Polyphase Filter Bank write-up

Stephen Fay

June 1st 2021

1 Introduction

2 Forward PFB

(note : the way i have written it the number of output channels is confusingly $n_{chan}+1$, not n_{chan}, to be fixed later???? not sure actually.)

For ease of notation let $n_{chan} \equiv n$, $n_{tap} \equiv m$ and $l_{block} \equiv b = 2(n_{chan}-1)$.
The forward PFB can be represented as three matrices: FSW, applied one after another. **F** is the discrete fourier operator `scipy.fft.rfft()`, it is lossless and easily invertible so it will not cause us too much trouble. **S** is a horizontal stack of n_{tap}, $2n_{chan} \times l_{block}$ square identity matrices; it chops up the signal into n_{tap} pieces and sums them pointwise with each other. And **W** is a square diagonal window matrix (i.e. the sinc function, sometimes multiplied by a hamming window).

$$W : [g_1, \cdots, g_{bm}] \mapsto [g_1 \cdot w_1, \cdots, g_{bm} \cdot w_{bm}]$$

Note: typical values are $n_{tap} = 4$ and $n_{chan} = 1025$.

What the PFB does is it takes a signal (1D array) of length kbm, where k is some natural number (lets say $k = 50$). It eats a bm chunk, and spits out n values. Then it slides along the input signal by b and repeats.

Ignoring the FFT component, i.e. considering only the action of SW on the time-stream. Our forward PFB can be represented in the following way

$$g(t) = [g_1, \cdots, g_{kbm}] \begin{array}{ll} \longrightarrow [g_1, \cdots, g_{bm}] & \xrightarrow{W} [g_1 w_1, \cdots, g_{bm} w_{bm}] \\ \longrightarrow [g_{b+1}, \cdots, g_{b+bm}] & \xrightarrow{W} [g_{b+1} w_1, \cdots, g_{bm+b} w_{bm}] \\ \cdots\cdots\cdots\cdots\cdots & \cdots\cdots\cdots\cdots\cdots\cdots \\ \longrightarrow [g_{kbm-bm}, \cdots, g_{kbm}] & \xrightarrow{W} [g_{kbm-bm} w_1, \cdots, g_{kbm} w_{bm}] \end{array}$$

$$\begin{array}{c} \xrightarrow{S} \\ \xrightarrow{S} \\ \cdots \\ \xrightarrow{S} \end{array} \quad \sum_{j=0}^{m-1} \begin{bmatrix} g_{jb+1}w_{jb+1}\,, & \cdots\,, & g_{b(j+m)}w_{b(j+m)} \\ g_{b(j+1)+1}w_{jb+1}\,, & \cdots\,, & g_{b(j+m+1)}w_{b(j+m)} \\ g_{jb+2b+1}w_{jb+1}\,, & \cdots\,, & g_{b(j+m+2)}w_{b(j+m)} \\ \cdots & \cdots & \cdots \\ g_{b(km-1)+1}w_{jb+1}\,, & \cdots\,, & g_{kbm}w_{b(j+m)} \end{bmatrix}$$

For clarity we write this block with $n_{tap} \equiv m = 4$ and $l_{block} = 2(n_{chan} - 1) \equiv b = 16$ and $k = 50$

$$\begin{bmatrix} g_1w_1 + g_{17}w_{17} + g_{33}w_{33} + g_{49}w_{49} & \cdots & g_{16}w_{16} + g_{32}w_{32} + g_{48}w_{48} + g_{64}w_{64} \\ g_{17}w_1 + g_{33}w_{17} + g_{49}w_{33} + g_{65}w_{49} & \cdots & g_{32}w_{16} + g_{48}w_{32} + g_{64}w_{48} + g_{80}w_{64} \\ \cdots & \cdots & \cdots \\ g_{3137}w_1 + g_{3153}w_{17} + g_{3169}w_{33} + g_{3185}w_{49} & \cdots & g_{3152}w_{16} + g_{3168}w_{32} + g_{3184}w_{48} + g_{3200}w_{64} \end{bmatrix} \tag{1}$$

Now the fourier transform. In Richard Shaw's implementation of the Chime PFB, he doesn't apply the S matrix, instead he applies FW to a block, (where F is `np.fft.rfft`), he applies he samples every n_{tap}'th frequency.

In the end the result is the same. let $\downarrow$ represent down-sampling by a factor of $n_{tap} \equiv m$, then $F_{bm}W = F_bSW$. Indeed they are both $bm \times b$ matrices, when applied to a time-stream vector v we see that

$$\begin{aligned} [\downarrow FWv]_k &= \sum_{j=0}^{bm-1} \exp\left\{-2\pi i \frac{mkj}{mb}\right\} w_j v_j \\ &= \sum_{j=0}^{b-1} \exp\left\{-2\pi i \frac{kj}{b}\right\} \left(\sum_{r=0}^{m-1} w_{br+j} v_{br+j} \right) \\ &= [FSWv]_k \end{aligned}$$

Note that w is a matrix but because it is diagonal we only use one index. There is a funny thing going on with the indexing when we use `rfft`, but everything works out. Applying the real fast fourier transform to an array of even length bm will return an array of length $bm/2 + 1$. In the first case when we down-sample we will end up with $b/2 + 1$ ($\equiv n$) samples, when we apply FSW we end up with the `rfft()` swallowing SWv - which has length $bm/m = b$, thus the output also has n output channels. It is left as an exercise to show that these are indeed equivalent.

3 Inverting the PFB

Reminder: for ease of notation we use $n = n_{chan}$, $m = n_{tap}$, $l_{block} = 2(n-1) = b$.

It is now easy to see that inverting the PFB amounts to retrieving the windowed time-stream points g_i from the above block matrix (1). Each column of this matrix is independent, and is it's own signal. Now we must find how to retrieve g_i from a column, below (2) we use the first column

$$v = \begin{bmatrix} g_1w_1 + g_{17}w_{17} + g_{33}w_{33} + g_{49}w_{49} \\ g_{17}w_1 + g_{33}w_{17} + g_{49}w_{33} + g_{65}w_{49} \\ \cdots \\ g_{3121}w_1 + g_{3137}w_{17} + g_{3153}w_{33} + g_{3169}w_{49} \\ g_{3137}w_1 + g_{3153}w_{17} + g_{3169}w_{33} + g_{3185}w_{49} \end{bmatrix} \longleftrightarrow \begin{bmatrix} \sum_{j=0}^{m-1} g_{bj+1}w_{bj+1} \\ \sum_{j=0}^{m-1} g_{bj+b+1}w_{bj+1} \\ \cdots \\ \sum_{j=0}^{m-1} g_{bmk-b(m-j)}w_{bj+1} \end{bmatrix} \tag{2}$$

Technically speaking this is an impossible problem. To solve this problem practically we introduce circulant boundary conditions by appending the first $m-1$ data-points of our vector (/array) v.

Notice that v this is equivalent to the convolution of

$$\widetilde{g} := [g_1, g_{17}, g_{33}, \cdots, g_{3121}, g_{3137}, g_1, g_{17}, g_{33}]$$

with a flipped, zero padded and rolled chunk of window

$$\widetilde{w} = [w_1, 0, 0, \cdots, 0, 0, w_{49}, w_{33}, w_{17}]$$

Indeed

$$h := \widetilde{w} * \widetilde{g} = [g_1 w_1 + g_{17} w_{17} + g_{33} w_{33} + g_{49} w_{49}\ ,\ g_{17} w_1 + g_{33} w_{17} + g_{49} w_{33} + g_{65} w_{49}\ , \text{etc.}]$$

Invoking the discrete convolution theorem, we see that

$$\mathcal{F}[\widetilde{w} * \widetilde{g}](\xi) \equiv \mathcal{F}[h](\xi) = H(\xi) = W(\xi) \cdot G(\xi)$$

solving for $\widetilde{g} = \mathcal{F}^{-1} G$, we recover the original time-stream.

$$\widetilde{g}[t] = \mathcal{F}^{-1}\Big[H(\xi)/W(\xi)\Big][t] = \mathcal{F}^{-1}\Big[\mathcal{F}[h](\xi)/\mathcal{F}[w](\xi)\Big] = [g_1, g_{17}, g_{33}, \cdots]$$

4 Quantisation effects

The above solution works well in theory. In practice, due to large volumes of data, the time-stream h is quantized. Which can be thought of as introducing noise to $H(\xi)$. Problems arise when $W(\xi) = \mathcal{F}[\widetilde{w}](\xi)$ approaches zero. i.e. when the fourier transform of $[w_1, w_{17}, w_{33}, w_{49}, 0, 0, 0, \cdots]$ is close to zero (the blue parts in Figure 1)

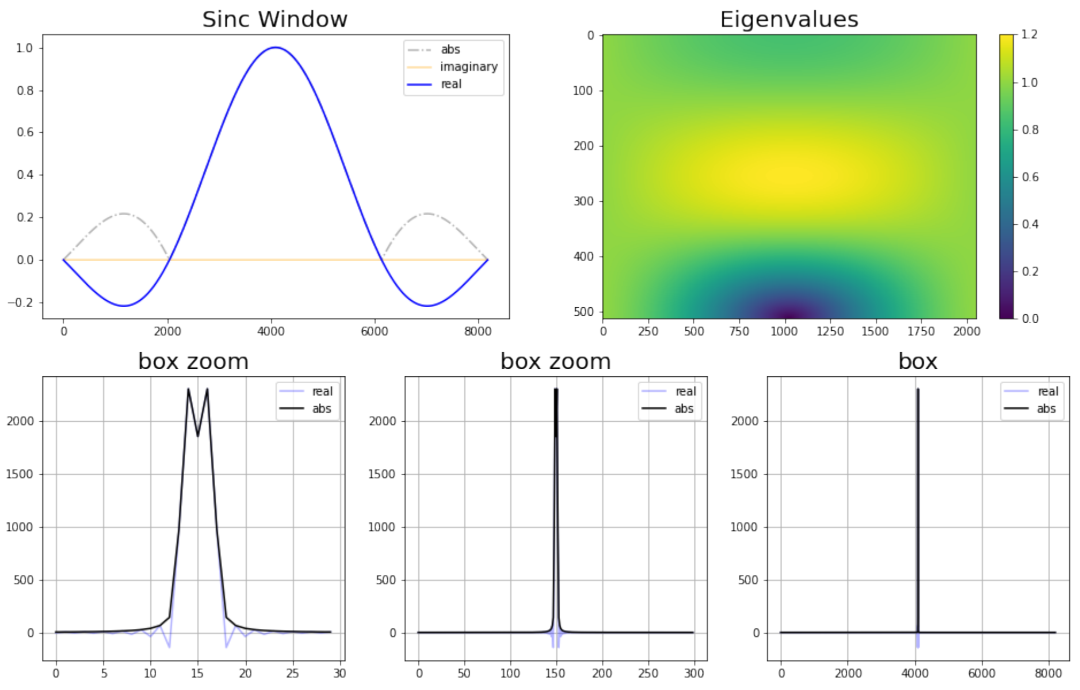

Figure 1: Top left is a plot of the Sinc window currently in use. Top right is the eigenvalue spectrum. The bottom three plots are zooms of the boxcar, i.e. the discrete fourier transform of the Sinc window.

Figure **??** shows us what happens when we introduce Gaussian noise to our filtered signal before inverting the filter, this is effectively what quantizing the signal does.

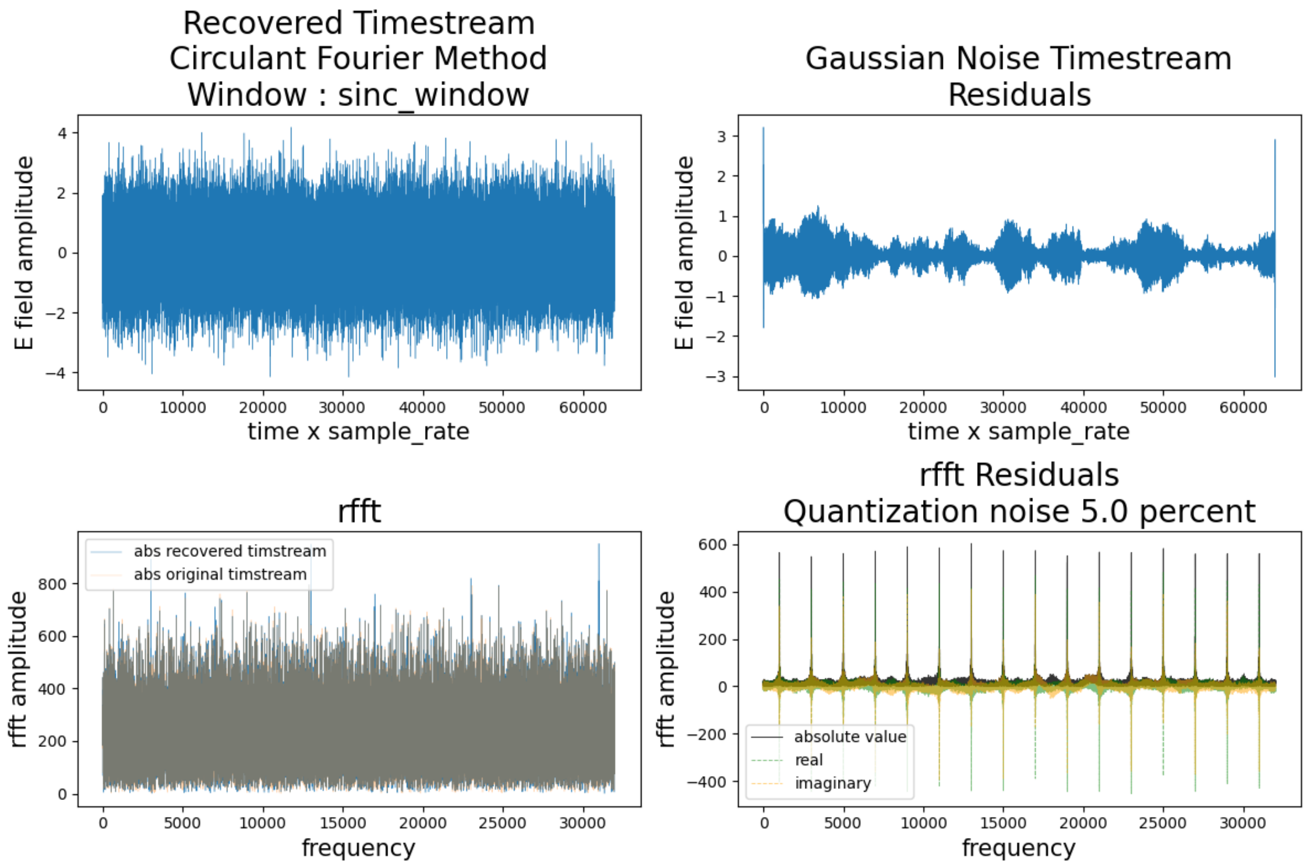

Figure 2: Our input timestream is gaussian noise, we pass it through the PFB, then add some more gaussian noise to it (with sigma at approximately 5% the sigma we used to generate the noise, it's not exact because of scaling from the FFT). Top left: the recovered timestream. Top right: original timstream minus the recovered timestream. Bottom left: the absolute value of the `rfft` (real fast fourier transform) of both the recovered timestream (blue) and the original timestream (orange). Bottom right: the `rfft` of the residuals above it; notice the sharp spikes at regular intervals.

Our task now is to find a window that is close to the sinc (i.e. the FT of the boxcar) so that there is minimal frequency leaking, but also such that

$$\mathcal{F}[w_{k+1}, w_{k+2n_{chan}}, w_{k+4n_{chan}}, w_{k+6n_{chan}}, 0, 0, 0, \cdots, 0, 0](\xi) = W_k(\xi)$$

is never zero.

5 Introducing the William Wallace Window

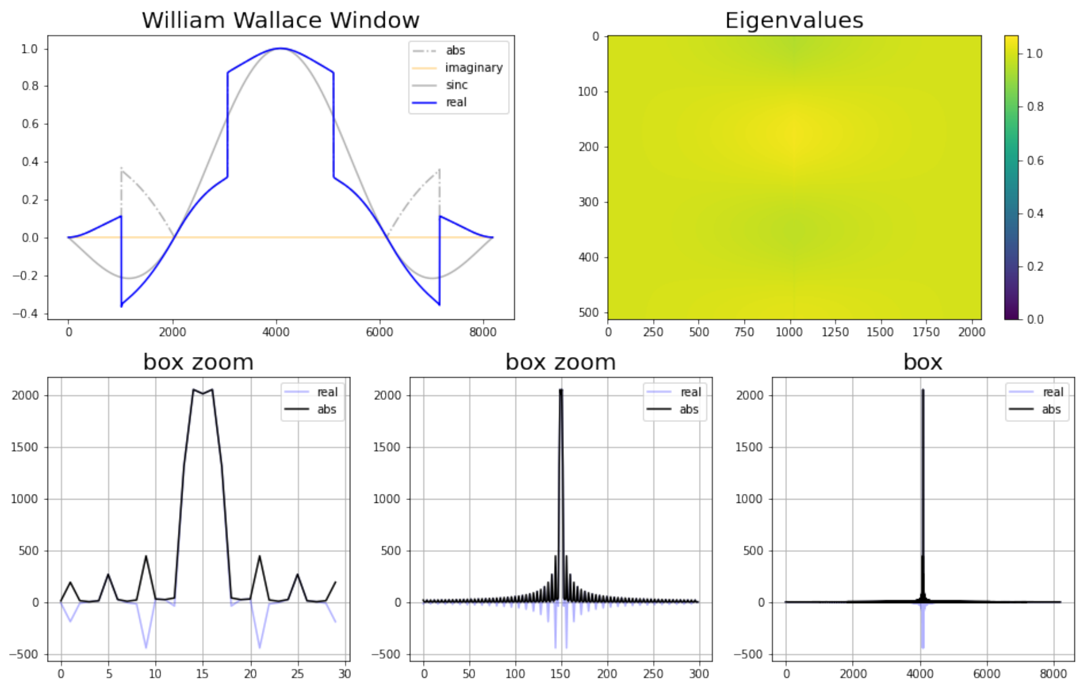

Figure 3: Top left is a plot of the William Wallace Window. Top right is it's eigenvalue spectrum to the same scale as the Sinc window's in figure 1 (see figure 6 in the appendix). The bottom three plots are zooms of the boxcar, i.e. the discrete fourier transform of the William Wallace Window.

Not only does the William Wallace window look like an outline of William Wallace, it also has eigenvalues close to 1.0, in figure 3. However there is unfortunately some frequency leaking. Perhaps we can find an intermediate window, somewhere between the sinc and the William Wallace Window that is more invertible than the sinc, but has less frequency leaking.

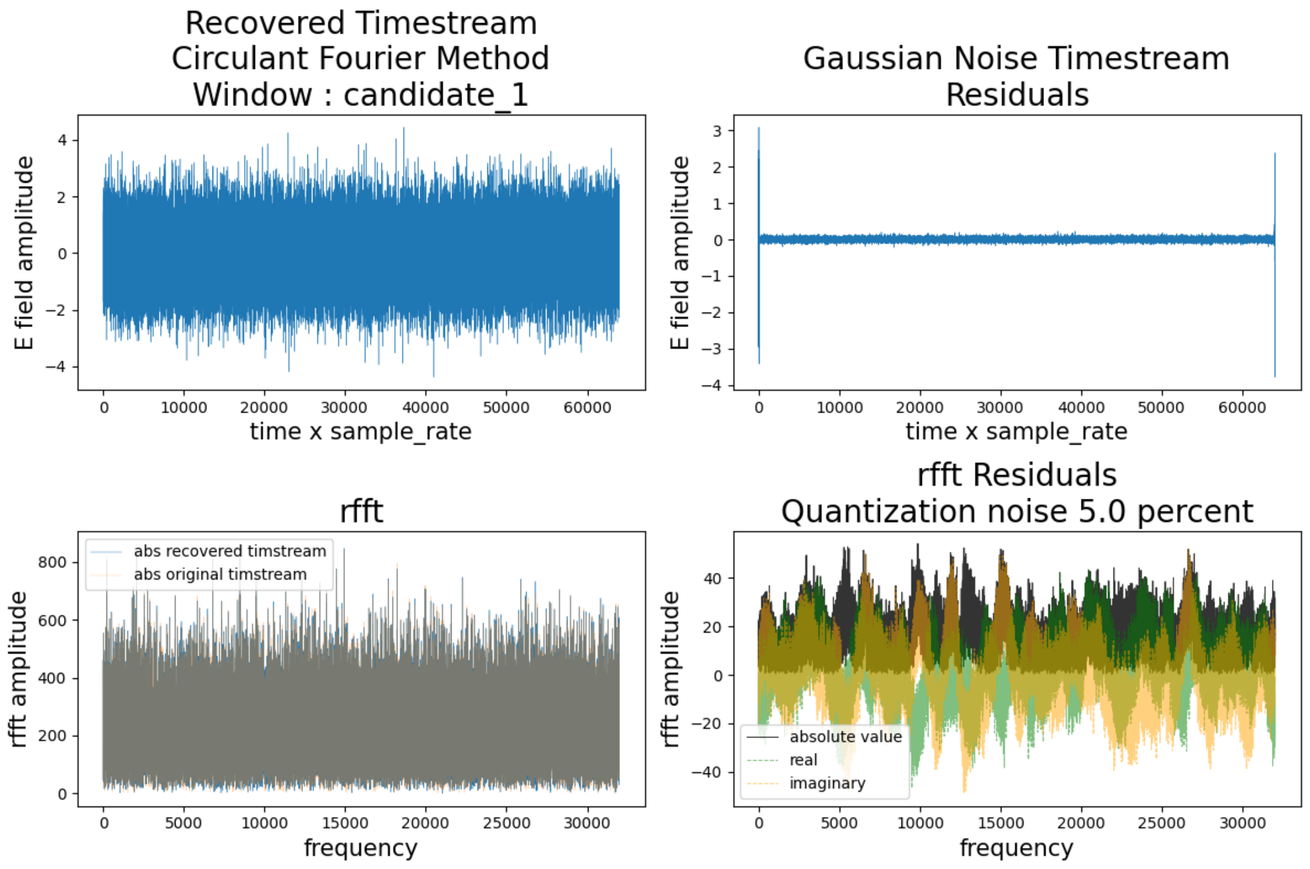

Figure 4: Our input timestream is Gaussian noise, we pass it through the William-Wallace-PFB, then add some more (also gaussian) noise to it (with sigma at approximately 5% the sigma we used to generate timestream, it's not exact because of scaling from the FFT, just approximate). Top left: the recovered timestream. Top right: original timstream minus the recovered timestream. Bottom left: the absolute value of the `rfft` (real fast fourier transform) of both the recovered timestream (blue) and the original timestream (orange). Bottom right: the `rfft` of the residuals above it; notice the lack of sharp spikes, and the scale – to be contrasted with figure 2.

5.1 Numerical criteria that measure how 'bad' the eigenvalues are.

Let Ω be the matrix of values frequency eigenvalues for each w-slice whose image is shown above. Let N be the number of entries in this matrix.

Loss functions:

- Estimate the % of elements of Ω that are smaller than a threshold value.
- Use the sum $\frac{1}{N}\sum_{i,j} 1/\Omega_{i,j}$ or $\frac{1}{N}\sum_{i,j} 1/\Omega_{i,j}^2$

5.2 Efficiently Computing and Sampling Eigenvalues.

Consider the DFT of a real valued array $[a, b, c, d, 0, 0, 0, 0, 0, \cdots, 0]$.

$$\mathcal{F}[a, b, c, d, 0, 0, \cdots, 0](k) = a + be^{-2\pi i\frac{k}{N}} + ce^{-2\pi i\frac{2k}{N}} + de^{-2\pi i\frac{3k}{N}}$$

in a notation where curly brackets denote phase, the k'th entry of the DFT is

$$a\{0\} + b\{k/N\} + c\{2k/N\} + d\{3k/N\}$$

The value of k ranges from 1 to $N/2$ (we can ignore $k > N/2$ because the signal is real, hence FFT symmetric). As we see from figure 1, the values that are causing us trouble by going to zero are the ones at the bottom, when $k = N/2$. These zeros correspond to symmetric w-chunks, where $a = d$ and $b = c$, because at $k = N/2$ the above sum becomes

$$a - b + c - d \mapsto 0$$

In order to evaluate our loss functions, we might want to efficiently sample absolute values from our matrix Ω. To do this we notice that this is the same as sampling a, b, c, d chunks from our window w, and then sampling from

$$|a + b\{\theta\} + c\{2\theta\} + d\{3\theta\}|^2$$

with $\theta \in [0, \pi]$. Which is just a handful of sin and cos functions. Furthermore, because in the full expression, the terms are at most 2 sinusoidals multiplying each other with phases θ,2θ and 3θ, we can be sure that the values of don't vary so much over small distances for small θ, so we can afford to sample sparsely.

6 Frequency Leaking

We use the sinc function because it's inverse is the boxcar. In fact with the parameters that we chose, if we fft our fourier transform we actually obtain something that looks the bottom right hand plot in figure 5

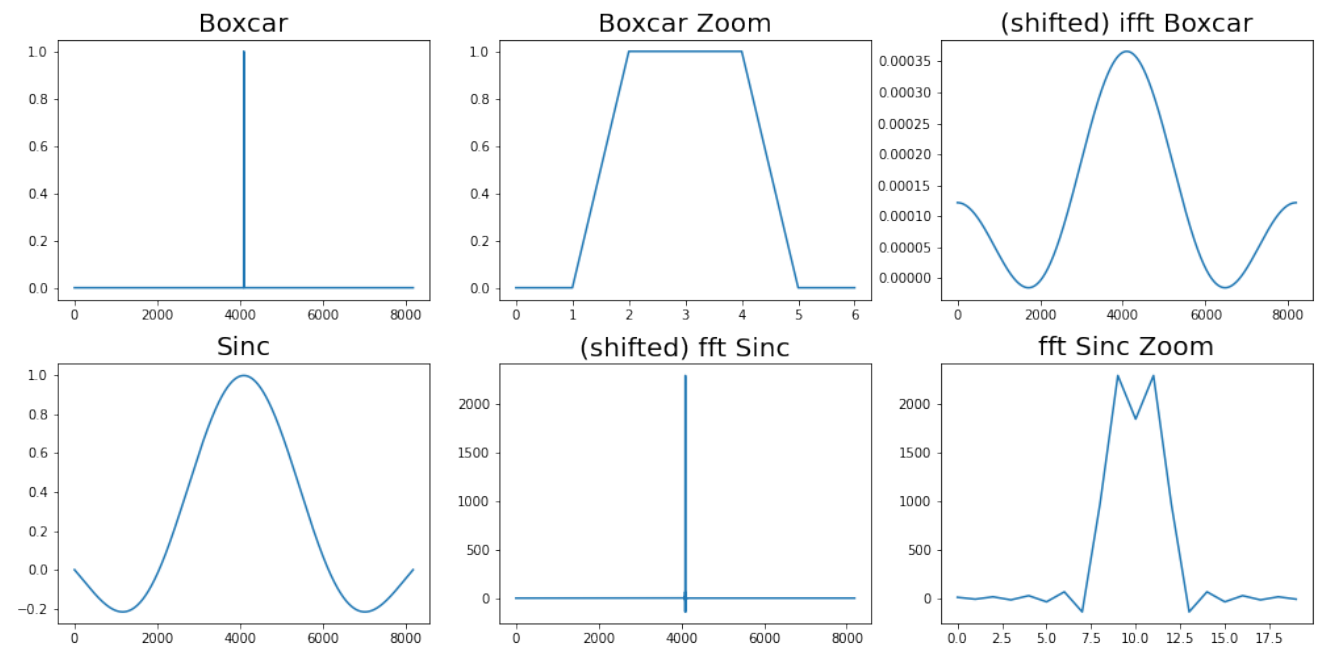

Figure 5: Top left middle: boxcar window, looks like a trapezium but this is an effect of the plotting (because it's so thin). Top right: the (shifted) inverse discrete fourier transform. Bottom left: the exact sinc function implemented in the pfb. Bottom right: the (shifted) discrete fourier transform.

6.1 Numerical criteria that measure how 'bad' the eigenvalues are.

We must measure by how much our result differs from the boxcar that we have, a boxcar that is 3 data-points thick. We must penalize out-of-band leaking: when there are non-zero values of our 'new boxcar' (boxcar in this context refers to `fft(sinc_window)`). And we must penalize in-band rejection: when our new boxcar is rejecting frequencies it ought not to reject.

The simplest way I can think of doing this is by attempting to minimize the norm of the difference between the arrays: `np.linalg.norm(fft(sinc_window) - fft(candidate_window))`

7 Loss function.

To find an optimal filter, we combine our criteria into a single loss function that will penalize both leaky windows and zero eigenvalues.

8 Appendix

8.1 Discrete Fourier Transform

The discrete fourier transform is a square symmetric matrix operator that acts on a complex or real vector space $\mathcal{F} : \mathbb{C}^N \to \mathbb{C}^N$. It's elements are $\mathcal{F}_{k,l} = e^{-2\pi i \frac{k \cdot l}{N}} / \sqrt{N}$. It looks like this

$$\mathcal{F} \equiv \frac{1}{\sqrt{N}} \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 & 1 \\ 1 & e^{-2\pi i/N} & e^{-2\pi i\frac{2}{N}} & \cdots & e^{-2\pi i\frac{N-2}{N}} & e^{-2\pi i\frac{N-1}{N}} \\ 1 & e^{-2\pi i\frac{2\cdot 1}{N}} & e^{-2\pi i\frac{2\cdot 2}{N}} & \cdots & e^{-2\pi i\frac{2(N-2)}{N}} & e^{-2\pi i\frac{2(N-1)}{N}} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & e^{-2\pi i\frac{N-1}{N}} & e^{-2\pi i\frac{2(N-1)}{N}} & \cdots & e^{-2\pi i\frac{(N-2)(N-1)}{N}} & e^{-2\pi i\frac{(N-1)^2}{N}} \end{pmatrix} \tag{3}$$

Taking the fourier transform of a signal (/ array / vector), with `scipy.fft.fft()` is equivalent to multiplying the signal by the above matrix (3). Your computer does this very efficiently using the 'fast fourier transform' which computes it in $N \log N$ time – much better than $\mathcal{O}(N^2)$!

Note: clearly this matrix is full rank since it's columns are independent. It is symmetric and it's eigenvalues are $\{\pm 1, \pm i\}$.

8.1.1 DFT on a real signal

Suppose g is a real length-N array. The DFT of g is G. By examining

$$\sqrt{N}G[j] = \sum_{k=0}^{N-1} g[k]e^{-2\pi i\frac{jk}{N}} \qquad \sqrt{N}G[N-j] = \sum_{k=0}^{N-1} g[k]e^{-2\pi i\frac{(N-j)k}{N}} = \sum_{k=0}^{N-1} g[k]e^{2\pi i\frac{jk}{N}}$$

we notice that the FFT has a the symmetry $G[j] = G[N-j]*$. Hence why we can use a more optimal DFT algorithm which computes only half the entries when dealing with real arrays; namely `scipy.fft.rfft`.

Note: there is a subtlty here, we are indexing starting from zero so the indices are $\{0, 1, \cdots, N-1\}$. The above holds for indices $j \neq 0$. This make sense because the first column is just ones and cannot be identified with any other column. As a result when if your input array is of length n the output of `rfft` will be an array of length $n/2+1$ for even arrays, and $(n+1)/2$ for odd arrays.

8.2 Convolution theorem

Given two Lebesgue integrable functions $g(t), h(t) : \mathbb{R} \to \mathbb{C}$ with fourier transforms G, H resp., i.e.

$$G(x) := \mathcal{F}[g](x) = \int_{-\infty}^{\infty} g(t)e^{-i2\pi xt}dt, \qquad H(x) = \int_{-\infty}^{\infty} h(t)e^{-i2\pi xt}dt, \qquad \text{for } x \in \mathbb{R}$$

then $\mathcal{F}[g * h(t)](x) = G(x) \cdot H(x)$.

Proof.

$$\begin{aligned} \mathcal{F}[g * h(t)](x) &= \mathcal{F}\left[\int_{-\infty}^{\infty} g(\xi)h(t-\xi)d\xi\right] \\ &= \int_{-\infty}^{\infty} dte^{-2\pi ixt} \int_{-\infty}^{\infty} d\xi g(\xi)h(t-\xi) \\ &\overset{fubini}{=} \int_{-\infty}^{\infty} d\xi e^{-2\pi ix\xi} g(\xi) \int_{-\infty}^{\infty} dte^{-2\pi ix(t-\xi)} h(t-\xi) \\ &= G(x) \cdot H(x) \end{aligned} \tag{4}$$

Similarly
$$\mathcal{F}^{-1}\left[G * H(x)\right](t) = g(t)\cdot h(t) \qquad (a.e.\ t)$$

□

8.2.1 Example: convolving a signal with a boxcar.

Let $g(t) : \mathbb{R} \to \mathbb{C}$ be a function and $G(\xi)$, it's fourier transform. Let $B(\xi)$ be a boxcar of width a in frequency space, i.e.
$$B(\xi) = \begin{cases} 1 & \text{if } |\xi| \le a/2 \\ 0 & \text{if } |\xi| > a/2 \end{cases}$$
We can sum frequencies of our signal in a range of width a like so
$$\int_{\xi_0 - a/2}^{\xi_0 + a/2} G(\xi) d\xi$$
which is the same as sampling a point from the convolution
$$\int_{-\infty}^{\infty} B(\xi) G(\xi_0 - \xi) d\xi$$
Invoking the convolution theorem we have
$$\mathcal{F}^{-1}[(B * G)(\xi)](t) = g(t)\cdot b(t)$$
where
$$b(t) = \mathcal{F}^{-1}B = \frac{\sin(\pi t a)}{\pi t} = a\cdot \operatorname{sinc}(\pi a t)$$
so we see that convolving the frequency space signal G with a boxcar is the same as fourier transforming the time series signal multiplied pointwise with the sinc function $g(t)\cdot \operatorname{sinc}(\pi a t)\cdot a$. Notice that if the window a is large, then $\operatorname{sinc}(\pi a t)$ is squished; and if a is small, $\operatorname{sinc}(\pi a t)$ will be quite spread out i.e. there will not be many periods in a given window.

8.3 Discrete Convolutions

The discrete convolution of two 1D arrays g and h of length N is
$$(g * h)[i] = \sum_{j=0}^{N-1} g[j]h[i - j \mod N]$$
We can think of the finite arrays (or vectors) g and h as representing discrete periodic signals of period N.

8.4 Convolution Theorem for Discrete Periodic signals.

For arrays $g, h \in \mathbb{C}^N$ the following equation is true:
$$\mathcal{F}\big[(g * h)[t]\big][\xi] = G[\xi]H[\xi] \tag{5}$$
Where $\mathcal{F}$ is the discrete fourier transform, and $G = \mathcal{F}g$ and $H = \mathcal{F}h$.

Proof.

$$\begin{aligned}
\mathcal{F}\big[(g*h)[t]\big][\xi] &= \sum_{t=0}^{N-1} e^{-2\pi it\xi/N}(g*h)[t] \\
&= \sum_{t=0}^{N-1} e^{-2\pi it\xi/N} \sum_{j=0}^{N-1} g[j]h[t-j] \\
&= \sum_{j=0}^{N-1} e^{-2\pi ij\xi/N} g[j] \sum_{t=0}^{N-1} e^{-2\pi i(t-j)\xi/N} h[t-j] \\
&= G[\xi]\cdot H[\xi]
\end{aligned}$$

□

8.5 Figures

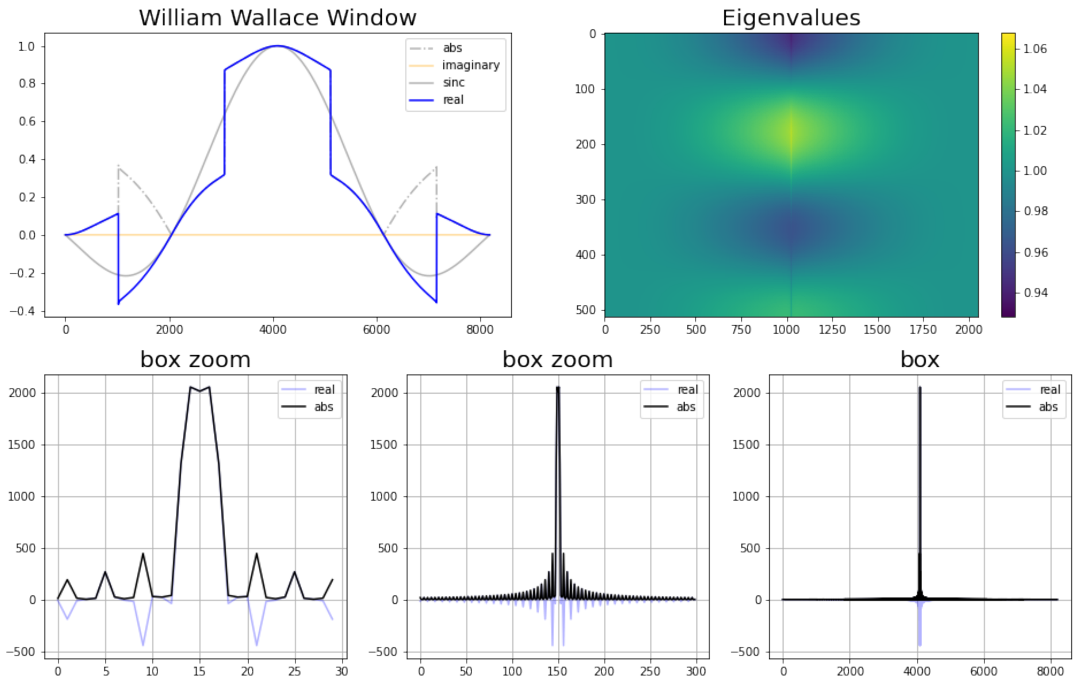

Figure 6: Top left is a plot of the William Wallace Window. Top right is it's eigenvalue spectrum, notice the scale. The bottom three plots are zooms of the boxcar, i.e. the discrete fourier transform of the William Wallace Window.