{\phi}_1^b) \qquad t_R + N' \leq \dim(\boldsymbol{\phi}_1^b).$$

2. An m-level dual DWT of $\boldsymbol{x}$ is possible if

$$s_L + s_R \leq \dim(\boldsymbol{\phi}_1^b) \qquad \tilde{t}_L + N' \leq \dim(\boldsymbol{\phi}_1^b) \qquad \tilde{t}_R + N' \leq \dim(\boldsymbol{\phi}_1^b).$$

Here $\dim(\boldsymbol{\phi}_1^b)$ is given by Equation (33).

Remark 3.2. While the condition from Theorem 6.2 in [1] is equivalent for the DWT and the dual DWT, the new conditions differ between the two. Therefore it may be that more levels can be applied in a DWT than in a dual DWT:

Remark 3.3. $\{\phi_{0,k}^b\}_{k=0}^{N'-1}$ can be expressed in terms of $\{\phi_{0,s}^b\}_{s=0}^{2N'-2+K_L-N+R}$, so that these in particular must be disjoint from the last, modified N' boundary functions, i.e., $2N'-1+K_L-N+R+N' \leq \dim(\boldsymbol{\phi}_1^b)$. In the same way, at the right edge one must have $2N'-1+K_R-N+R+N' \leq \dim(\boldsymbol{\phi}_1^b)$. Combining these one gets

$$\max(K_L, K_R) + 3N' - 1 - N + R \leq \dim(\boldsymbol{\phi}_1^b).$$

Since there already is a constraint on $K_L + K_R$, the left hand side is minimized if one chooses K_L and K_R as equal as possible. Thus, the best way to secure that the $\phi_{0,k}^b$ don't need both left and right edge boundary functions in their synthesis is to choose K_L and K_R as equal as possible. This strategy is taken by the software implementation (due to the more complicated expression for N_0, the same argument for when the $\psi_{0,k}^b$ don't need both left and right edge boundary functions in their synthesis is more involved[2]).

4 Computational expressions for the DWT and the IDWT

At the left edge, G and G^b are equal except for the first s_L- and last s_R columns. It is also seen that both are zero below the first t_L rows of the first s_L columns, and also above the last t_R rows of the last s_R columns. Denoting by G^w the matrix one obtains when cutting a segment from G corresponding to an interval of rows and columns (the rows and columns will be understood from the context. G^w can be interpreted as applying G to a zero-padded input), it follows that

$$G^b - G^w = \begin{pmatrix} A_L & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & A_R \end{pmatrix},$$

where A_L is $t_L \times s_L$, A_R is $t_R \times s_R$, and can be found by taking the computations made for G^b, and subtracting the known values for G^w. The following result can now be proved.

[2]One could also argue in other ways that $K_L \approx K_R$ is optimal: Otherwise one of them must be large, and the corresponding polynomials c_k must be evaluated at many points, leading to high coordinate values $c_k(n)$.

Theorem 4.1. There exist matrices A_L, $\tilde{A}_L$, A_R, and $\tilde{A}_R$ of dimension $t_L \times s_L$, $\tilde{t}_L \times s_L$, $t_R \times s_R$, and $\tilde{t}_R \times s_R$, respectively, so that

$$G^b = G^w + \begin{pmatrix} A_L & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & A_R \end{pmatrix} \qquad \tilde{G}^b = \tilde{G}^w + \begin{pmatrix} \tilde{A}_L & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \tilde{A}_R \end{pmatrix}$$

$$H^b = (\tilde{G}^w)^T + \begin{pmatrix} \tilde{A}_L^T & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \tilde{A}_R^T \end{pmatrix} \qquad \tilde{H}^b = (G^w)^T + \begin{pmatrix} A_L^T & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & A_R^T \end{pmatrix}. \tag{2}$$

A_L, $\tilde{A}_L$, A_R, and $\tilde{A}_R$ are also called *tail handling components*.

Proof. The expressions for G^b and $\tilde{G}^b$ are immediate from the preceding comments. By biorthogonality $(\tilde{G}^b)^T G^b = I$, so that $H^b = (\tilde{G}^b)^T$. Similarly it follows that $\tilde{H}^b = (G^b)^T$. The expressions for H^b and $\tilde{H}^b$ now follow by transposing the two matrices on the right hand sides in the expressions for G^b and $\tilde{G}^b$. □

Theorem 4.1 gives the following computational recipe for the IDWT: Apply G^w to the input, apply A_L to the head, and A_R to the tail of the input, and add the two. Note that

1. for least dissimilar biorthogonal wavelets as above (i.e., $K_L = K_R = N$), A_L and $\tilde{A}_L$ will have different number of rows (but the same number of columns). A_R can be computed by flipping A_L in both directions (i.e., $A_R = ((A_L)^{\updownarrow})^{\leftrightarrow}$), and similarly for $\tilde{A}_L$ and $\tilde{A}_R$.

2. for orthonormal wavelets as above, all tail handling components will have $2N$ columns and $3N - 1$ rows, and $A_L = \tilde{A}_L$, $A_R = \tilde{A}_R$.

Theorem 4.2. For orthonormal wavelets, and for least dissimilar biorthogonal wavelets, assume that $G = \prod_i A_i B_i$ is a lifting factorization. If we replace G^w by $\prod_i A_i^w B_i^w$, Theorem 4.1 still holds, with the tail handling components updated accordingly (but with the same dimensions).

Proof. Both A_i/B_i and A_i^w/B_i^w are expressed in terms of elementary row operations where every even/odd row is changed by adding multiples of the preceeding/succeeding rows. The difference is that A_i/B_i also apply row operations which involve rows < 0 and rows $\geq M$. As a result, $\prod_i A_i B_i$ and $\prod_i A_i^w B_i^w$ will differ in the first and last rows, and the number of different rows will depend on the number of lifting steps. Since both types of wavelets require at most $N + 1$ lifting steps (see [10, 11]), and since the first columns to the right of $A_L/\tilde{A}_L$ start with at least N zeros, it will be impossible to obtain different values below the first N rows (that would require at least $N + 2$ lifting steps). Since the $A_L/\tilde{A}_L$ have at least $N + 1$ rows in these cases, it is also impossible to propagate changes below the $A_L/\tilde{A}_L$ segments. The same can be concluded for the right tail handling components, and it follows that, although the tail handling components themselves are different, everything outside those segments are unchanged. □

5 Lifting

Remark 5 does not go into details on how the lifting steps are defined. For orthonormal wavelets, where the preceding and succeeding components update every second component with different

weights, [10, 11] state these steps as

$$\begin{pmatrix} \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \\ \cdots & 1 & \lambda_1 & 0 & 0 & 0 & \cdots \\ \cdots & 0 & 1 & 0 & 0 & 0 & \cdots \\ \cdots & 0 & \lambda_2 & \underline{1} & \lambda_1 & 0 & \cdots \\ \cdots & 0 & 0 & 0 & 1 & 0 & \cdots \\ \cdots & 0 & 0 & 0 & \lambda_2 & 1 & \cdots \\ \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix} \text{ and } \begin{pmatrix} \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \\ \cdots & 1 & 0 & 0 & 0 & 0 & \cdots \\ \cdots & \lambda_2 & 1 & \lambda_1 & 0 & 0 & \cdots \\ \cdots & 0 & 0 & \underline{1} & 0 & 0 & \cdots \\ \cdots & 0 & 0 & \lambda_2 & 1 & \lambda_1 & \cdots \\ \cdots & 0 & 0 & 0 & 0 & 1 & \cdots \\ \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix},$$

and the Spline 5/3 and CDF 9/7 wavelets can also use these, with $\lambda_1 = \lambda_2$. Regardless of whether $\lambda_1 = \lambda_2$, we call such steps *elementary lifting steps of even and odd type*, respectively. The transpose of an elementary lifting step of even/odd type is an elementary lifting step of odd/even type, regardless whether we consider a G- or G^w-lifting factorization (G^w is defined in Section 4).

Denote now by A_i and B_i lifting steps of even and odd type, respectively, so that $G = \prod_i A_i B_i$. Rather than applying G^w, we would like to use the zero-padded versions A_i^w and B_i^w as our building blocks, i.e. applying $\prod_i A_i^w B_i^w$ (which is in general different from G^w). These take the form

$$\begin{pmatrix} 1 & \lambda_1 & 0 & 0 & \cdots \\ 0 & 1 & 0 & 0 & \cdots \\ 0 & \lambda_2 & 1 & \lambda_1 & \cdots \\ 0 & 0 & 0 & 1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix} \text{ and } \begin{pmatrix} 1 & 0 & 0 & 0 & \cdots \\ \lambda_2 & 1 & \lambda_1 & 0 & \cdots \\ 0 & 0 & 1 & 0 & \cdots \\ 0 & 0 & \lambda_2 & 1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}, \tag{3}$$

respectively. The following result explains that boundary wavelets can be handled with a known lifting factorizations in the same way as stated in Theorem 4.1.

A direct computation based on filters can be used to compute (2). The software implementation also adopts a lifting-based procedure for some wavelets, which essentially halves the number of operations compared to a direct filter-based approach [10, 11]. Some elaboration is needed to explain this adoption. The change of coordinates between $\boldsymbol{\phi}_m$ and $\mathcal{C}_m$ can be factored into what are called *lifting steps* [12, 13]. Also, the Spline 5/3 and CDF 9/7 wavelets only need such steps which update every second component using only the preceding and succeeding components (and with the same weight). The JPEG2000 standard [14] uses these lifting steps for those wavelets. It can be shown [10, 11] that the orthonormal wavelet transforms also can be expressed in terms of the same simple lifting steps (but not necessarily with the same weight on the preceding and sycceeding components), a fact not often mentioned in the literature. The software implementation employs these lifting steps for the mentioned wavelets.

If $G = \prod_i A_i$ is a factorization of a wavelet transform into lifting steps, it is not the case that $G^w = \prod_i A_i^w$ (where the w again denotes cutting a finite segment of the infinite matrix). It is, however, not too hard to see that G^w and $\prod_i A_i^w$ differ only in the blocks in the same places as A_L and A_R. Therefore, an implementation can replace G^w with $\prod_i A_i^w$ (so that lifting is used) if the tail handling components are updated accordingly. Note also that, whenever A is a lifting step, A^T is also a lifting step, and $(A^T)^w = (A^w)^T$. This means that all equations in (2) are easily adapted to lifting. The technical report [2] elaborates further on how the software implementation adopts lifting.

6 Preconditioning

If a wavelet has N vanishing moments, any polynomial of degree less than N will have coordinates in $\boldsymbol{\phi}_m$ which is a polynomial sequence, and have only zero-coordinates in any $\boldsymbol{\psi}_m$. With the introduction of adapted boundary functions, this will be violated at the boundaries. *Preconditioning* (see also Section 5 in [5]) means to transform the coordinates of the adapted boundary functions using a linear transformation, so that any polynomial has coordinates from a polynomial sequence. Section 6 in the technical report [2] explains how the software implementation computes this linear transformation.

Let us elaborate on how to find the preconditioning matrices. Assume again N and $\tilde{N}$ are vanishing moments. The coordinates of a polynomial $p(t)$ are

$$[p(t)]_{\boldsymbol{\phi}_0} = \left(\int_{-\infty}^{\infty} p(t)\tilde{\phi}_{0,n}(t)dt\right)_{n=-\infty}^{\infty}$$
$$[p(t)]_{\boldsymbol{\phi}_0^b} = \left(\int_{0}^{\infty} p(t)\tilde{\phi^b}_{0,n}(t)dt\right)_{n=0}^{\infty}.$$

For $n \geq K$ these are both part of a polynomial sequence, but for $n < K$ only the $\boldsymbol{\phi}_0$-coordinates are from the same polynomial sequence. Preconditioning boils down to finding a linear transformation A so that

$$A\begin{pmatrix} \int_{-\infty}^{\infty} p_k(t)\tilde{\phi}_{0,K-N}(t)dt \\ \vdots \\ \int_{-\infty}^{\infty} p_k(t)\tilde{\phi}_{0,K-1}(t)dt \end{pmatrix} = \begin{pmatrix} \int_{0}^{\infty} p_k(t)\tilde{\phi^b}_{0,0}(t)dt \\ \vdots \\ \int_{0}^{\infty} p_k(t)\tilde{\phi^b}_{0,N-1}(t)dt \end{pmatrix} \tag{4}$$

for $\{p_k\}_{0\leq k<N}$, any given polynomial basis. Let us use the polynomials $p_k(t) = \sum_{n=-\infty}^{\infty} c_k(n)\phi_{0,n}(t)$, with c_k the final polynomials we obtained after biorthogonalization, expanded polynomially to infinity in each direction. The input to A in (4) can clearly be written as the $N \times N$-matrix $C_{[K-N,K),:}$. For the right hand side we have that

$$\begin{aligned}
&\int_0^{\infty} p_k(t)\tilde{\phi^b}_{0,r}(t)dt = \int_0^{\infty}\left(\sum_{n=-\infty}^{\infty} c_k(n)\phi_{0,n}(t)\right)\tilde{\phi^b}_{0,r}(t)dt \\
&= \int_0^{\infty}\left(\sum_{n=-R+1}^{K-1} c_k(n)\phi_{0,n}(t)\right)\tilde{\phi^b}_{0,r}(t)dt \\
&+ \int_0^{\infty}\left(\sum_{n=K}^{\infty} c_k(n)\phi_{0,n}(t)\right)\tilde{\phi^b}_{0,r}(t)dt \\
&= \delta_{k,r} + \sum_{n=K}^{\infty}\sum_{t=-\tilde{R}+1}^{\tilde{K}-1} c_k(n)\tilde{c}_r(t)\int_0^{\infty}\phi_{0,n}(t)\tilde{\phi}_{0,t}(t)dt = \delta_{k,r}.
\end{aligned}$$

Therefore $AC_{[K-N,K),:} = I_N$. i.e. $A = (C_{[K-N,K),:})^{-1}$. In particular when $K = N$ we have that $A = (C_{[0,N),[0,N)})^{-1}$. In conclusion, a unique preconditioning matrix can be found by keeping track of the matrices C and $\tilde{C}$, as is done in Theorem 3.3.

Preconditioning is useful to test whether a wavelet transforms is implemented correctly: When preconditioning is turned on, and a polynomial sequence is sent to the DWT, one should obtain a zero detail component. One should also obtain a polynomial sequence for the low-resolution component,

if this low-resolution component is "postconditioned" in the logical way (it has been observed that not all implementations apply a postconditioning, however).

7 Lossless computations for Spline 5/3 wavelet

The Spline 5/3 wavelet introduced by Cohen, Daubechies and Feauveau in [4] is a biorthogonal wavelet with $N = \tilde{N} = 2$. It has dyadic filter coefficients, making it ideal for a lossless wavelet transform. For this wavelet, and for vector length of the form $M2^m + 1$ (which implies $K_L = K_R = \tilde{K}_L = \tilde{K}_R = 2$ as well), the software implementation computes the preconditioning matrices

$$A^{-1} = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix} \qquad \tilde{A}^{-1} = \begin{pmatrix} -3 & -1/2 \\ 0 & 1 \end{pmatrix}$$

and the tail handling components

$$A_L = \begin{pmatrix} 0 & 7/8 & 0 & 0 \\ -1/2 & -39/32 & 0 & 5/16 \\ 0 & 7/16 & 0 & 5/8 \\ 0 & 7/32 & 0 & -15/8 \\ 0 & 0 & 0 & 5/8 \\ 0 & 0 & 0 & 5/16 \end{pmatrix}$$

$$\tilde{A}_L = \begin{pmatrix} -1/2 & 5/4 & 3/8 & 0 \\ -2 & 3/2 & -1/4 & 0 \\ 1 & -3/4 & 1/8 & 5/4 \\ 0 & 0 & 0 & -5/2 \\ 0 & 0 & 0 & 5/4 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

Again dyadic fractions are seen everywhere. The matrices A_R and $\tilde{A}_R$ in this case equals A_L and $\tilde{A}_L$ with rows and columns flipped, so that they too are dyadic fractions. Since the lifting steps are known also to consist of dyadic fractions in this case, everything can be implemented in a lossless manner. The test code in the software implementation verifies that the DWT and the IDWT invert each error with 0 (exactly) error. For general Spline wavelets the computed matrices will consist of rational numbers, but a simple run of the code shows that they are not necessarily dyadic fractions, and round off errors may occur.

8 Scaling the scaling functions appropriately

In order for filters to give rise to a refinement relation for a scaling function, we must have that $\sum_n (G_0)_n = \sqrt{2}$. For spline wavelets we would like to use filter coefficients which are dyadic fractions, for which $\sum_n (G_0)_n$ would be different from $\sqrt{2}$. We will see that this can be addressed through a different scaling at each resolution. To see this, write $\hat{\phi}_{1,n}(t) = k\phi_{1,n}(t) = k2^{1/2}\phi(2t - n)$. Then we have

$$\phi = \sum_n (G_0)_n \hat{\phi}_{1,n} = \sum_n \frac{(G_0)_n}{\sum_n (G_0)_n} \sqrt{2} \frac{\sum_n (G_0)_n}{\sqrt{2}} k\phi_{1,n}.$$

It follows that, if ϕ is the scaling function in an MRA, then $\phi = \sum_n (G_0)_n \hat{\phi}_{1,n}$ for $k = \frac{\sqrt{2}}{\sum_n (G_0)_n}$. All in all, when $\sum_n (G_0)_n \neq \sqrt{2}$, we can still state refinement relations, but where the functions at resolution 1 are scaled appropriately.

9 Finding the Gramm matrix for the internal scaling functions

The following is also found in [8]. Define the infinite Gramm matrix $(\langle \boldsymbol{\phi}_m, \boldsymbol{\phi}_m \rangle)$ with (k,n)-entry $\int_{-\infty}^{\infty} \phi_{m,n}(t)\phi_{m,k}(t)\,dt$. Using the refinement relation we can write

$$(\langle \boldsymbol{\phi}_0, \boldsymbol{\phi}_0 \rangle) = (G_{:,e})^T (\langle \boldsymbol{\phi}_1, \boldsymbol{\phi}_1 \rangle) G_{:,e}.$$

Letting $\boldsymbol{x}_m$ be the column vector with entries $\int_{-\infty}^{\infty} \phi_{m,n}(t)\phi_{m,0}(t)\,dt$, we see that

$$\boldsymbol{x}_0 = (G_{:,e})^T (\langle \boldsymbol{\phi}_1, \boldsymbol{\phi}_1 \rangle) g_0 = ((G_0)_{:,e})^T G_0 \boldsymbol{x}_1,$$

where we used that filters commute.

Assuming now that scaling functions at resolution 1 are scaled as in Section 8, this means that $\boldsymbol{x}_1 = \frac{2}{(\sum_n (G_0)_n)^2} \boldsymbol{x}_0$. It follows that $(\boldsymbol{x}_0, 1)$ is an eigenpair for $\frac{2}{(\sum_n (G_0)_n)^2} ((G_0)_{:,e})^T G_0$.

Since ϕ has support $[L,R]$, we note also that $\int_{-\infty}^{\infty} \phi_{0,n}(t)\phi_{0,0}(t)\,dt$ can be nonzero only for $n \in J = [L-R+1, R-L-1]$. Restricting to those indices we get the matrix

$$\frac{2}{(\sum_n (G_0)_n)^2} ((G_0)_{:,2J})^T (G_0)_{:,J}.$$

Since G_0 has support $[L,R]$, it is also easy here to restrict the second index: If $J = [J_1, J_2]$, this second index can be restricted to $I = [L+J_1, R+J_2]$. The function `find_x(L, R, g0)` in `bw_compute_left.m` returns $\boldsymbol{x}_0$ in this way, assuming that $\int_{-\infty}^{\infty} \phi(t)\,dt = \int_{-\infty}^{\infty} \tilde{\phi}(t)\,dt = 1$. Both these imply that $\sum_n \phi_{0,n}(t) = \sum_n \tilde{\phi}_{0,n}(t) = 1$, so that the sum of the components in $\boldsymbol{x}_0$ should equal one.

10 Finding the Gramm matrix of the boundary scaling functions

Writing $\boldsymbol{\phi}_0^{b,1} = (X_e)^T \boldsymbol{\phi}_1^{b,1} + (Z_e)^T \boldsymbol{\phi}_1^{b,2}$, where $\boldsymbol{\phi}_m^{b,1} = \{\phi_{m,k}^b\}_{k=0}^{N-1}$ are the boundary functions, $\boldsymbol{\phi}_m^{b,2} = \{\phi_{m,k}^b\}_{k=N}^{K+R+N-2} = \{\phi_{m,k}\}_{k=K}^{2K+R-2}$ the internal functions, we can write

$$\begin{aligned}(\langle \boldsymbol{\phi}_0^{b,1}, \boldsymbol{\phi}_0^{b,1} \rangle) =& (X_e)^T (\langle \boldsymbol{\phi}_1^{b,1}, \boldsymbol{\phi}_1^{b,1} \rangle) X_e \\ &+ (Z_e)^T (\langle \boldsymbol{\phi}_1^{b,2}, \boldsymbol{\phi}_1^{b,2} \rangle) Z_e \\ &+ (X_e)^T (\langle \boldsymbol{\phi}_1^{b,1}, \boldsymbol{\phi}_1^{b,2} \rangle) Z_e \\ &+ (Z_e)^T (\langle \boldsymbol{\phi}_1^{b,2}, \boldsymbol{\phi}_1^{b,1} \rangle) X_e. \end{aligned} \tag{5}$$

Inserting $\boldsymbol{\phi}_1^{b,1} = C^T \begin{pmatrix} \phi_{1,-R+1}|[0,\infty) \\ \cdots \\ \phi_{1,K-1}|[0,\infty) \end{pmatrix}$ we get

$$(\langle \boldsymbol{\phi}_1^{b,1}, \boldsymbol{\phi}_1^{b,2} \rangle) = C^T \begin{pmatrix} \phi_{1,-R+1}|[0,\infty) \\ \cdots \\ \phi_{1,K-1}|[0,\infty) \end{pmatrix} \begin{pmatrix} \phi_{1,K} & \cdots & \phi_{1,2K+R-2} \end{pmatrix} = C^T S_{[-R+1,K-1],[K,2K+R-2]}, \tag{6}$$

where S is the filter with coefficients $\boldsymbol{x}_1$. In the same way,

$$(\langle \boldsymbol{\phi}_1^{b,2}, \boldsymbol{\phi}_1^{b,2} \rangle) = S_{[K,2K+R-2],[K,2K+R-2]}. \tag{7}$$

Using that $(\langle \boldsymbol{\phi}_1^{b,1}, \boldsymbol{\phi}_1^{b,1} \rangle) = \frac{2}{(\sum_n (G_0)_n)^2} (\langle \boldsymbol{\phi}_0^{b,1}, \boldsymbol{\phi}_0^{b,1} \rangle)$, (5) can be written as

$$\begin{aligned}
&(\langle \boldsymbol{\phi}_0^{b,1}, \boldsymbol{\phi}_0^{b,1} \rangle) - \frac{2}{(\sum_n (G_0)_n)^2} (X_e)^T (\langle \boldsymbol{\phi}_0^{b,1}, \boldsymbol{\phi}_0^{b,1} \rangle) X_e \\
&= (Z_e)^T (\langle \boldsymbol{\phi}_1^{b,2}, \boldsymbol{\phi}_1^{b,2} \rangle) Z_e + (X_e)^T (\langle \boldsymbol{\phi}_1^{b,1}, \boldsymbol{\phi}_1^{b,2} \rangle) Z_e + (Z_e)^T (\langle \boldsymbol{\phi}_1^{b,2}, \boldsymbol{\phi}_1^{b,1} \rangle) X_e.
\end{aligned}$$

After inserting (6) and (7), this is easily solved in terms of tensor products as in the paper. The function `find_phi_grammian` in `bw_compute_left.m` calls the function `find_x`, and then finds and returns

1. $(\langle \boldsymbol{\phi}_0^{b,1}, \boldsymbol{\phi}_0^{b,1} \rangle)$ in the variable `phi1_phi1`,
2. $(\langle \boldsymbol{\phi}_1^{b,1}, \boldsymbol{\phi}_1^{b,2} \rangle)$ in the variable `phi1_phi2`,
3. $(\langle \boldsymbol{\phi}_1^{b,2}, \boldsymbol{\phi}_1^{b,2} \rangle)$ in the variable `phi2_phi2`,
4. $\|\phi\|^2$ in the variable `norm_phi_internal_squared`.

These are then used to normalize the boundary scaling functions so that they have equal 2-norms as the internal scaling functions. This is a simple measure taken to improve the Riesz bounds.

Let us apply a change of coordinates, and let P be the change of coordinate matrix from the new to the old basis as in the paper. Since $\boldsymbol{\phi}^{b,2} = P^T \boldsymbol{\phi}^{b,1}$,

$$(\langle \boldsymbol{\phi}^{b,2}, \boldsymbol{\phi}^{b,2} \rangle) = P^T (\langle \boldsymbol{\phi}^{b,1}, \boldsymbol{\phi}^{b,1} \rangle) P.$$

If we choose P to be the diagonal matrix with entries $\sqrt{\frac{\langle \phi, \phi \rangle}{\langle \phi_{0,k}^{b,1}, \phi_{0,k}^{b,1} \rangle}}$ on the diagonal, the matrix $(\langle \boldsymbol{\phi}^{b,2}, \boldsymbol{\phi}^{b,2} \rangle)$ will have all entries on the diagonal equal to $\langle \phi, \phi \rangle$.

11 Finding the Gramm matrix of the boundary mother wavelets

For the boundary mother wavelets we write

$$\begin{aligned}
(\langle \boldsymbol{\psi}_0^{b,1}, \boldsymbol{\psi}_0^{b,1} \rangle) = &\frac{2}{(\sum_n (G_0)_n)^2} (X_o)^T (\langle \boldsymbol{\phi}_0^{b,1}, \boldsymbol{\phi}_0^{b,1} \rangle) X_o \\
&+ (Z_o)^T (\langle \boldsymbol{\phi}_1^{b,2}, \boldsymbol{\phi}_1^{b,2} \rangle) Z_o \\
&+ (X_o)^T (\langle \boldsymbol{\phi}_1^{b,1}, \boldsymbol{\phi}_1^{b,2} \rangle) Z_o \\
&+ (Z_o)^T (\langle \boldsymbol{\phi}_1^{b,2}, \boldsymbol{\phi}_1^{b,1} \rangle) X_o,
\end{aligned}$$

where all quantities on the right hand side already have been computed. The function `find_psi_grammian` in `bw_compute_left.m`. These are then used to normalize the boundary mother wavelets so that they have equal 2-norms as the internal mother wavelets.

12 Computing the values of the scaling functions at the integers

In the function `find_wav_props_from_filters` in `wl_find_kernel_from_from_filters.m`, the values of the scaling function at the integers are computed. We can rewrite

$$\phi(k) = \frac{2}{\sum_k (G_0)_k} \sum_n (G_0)_{n,0} \phi(2k - n)$$

at integer values $k \in [L+1, R-1]$ as

$$\phi(k) = \frac{2}{\sum_k (G_0)_k} \sum_n (G_0)_{2k,2k-n} \phi(2k-n).$$

With $\boldsymbol{z} = (\phi(L+1), \ldots, \phi(R-1))$, this gives

$$(G_0)_{2[L+1,R-1],[L+1,R-1]} \boldsymbol{z} = \frac{\sum_k (G_0)_k}{2} \boldsymbol{z},$$

so that we can obtain the vector of nonzero function values as an eigenvector for $(G_0)_{2[L+1,R-1],[L+1,R-1]}$ with $\frac{\sum_k (G_0)_k}{2}$ as the corresponding eigenvalue. This vector of function values is normalized so that the sum of the function values is one (i.e. so that the standard requirement $\int \phi(t)\, dt = 1$ is met), and stored in the `phivals` variable in the structs `wav_props` (for the wavelet) and `dual_wav_props` (for the dual wavelet).

We also need the values of the boundary scaling functions at the integers. Recall from the paper first that

$$\dim(\boldsymbol{x}) = 2^m M + 1 - (K_L - N) - (K_R - N) - L - R.$$

We would like $\dim(\boldsymbol{x})$ samples at $k2^{-m}$, and there are $2^m M + 1$ such points. We therefore disregard the first $(K_L - N)$ such points at the left edge, and the last $(K_R - N) + L + R$ ones at the right edge, so that we consider the points $\{x_k\}_{k=0}^{\dim(\boldsymbol{x})-1}$, where $x_k = K_L - N + k$.

For $n \geq N'$, $\phi^b_{0,n} = \phi_{0,n+K_L-N}$ has support $n + K_L - N + [L, R]$, so that $\phi^b_{0,n}(x_k) \neq 0$ only when $n + K_L - N + L + 1 \leq x_k \leq n + K_L - N + R - 1$, i.e. when $n + K_L - N + L + 1 \leq k + K_L - N \leq n + K_L - N + R - 1$, i.e. when $n + L + 1 \leq k \leq n + R - 1$.

For $n < N'$, $\phi^b_{0,n}$ has support $[0, n + K_L - N + R]$, so that $\phi^b_{0,n}(x_k) \neq 0$ only when $0 \leq x_k \leq n + K_L - N + R - 1$, i.e., when $N - K_L \leq k \leq n + R - 1$, which is equivalent to $0 \leq k \leq n + R - 1$. It follows that the matrix of function values is $\max(-L, N, R)$-banded

We have that

$$\begin{aligned}
&\begin{pmatrix}
\phi^b_{0,0}(K-N) & \phi^b_{0,1}(K-N) & \cdots & \phi^b_{0,N'-1}(K-N) \\
\phi^b_{0,0}(K-N+1) & \phi^b_{0,1}(K-N+1) & \cdots & \phi^b_{0,N'-1}(K-N+1) \\
\vdots & \vdots & \vdots & \vdots \\
\phi^b_{0,0}(K+R-2) & \phi^b_{0,1}(K+R-2) & \cdots & \phi^b_{0,N'-1}(K+R-2)
\end{pmatrix} \\
&= \begin{pmatrix}
\phi_{0,-R+1}(K-N) & \cdots & \phi_{0,K-1}(K-N) \\
\phi_{0,-R+1}(K-N+1) & \cdots & \phi_{0,K-1}(K-N+1) \\
\vdots & \vdots & \vdots \\
\phi_{0,-R+1}(K+R-2) & \cdots & \phi_{0,K-1}(K+R-2)
\end{pmatrix} C \\
&= \begin{pmatrix}
\phi(K-N+R-1) & \cdots & \phi(-N+1) \\
\phi(K-N+R) & \cdots & \phi(-N+2) \\
\vdots & \vdots & \vdots \\
\phi(K+2R-3) & \cdots & \phi(R-1)
\end{pmatrix} C.
\end{aligned}$$

If S is the filter with coefficients $\boldsymbol{z}$, this equals

- $S_{[-N+1,R-1],[2-K_L-R,0]} C$ at the left, and
- $S_{[-N+1+L+R,R-1],[2-K_R-R,0]} C$ at the right.

This is stored in the variables `phibvals_left` and `phibvals_right`.

We now need to consider the $L \times L$-matrix B with entries $b_{k,n} = \phi^b_{0,n}(k)$. This is banded with

- $d_1 = \max(K_L - N + R - 1, N)$ lower bands,
- $d_2 = \max(K_R - N + R, N)$ upper bands.

To solve systems with B, or multiply with B, we can make a LU factorization of B which exploits the banded structure (so that only the bands are stored). Since in column k only entries $k-d_2, \ldots, k+d_1$ can be nonzero, we can store B in a $(d_1 + d_2 + 1) \times L$ matrix by replacing the row index with that index modulo $d_1 + d_2 + 1$.

13 When $N \neq \tilde{N}$

We need to generalize the tensor product equation 19 in Section 4.2 of [1] when $\tilde{N} > N$. For $r < N$, $s < \tilde{N}$ we get

$$
\begin{aligned}
Y &= \left\langle \begin{pmatrix} \phi^b_{0,0} \\ \vdots \\ \phi^b_{0,N-1} \end{pmatrix}, \begin{pmatrix} \tilde{\phi^b}_{0,0} \\ \vdots \\ \tilde{\phi^b}_{0,\tilde{N}-1} \end{pmatrix} \right\rangle \\
&= \left\langle (X_e)^T \begin{pmatrix} \phi^b_{1,0} \\ \vdots \\ \phi^b_{1,N-1} \end{pmatrix} + (Z_e)^T \begin{pmatrix} \phi^b_{1,N} \\ \vdots \end{pmatrix}, (\tilde{X}_e)^T \begin{pmatrix} \tilde{\phi^b}_{1,0} \\ \vdots \\ \tilde{\phi^b}_{1,\tilde{N}-1} \end{pmatrix} + (\tilde{Z}_e)^T \begin{pmatrix} \tilde{\phi^b}_{1,\tilde{N}} \\ \vdots \end{pmatrix} \right\rangle \\
&= (X_e)^T \left\langle \begin{pmatrix} \phi^b_{1,0} \\ \vdots \\ \phi^b_{1,N-1} \end{pmatrix}, \begin{pmatrix} \tilde{\phi^b}_{1,0} \\ \vdots \\ \tilde{\phi^b}_{1,\tilde{N}-1} \end{pmatrix} \right\rangle \tilde{X}_e + (X_e)^T \left\langle \begin{pmatrix} \phi^b_{1,0} \\ \vdots \\ \phi^b_{1,N-1} \end{pmatrix}, \begin{pmatrix} \tilde{\phi^b}_{1,\tilde{N}} \\ \vdots \end{pmatrix} \right\rangle \tilde{Z}_e \\
&\quad + (Z_e)^T \left\langle \begin{pmatrix} \phi^b_{1,N} \\ \vdots \end{pmatrix}, \begin{pmatrix} \tilde{\phi^b}_{1,0} \\ \vdots \\ \tilde{\phi^b}_{1,N-1} \end{pmatrix} \right\rangle \tilde{X}_e + (Z_e)^T \left\langle \begin{pmatrix} \phi^b_{1,N} \\ \vdots \end{pmatrix}, \begin{pmatrix} \tilde{\phi^b}_{1,\tilde{N}} \\ \vdots \end{pmatrix} \right\rangle \tilde{Z}_e \\
&= (X_e)^T Y \tilde{X}_e + (Z_e)^T \left\langle \begin{pmatrix} 0_{\tilde{N}-N,:} \\ I_{:,:} \end{pmatrix} \right\rangle \tilde{Z}_e \\
&\quad + (X_e)^T C^T \left\langle \begin{pmatrix} \phi_{1,-R+1} \\ \vdots \\ \phi_{1,K-1} \end{pmatrix}, \begin{pmatrix} \tilde{\phi^b}_{1,\tilde{N}} \\ \vdots \end{pmatrix} \right\rangle \tilde{Z}_e + (Z_e)^T \left\langle \begin{pmatrix} \phi^b_{1,N} \\ \vdots \end{pmatrix}, \begin{pmatrix} \tilde{\phi}_{1,-\tilde{R}+1} \\ \vdots \\ \tilde{\phi}_{1,\tilde{K}-1} \end{pmatrix} \right\rangle \tilde{C} \tilde{X}_e \\
&= (X_e)^T Y \tilde{X}_e + (Z_e)^T \left\langle \begin{pmatrix} 0_{\tilde{N}-N,:} \\ I_{:,:} \end{pmatrix} \right\rangle \tilde{Z}_e + (Z_e)^T \left\langle \begin{pmatrix} 0 & I_{\tilde{N}-N} \\ 0 & 0 \end{pmatrix} \right\rangle \tilde{C} \tilde{X}_e \\
&= (X_e)^T Y \tilde{X}_e + ((Z_e)_{[\tilde{N},\infty),:})^T \tilde{Z}_e + ((Z_e)_{[N,\tilde{N}),:})^T \tilde{C}_{[K,\tilde{K}),:} \tilde{X}_e.
\end{aligned}
$$

This gives us the new tensor product equation for finding Y. We also have

$$\left\langle \begin{pmatrix} \phi_{0,K} \\ \vdots \\ \phi_{0,\tilde{K}-1} \end{pmatrix}, \begin{pmatrix} \tilde{\phi^b}_{0,0} \\ \vdots \\ \tilde{\phi^b}_{0,\tilde{N}-1} \end{pmatrix} \right\rangle = \left\langle \begin{pmatrix} \phi_{0,K} \\ \vdots \\ \phi_{0,\tilde{K}-1} \end{pmatrix}, \tilde{C}^T \begin{pmatrix} \tilde{\phi}_{0,-\tilde{R}+1} \\ \vdots \\ \tilde{\phi}_{0,\tilde{K}-1} \end{pmatrix} \right\rangle$$
$$= \begin{pmatrix} 0 & I_{\tilde{N}-N} \end{pmatrix} \tilde{C} = \tilde{C}_{[K,\tilde{K}),:},$$

so that the overall Gramm matrix is $\begin{pmatrix} Y \\ \tilde{C}_{[K,\tilde{K}),:} \end{pmatrix}$. If $Y = LU$ with U rectangular, then

$$\begin{pmatrix} Y \\ \tilde{C}_{[K,\tilde{K}),:} \end{pmatrix} = \begin{pmatrix} L & 0 \\ 0 & I_{\tilde{N}-N} \end{pmatrix} \begin{pmatrix} U \\ \tilde{C}_{[K,\tilde{K}),:} \end{pmatrix}.$$

The corresponding change of coordinates modifies only internal boundary functions, and will preserve staggered supports.

The case $N > \tilde{N}$ goes in a similar way.

Let us make some comments on how we need to change those arguments in order to compute Y in the case of biorthogonal Spline Wavelets on the interval as defined in [9]. In this case we have

$$\begin{aligned}
Y &= \left\langle \begin{pmatrix} \phi^b_{0,0} \\ \vdots \\ \phi^b_{0,s-1} \end{pmatrix}, \begin{pmatrix} \tilde{\phi^b}_{0,d} \\ \vdots \\ \tilde{\phi^b}_{0,d+\tilde{s}-1} \end{pmatrix} \right\rangle \\
&= \left\langle (X_e)^T \begin{pmatrix} \phi^b_{1,0} \\ \vdots \\ \phi^b_{1,s-1} \end{pmatrix} + (Z_e)^T \begin{pmatrix} \phi_{1,-L} \\ \vdots \end{pmatrix}, (\tilde{X}_e)^T \begin{pmatrix} \tilde{\phi^b}_{1,d} \\ \vdots \\ \tilde{\phi^b}_{1,d+\tilde{s}-1} \end{pmatrix} + (\tilde{Z}_e)^T \begin{pmatrix} \tilde{\phi}_{1,-\tilde{L}} \\ \vdots \end{pmatrix} \right\rangle \\
&= (X_e)^T \left\langle \begin{pmatrix} \phi^b_{1,0} \\ \vdots \\ \phi^b_{1,s-1} \end{pmatrix}, \begin{pmatrix} \tilde{\phi^b}_{1,d} \\ \vdots \\ \tilde{\phi^b}_{1,d+\tilde{s}-1} \end{pmatrix} \right\rangle \tilde{X}_e + (X_e)^T \left\langle \begin{pmatrix} \phi^b_{1,0} \\ \vdots \\ \phi^b_{1,s-1} \end{pmatrix}, \begin{pmatrix} \tilde{\phi}_{1,-\tilde{L}} \\ \vdots \end{pmatrix} \right\rangle \tilde{Z}_e \\
&\quad + (Z_e)^T \left\langle \begin{pmatrix} \phi_{1,-L} \\ \vdots \end{pmatrix}, \begin{pmatrix} \tilde{\phi^b}_{1,d-1} \\ \vdots \\ \tilde{\phi^b}_{1,d+\tilde{s}-1} \end{pmatrix} \right\rangle \tilde{X}_e + (Z_e)^T \left\langle \begin{pmatrix} \phi_{1,-L} \\ \vdots \end{pmatrix}, \begin{pmatrix} \tilde{\phi}_{1,-\tilde{L}} \\ \vdots \end{pmatrix} \right\rangle \tilde{Z}_e \\
&= (X_e)^T Y \tilde{X}_e + (Z_e)^T \left\langle \begin{pmatrix} 0_{L-\tilde{L},:} \\ I_{:,:} \end{pmatrix} \right\rangle \tilde{Z}_e \\
&\quad + (X_e)^T C^T \left\langle \begin{pmatrix} \phi_{1,-R+1} \\ \vdots \\ \phi_{1,-L-1} \end{pmatrix}, \begin{pmatrix} \tilde{\phi}_{1,-\tilde{L}} \\ \vdots \end{pmatrix} \right\rangle \tilde{Z}_e + (Z_e)^T \left\langle \begin{pmatrix} \phi_{1,-L} \\ \vdots \end{pmatrix}, \begin{pmatrix} \tilde{\phi}_{1,-\tilde{R}+1} \\ \vdots \\ \tilde{\phi}_{1,-\tilde{L}-1} \end{pmatrix} \right\rangle \tilde{C} \tilde{X}_e \\
&= (X_e)^T Y \tilde{X}_e + (Z_e)^T \left\langle \begin{pmatrix} 0_{L-\tilde{L},:} \\ I_{:,:} \end{pmatrix} \right\rangle \tilde{Z}_e + (Z_e)^T \left\langle \begin{pmatrix} 0 & I_{L-\tilde{L}} \\ 0 & 0 \end{pmatrix} \right\rangle \tilde{C} \tilde{X}_e \\
&= (X_e)^T Y \tilde{X}_e + ((Z_e)_{[-\tilde{L},\infty),:})^T \tilde{Z}_e + ((Z_e)_{[-L,-\tilde{L}),:})^T \tilde{C}_{[-L,-\tilde{L}),:} \tilde{X}_e.
\end{aligned}$$

This gives us an equation involving a tensor product for finding Y.

Details in Section 4.2: If $\tilde{N} > N$, it is straightforward to see that the extended X'_e have the same eigenvalues, with $\tilde{N} - N$ zeros added. It follows that $\rho(X'_e) < 1$, so that $I - (X_e)^T \otimes (\tilde{X}_e)^T$ is nonsingular, so that Y is unique also in this case.

14 Overview of software implementation

The code uses the same variable names as in [1], and the steps provided by Lemma 3.3, 4.1, and 5.7, are followed and referred to from the code.

14.1 Main functionality

The software has two main functions `dwt_impl` and `idwt_impl` which implement a DWT and a IDWT, respectively. These functions can be called with a variety of arguments, specifying the type of wavelet transform and its properties. To compute an `m`-level DWT on a vector `x`, using a wavelet with name `wname`, boundary handling mode `bd_mode`, and prefiltering mode `prefilter_mode`, simply write

```
y = dwt_impl(x, wname, m, bd_mode, prefilter_mode);
```

Here `wname` will be one of the following supported wavelets:

- `'cdf97'` : CDF 9/7 wavelet with $N = \tilde{N} = 4$ vanishing moments.
- `'spline53'` : Spline 5/3 wavelet with $N = \tilde{N} = 2$ vanishing moments.
- `'splineX.Y'` : Spline wavelet with `X` number of vanishing moments for the wavelet, `Y` for the dual wavelet,
- `'pwl0'` : Piecewise linear wavelets with 0 vanishing moments,
- `'pwl2'` : Piecewise linear wavelets with 2 vanishing moments,
- `'Haar'` : The Haar wavelet,
- `'dbX'` : Daubechies orthonormal wavelet with `X` vanishing moments,
- `'symX'` : Symlets. A close to symmetric, orthonormal (Daubechies) wavelet with `X` vanishing moments.

Likewise `bd_mode` can take the values

- `'per'` : Periodic extension,
- `'symm'` : Symmetric extension (not for orthonormal wavelets),
- `'none'` : No boundary handling, in the sense that the input is zero-padded. This corresponds to applying G^w (if the wavelet is implemented in terms of convolution), or $\prod A_i^w B_i^w$ (if the wavelet is implemented in terms of lifting steps).
- `'bd'` : Boundary handling as described in [1].

Note that, when the `'bd'` mode is invoked, computations in the `'none'` mode are also performed by the software, in order to compute the tail handling components. `prefilter_mode` can take the values

- `'none'` : No prefiltering (default),
- `'bd_pre'` : Boundary wavelets with preconditioning as described in Section 6.

Not all combinations of these arguments make sense. For instance it is not possible to apply a symmetric boundary extension to an orthonormal wavelet. In such cases the functions halt, issuing an error.

`dwt_impl` also accepts the following parameters.

- `dims`: The number of dimensions to apply the DWT to. If the input is two-dimensional, this enables the caller to specify whether a two dimensional DWT should be applied, or a one dimenionsal DWT vectorized on the second axis.
- `dual`: Whether the dual transform should be applied or not.
- `transpose`: Whether the transform should be transposed.
- `data_layout`: How data should be assembled. Possible values are `resolution` (lowest resolution first, default), and `time` (sort according to time).

If one wants the matrix G^b (as defined in [1]), or segments of the infinite DWT matrix (which is needed in several results in [1]), `data_layout` should be set to `time`.

14.2 Internal functions and efficient computations

The `dwt_impl` and `idwt_impl` functions compute the filter coefficients and tail handling components (Theorem 4.1) on the fly each time the functions are invoked. This allows the software to compute different boundary functions for different input sizes and makes the functions user friendly. At the same time, this increases the computational time as each call recomputes coefficients. To allow for using precomputed boundary functions, the software has the functions `dwt1_impl_internal` and `idwt1_impl_internal` which will (given the right input) compute a one dimensional DWT and IDWT, respectively, using precomputed boundary functions. Similar functions exist for two and three dimensions.

To use these internal functions we need to first compute the filter coefficients manually. The complete code can be as follows.

```
[wav_props, dual_wav_props] = find_wav_props(m, wname, bd_mode, size(x,1));
[f, prefilter] = find_kernel(wav_props, dual_wav_props, forward, dual, ...
                             transpose, prefilter_mode);

offsets = [wav_propsx.offset_L, wav_propsx.offset_R];

y = dwt1_impl_internal(x, f, m, bd_mode, prefilter, offsets);
```

Let us go through the different pieces of this code.

- `wav_props` and `dual_wav_props`, as returned by `find_wav_props`, are structures (objects in Python) which contain all filter coefficients and matrices related to the boundary handling.
- The `find_kernel` functions return two function handles:
    1. `f(x, bd_mode)` preforms a one level DWT on `x` with boundary mode `bd_mode`,

2. `prefilter(x, forward)` filters `x`, where `forward` is either 0 (postfiltering) or 1 (prefiltering).

- The `offset` parameter is only necessary when using boundary wavelets. It tells `dwt1_impl_internal` that there are $N - K_L$ and $N - K_R$ extra wavelets at each boundary, as described in section 5 in [1].

Since the internal functions avoid precomputation, their execution times should be comparable with those of convolution, since the DWT/IDWT can be expressed in terms of this (and possibly lifting).

Following the code above, it is possible to experiment with custom made kernels - simply implement your own kernel function (taking the same arguments as the ones defined in the software), and use it as input to `dwt1_impl_internal`.

15 Code example – Plotting the boundary adapted functions

Let us consider an example on how to use our code to plot the boundary functions for a Daubechies 2 (DB2) wavelet basis. Assume that $\dim(\boldsymbol{x})$ is a power of two, so that $K_L = K_R = N = 2$. In the basis $\boldsymbol{\phi}_0^b \oplus \boldsymbol{\psi}_0^b$ for V_1^b (which is $2M$-dimensional), one has that

- $\{\boldsymbol{e}_k\}_{k=1}^{N}$ are the coordinates of the scaling functions modified at the left edge (i.e., $\{\phi_{0,k}^b\}_{k=0}^{N-1}$),
- $\{\boldsymbol{e}_k\}_{k=M-N+1}^{M}$ are the coordinates of the scaling functions modified at the right edge,
- $\{\boldsymbol{e}_k\}_{k=M+1}^{M+N}$ are the coordinates of the mother wavelets modified at the left edge (i.e., $\{\psi_{0,k}^b\}_{k=0}^{N-1}$),
- $\{\boldsymbol{e}_k\}_{k=2M-N+1}^{2M}$ are the coordinates of the mother wavelets modified at the right edge,

where we use the MatLab convention of starting vector indices at 1 for the $\boldsymbol{e}_k$'s. The cascade algorithm uses the IDWT to make a change of coordinates to $\boldsymbol{\phi}_m^b$. These coordinates represent good approximations to the functions values when m is high. Iterating $\boldsymbol{e}_k$ through the k-indices listed above, the code snippet below will display each boundary function. A few comments on the code are in order. The left boundary functions are supported on $[0, 2N-1]$, the right edge functions on $[M-2N+1, M]$. This implies that, with $[0, 4N]$ the plot interval, there will be no overlaps between the left and right edge functions (note, however, that Theorem 3.1 says that the left and right boundary functions will be correctly plotted even with $[0, 2N]$ as plot interval). If we apply the cascade algorithm with `m` levels, and choose $\boldsymbol{e}_k \in \mathbb{R}^{4 \cdot 2^m N}$, an approximation to the edge functions is shown.

```
t = linspace(0, 4*N, 4*N*2^m);
coords = idwt_impl(e_k, 'db2', m, 'bd', 'bd_pre');
plot(t, 2^(m/2)*coords, 'k-');
```

In the code example we also use the option `'bd_pre'` to apply preconditioning of the input vector. This ensures that on the first part of the interval (i.e. on $[0, 1]$) and the last part (i.e. on $[4N-1, 4N]$) the plotted sequence sequence is polynomial for the left and right boundary functions, respectively. The resulting plots can be seen in Figure 1.

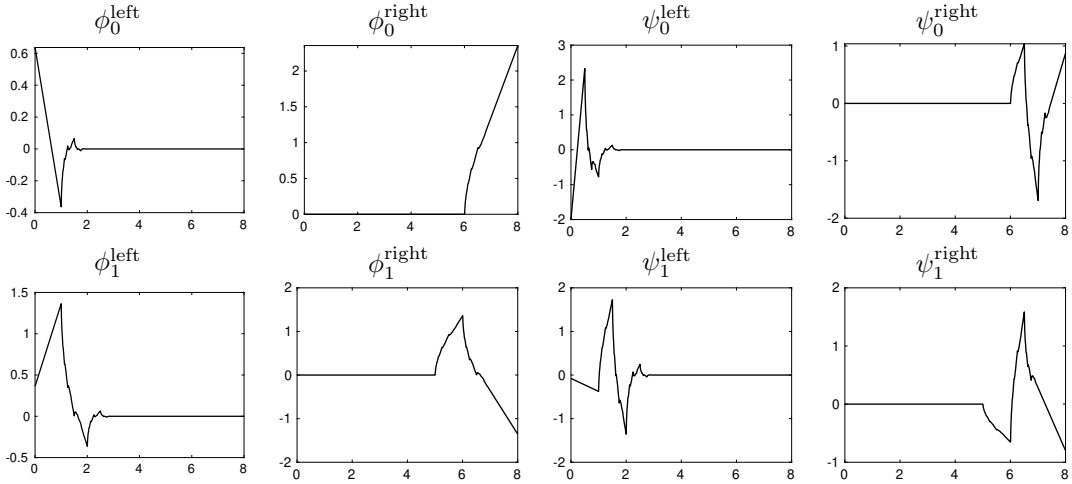

Figure 1: The boundary functions for the DB2 wavelet, computed using the code from Section 15.

References

[1] V. Antun and Ø. Ryan. On the unification of schemes and software for wavelets on the interval. 2019.

[2] V. Antun and Ø. Ryan. Technical report for the paper "on the unification of schemes and software for wavelets on the interval". Technical report, UIO, 2019.

[3] R. W. Barnard, G. Dahlquist, K. Pearce, L. Reichel, and K. C. Richards. Gram polynomials and the Kummer function. *J. Approx. theory*, 94:128–143, 1998.

[4] A. Cohen, I. Daubechies, and J-C. Feauveau. Biorthogonal bases of compactly supported wavelets. *Communications on Pure and Applied Mathematics*, 45(5):485–560, June 1992.

[5] Albert Cohen, Ingrid Daubechies, and Pierre Vial. Wavelets on the interval and fast wavelet transforms. *Applied and Computational Harmonic Analysis*, 1(1):54–81, 1993.

[6] G. Dahlquist and A. Bjorck. *Numerical methods in Scientific computing*, volume 1. Siam, 2008.

[7] W. Dahmen, A. Kunoth, and K. Urban. Wavelets in numerical analysis and their quantitative properties. In *Proceedings of Chamonix*, page 1, 1996.

[8] M. Primbs. On the computation of Gramian matrices for refinable bases on the interval. *International Journal of Wavelets, Multiresolution and Information Processing*, 6(03):459–479, 2008.

[9] M. Primbs. New stable biorthogonal Spline-wavelets on the interval. *Results in Mathematics*, 57:121–162, 2010.

[10] Ø. Ryan. *Linear algebra, signal processing, and wavelets. A unified approach. MATLAB version.* Springer, 2019.

[11] Ø. Ryan. *Linear algebra, signal processing, and wavelets. A unified approach. Python version.* Springer, 2019.

[12] W. Sweldens. The lifting scheme: A new philosophy in biorthogonal wavelet constructions. *Wavelet Applications in Signal and Image Processing III*, pages 68–79, 1995.

[13] W. Sweldens. The lifting scheme: A custom-design construction of biorthogonal wavelets. *Applied and Computational Harmonic Analysis*, 3:186–200, 1996.

[14] D. S. Taubman and M. W. Marcellin. *JPEG2000 Image Compression. Fundamentals, Standards and Practice.* Kluwer Academic Publishers, 2002.