Inverting the Polyphase Filter Bank draft

Stephen Fay, Jon Sievers

March 14, 2022

Contents

Abstract

The polyphase filter bank (PFB) is a widely used digital signal processing tool used for channelizing input from radio telescopes. Quantization of the channelized signal leads to blow ups in error. We present a practical method for inverting the PFB with minimal quantization-induced error that requires as little as 3% extra bandwidth.

1 Introduction

Polyphase filter banks (PFBs) are widely used in digital signal processing to provide flat response within frequency channels and excellent rejection of out-of-band signals. Several current and upcoming radio arrays (CHIME, HIRAX, ALBATROS, CHORD) will record baseband data that has gone through a PFB, which will later be post-processed. A regular feature of the post-processing is to rechannelize data since finer frequency resolution can be provided by the original (usually FPGA-based) PFBs is often required. The natural way to do this is to invert the PFB to return to the original electric field samples.

Quantizing the output of a PFB leads to channel-specific loss of information, this causes spurious time-localized blow ups in the inverted PFB error (cite paper, JS's write-up). By saving a small amount of terms of the raw input we are able to re-construct a pretty accurate inverse. This is accomplished by casting the problem as a chi-squared minimization. We use the conjugate gradient method to solve it efficiently.

In section 2 we present the forward PFB mathematically, in the language of linear algebra. In section 3 we cover a first attempt at inverting the PFB and take a look at the errors induced by quantization. We also briefly discuss a failed attempt at modifying the sinc-hanning window to rid the virgin PFB of blow-ups in error. In section 4.2 we demonstrate how a signal can be reconstructed from a minimal amount of data.

2 Forward PFB

For ease of notation $n \equiv n_{chan}$ *is the number of output channels of the PFB,* $m \equiv n_{tap}$ *is the number of taps,* $b \equiv l_{block} = 2(n-1)$ *is the length of a segment (or 'block'). Typical numbers are* $n = 1025$*,* $m = 4$ *and* $b = 2048$*.*

The goal of the PFB is to split a stream of electric field data up into frequency channels, where the response inside a channel is flat, and the response outside the channel dies as quickly as possible.

The ideal thing to do would be to Fourier transform the electric field and average over neighbouring frequency channels (average over n_{tap} channels). This is equivalent to convolving the output of the DFT'd signal with a boxcar then down-sampling by a factor of n_{tap}. By the discrete convolution theorem, this is equivalent to fourier transforming the pointwise multiplication of the original signal with the DFT of the boxcar, and down-sampling that. Down-sampling a fourier transform (by a factor of n_{tap}) is the same as fourier-transforming a shorter signal, each term of this shorter signal is just the sum of n_{tap} terms of our original signal. In this way, by reversing the order of operations, we are able to down-sample and compress before executing the fourier transform.

pointwise multiplied by the , this then cut out a boxcar in Fourier space. The PFB tries to do this as well and as efficiently as possible given the real-world constraints of digital signal processing.

We would like to
In theory we would like to fourier transform window of length $b \times m$ of our electric field timestream, , decimate the output by a factor of m, slide the window along the signal by b and repeat.

The core computation of the PFB is the successive application of three matrices **FSW**: **F** is a real DFT; **S** is a horizontal stack of m square $b \times b$ identity matrices; it chops up a segment of raw signal into n_{tap} pieces and sums them pointwise with each other; and **W** is a square diagonal window matrix, it multiplies the raw input block by a sinc-hanning window.

$$W : [g_1, \cdots, g_{bm}] \mapsto [g_1 \cdot w_1, \cdots, g_{bm} \cdot w_{bm}] \tag{1}$$

The forward PFB takes a signal (a *real* 1D array) of length kbm, where k is some natural number (lets say $k = 50$, in reality k will be on the order of 10^5). It eats a bm long segment and spits out n *complex* values, one for each frequency channel. 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

$$\begin{array}{rcl} g(t) = [g_1, \cdots, g_{kbm}] & \begin{array}{l} \longrightarrow [g_1, \cdots, g_{bm}] \\ \longrightarrow [g_{b+1}, \cdots, g_{b+bm}] \\ \cdots\cdots\cdots\cdots\cdots\cdots \\ \longrightarrow [g_{kbm-bm}, \cdots, g_{kbm}] \end{array} & \begin{array}{l} \xrightarrow{W} [g_1 w_1, \cdots, g_{bm} w_{bm}] \\ \xrightarrow{W} [g_{b+1} w_1, \cdots, g_{bm+b} w_{bm}] \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \xrightarrow{W} [g_{kbm-bm} w_1, \cdots, g_{kbm} w_{bm}] \end{array} \\ \begin{array}{l} \xrightarrow{S} \\ \xrightarrow{S} \\ \cdots \\ \xrightarrow{S} \end{array} & \sum_{j=0}^{m-1} & \begin{bmatrix} g_{jb+1} w_{jb+1}\,, & \cdots, & g_{b(j+m)} w_{b(j+m)} \\ g_{b(j+1)+1} w_{jb+1}\,, & \cdots, & g_{b(j+m+1)} w_{b(j+m)} \\ g_{jb+2b+1} w_{jb+1}\,, & \cdots, & g_{b(j+m+2)} w_{b(j+m)} \\ \cdots & \cdots & \cdots \\ g_{b(km-1)+1} w_{jb+1}\,, & \cdots, & g_{kbm} w_{b(j+m)} \end{bmatrix} \end{array}$$

For clarity we write this block with $m = 4$ and $b = 16$ and $k = 50$

$$\begin{bmatrix} g_1 w_1 + g_{17} w_{17} + g_{33} w_{33} + g_{49} w_{49} & \cdots & g_{16} w_{16} + g_{32} w_{32} + g_{48} w_{48} + g_{64} w_{64} \\ g_{17} w_1 + g_{33} w_{17} + g_{49} w_{33} + g_{65} w_{49} & \cdots & g_{32} w_{16} + g_{48} w_{32} + g_{64} w_{48} + g_{80} w_{64} \\ \cdots & \cdots & \cdots \\ g_{3137} w_1 + g_{3153} w_{17} + g_{3169} w_{33} + g_{3185} w_{49} & \cdots & g_{3152} w_{16} + g_{3168} w_{32} + g_{3184} w_{48} + g_{3200} w_{64} \end{bmatrix} \tag{2}$$

Now the fourier transform. In some implementations (such as Richard Shaw's python implementation of the Chime PFB) the S matrix is not applied, instead FW is down-sampled by a factor of m.

In the end the result is the same. Let $\downarrow$ represent down-sampling by a factor of $n_{tap} \equiv m$, and let F_b be the $b \times b$ DFT matrix, then $\downarrow F_{bm} W = F_b S W$. To make this evident first observe that they are both $bm \times b$ matrices, and that their action on an arbitrary vector v is the same

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

Where w_j is the jj'th entry of W, we use a single subscript because W is diagonal. 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.

If we flatten the output of the PFB by concatenating each row, the output is practically the same length as the input time-stream (it's actually $(n_{tap}-1)\cdot l_{block}$ shorter, but we can ignore that given that the input is very long). If we let $m = 4$, then **FSW** is the discrete fourier transform of a stack of four diagonal square matrices $SW = [D_1, D_2, D_3, D_4]$. So we can represent the (row-wise) inverse fourier transformed PFB as a large almost-square matrix

$$
\begin{bmatrix}
D_1 & D_2 & D_3 & D_4 & & & & \\
 & D_1 & D_2 & D_3 & D_4 & & & \\
 & & D_1 & D_2 & D_3 & D_4 & & \\
 & & & \ddots & \ddots & \ddots & \ddots & \\
 & & & & D_1 & D_2 & D_3 & D_4
\end{bmatrix} \tag{3}
$$

3 Inverting the PFB, a first attempt

Reminder: for ease of notation we use $n = n_{chan}$ is the number of output channels of the PFB, $m = n_{tap}$ is the number of taps, $l_{block} = 2(n-1) = b$ is the length of a segment.

Inverting the PFB amounts to retrieving the windowed time-stream points g_i from the above block matrix (2). The first column v of this matrix is (4)

$$
h^{(1)} = \begin{bmatrix}
g_1 w_1 + g_{17} w_{17} + g_{33} w_{33} + g_{49} w_{49} \\
g_{17} w_1 + g_{33} w_{17} + g_{49} w_{33} + g_{65} w_{49} \\
\cdots \\
g_{3121} w_1 + g_{3137} w_{17} + g_{3153} w_{33} + g_{3169} w_{49} \\
g_{3137} w_1 + g_{3153} w_{17} + g_{3169} w_{33} + g_{3185} w_{49}
\end{bmatrix} \longleftrightarrow \begin{bmatrix}
\sum_{j=0}^{m-1} g_{bj+1} w_{bj+1} \\
\sum_{j=0}^{m-1} g_{bj+b+1} w_{bj+1} \\
\cdots \\
\sum_{j=0}^{m-1} g_{bmk-b(m-j)} w_{bj+1}
\end{bmatrix} \tag{4}
$$

Technically speaking this problem is insoluble. To solve it practically we introduce circulant boundary conditions by appending the first $m-1$ terms of h_1 to it's tail.

Notice that v this is equivalent to the convolution of

$$\widetilde{g}^{(1)} := [g_1, g_{17}, g_{33}, \cdots, g_{3121}, g_{3137}, g_1, g_{17}, g_{33}]$$

with a flipped, zero padded and rolled 'chunk' of window

$$\widetilde{w}^{(1)} = [w_1, 0, 0, \cdots, 0, 0, w_{49}, w_{33}, w_{17}]$$

Indeed

$$h^{(1)} := \widetilde{w}^{(1)} * \widetilde{g}^{(1)} = [g_1 w_1 + g_{17} w_{17} + g_{33} w_{33} + g_{49} w_{49}\ ,\ g_{17} w_1 + g_{33} w_{17} + g_{49} w_{33} + g_{65} w_{49}\ ,\ \text{etc.}]$$

Note on the notation: $\widetilde{g}^{(r)}[t]$ is the t'th element of the r'th recovered time-stream $[g_r, g_{r+17}, \cdots]$. The integer r denotes the channel index. We use $r = 1$ so that the notation doesn't become heavy-handed, but this method works for every channel.

Invoking the discrete convolution theorem (see appendix section 7.4) we see that

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

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

$$\widetilde{g}^{(1)}[t] = \mathcal{F}^{-1}\Big[\widehat{h}^{(1)}(\xi)/\widehat{w}^{(1)}(\xi)\Big][t] = \mathcal{F}^{-1}\Big[\mathcal{F}[h^{(1)}](\xi)/\mathcal{F}[\widetilde{w}^{(1)}](\xi)\Big] = [g_1, g_{17}, g_{33}, \cdots] \tag{5}$$

3.1 Quantisation effects

This solution works well if our data is clean and precise. But in practice the vectors $h^{(i)}$ are quantized. Which is effectively the same as introducing Gaussian random noise to $\widehat{h}(\xi)$. Problems arise when $\widehat{w}^{(i)}(\xi) = \mathcal{F}[\widetilde{w}^{(i)}](\xi)$ 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)

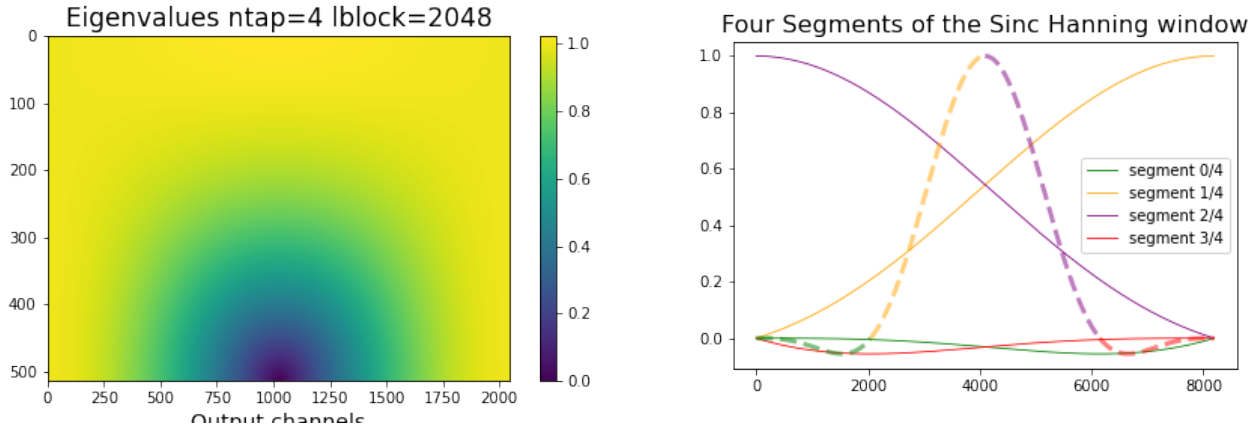

Figure 1: Left Hand Side: On the x axis we have the `irfft` (inverse real discrete fourier transform) of the 1025 output channels of the PFB (rem $(1025-1)\times 2 = 2048$.) The y axis are the absolute values of the fourier transforms of the zero padded sinc-hanning window chunks: $\mathrm{DFT}[w_x, w_{x+2048}, w_{x+2\cdot 2048}, w_{x+3\cdot 2048}, 0, \cdots, 0]$. The middle columns have the unfortunate property that it's four terms are symmetric $w_x \approx w_{x+3\cdot 2048}$ and $w_{x+2048} \approx w_{x+2\cdot 2048}$ the fourier transform contains zeros.
Right Hand Side: The sinc-hanning window and it's four segments. One column of the LHS is generated by concatenating four terms -one from each segment of the corresponding vertical slice of the RHS image- with many zeros (≈ 1000), applying the `rfft` and taking the modulus of each term in the resulting vector.

Lets examine more closely how the mean squared error (MSE) from quantization carries forward. The effect of quantizing a signal $h[n]$ is for our purposes identical to adding a random variable $x[n]$ which is a uniform probability distribution on $(-\Delta/2, \Delta/2]$. Where Δ is the quantization interval. It is easy to see that the MSE induced from the quantization is $\Delta^2/12$ (6)

$$\mathrm{MSE} = \frac{1}{\Delta}\int_{-\Delta/2}^{\Delta/2} dx\, x^2 = \Delta^2/12 \tag{6}$$

An important fact that simplifies our lives is that the expected value of the MSE of the discrete fourier transform of a quantized signal is identical to that of the quantized signal, i.e.

$$\langle |x[n]^2| \rangle = \langle |\widehat{x}[\xi]|^2 \rangle = \left\langle \left| \frac{1}{\sqrt{N}} \sum_{n=1}^{N} x[n] e^{-2\pi i \xi n/N} \right|^2 \right\rangle \tag{7}$$

since

$$\begin{aligned} \langle |\widehat{x}[\xi]|^2 \rangle &= \left\langle \left| \frac{1}{\sqrt{N}} \sum_{n=1}^{N} x[n] e^{-2\pi i \xi n/N} \right|^2 \right\rangle \\ &= \left\langle \frac{1}{N} \sum_{n} x[n]x[n]^* + \frac{1}{N} \sum_{n>m} \left(x[n]x[m]^* + x[n]^* x[m] \right) \right\rangle \\ &= \langle \langle |x[n]|^2 \rangle \rangle = \langle |x[n]|^2 \rangle \end{aligned} \tag{8}$$

The same is true for the inverse DFT. Letting $\sigma^2 = \Delta^2/12$ denote the MSE of our output and channalized signal h (output of forward pfb) and using equation (5) we find the MSE of the reconstructed signal $\widetilde{g}$ (output of the inverse pfb).

$$\begin{aligned}
\mathrm{MSE}(\widetilde{g}^{(n)}[t]) &= \left\langle \left| \mathcal{F}^{-1}[\widehat{x}(\xi)/\widehat{w}^{(n)}(\xi)] \right|^2 \right\rangle \\
&= \left\langle \frac{1}{k} \sum_{\xi=1}^{k} |\widehat{x}(\xi)/\widehat{w}^{(n)}(\xi)|^2 \right\rangle \\
&= \langle \widehat{x}^2 \rangle \cdot \frac{1}{k} \sum_{\xi=1}^{k} \frac{1}{|\widehat{w}^{(n)}(\xi)|^2} \qquad (9) \\
&= \sigma^2 \cdot R[n]^2 \cdot 12 \qquad \text{where} \quad R[n] := \sqrt{\frac{1}{12 \cdot k} \sum_{\xi=1}^{k} \frac{1}{|\widehat{w}^{(n)}(\xi)|^2}} \\
\Rightarrow \mathrm{RMSE}(\widetilde{g}^{(n)}[t]) &= \Delta \cdot R[n]
\end{aligned}$$

Where $\widehat{x}$ is the same uniform random variable on $[-\Delta/2, \Delta/2)$ as before. See figure 2

When $\widehat{w}^{(n)}[\xi]$ has near-zero terms $R[n]$ becomes very large and so too does the the error.

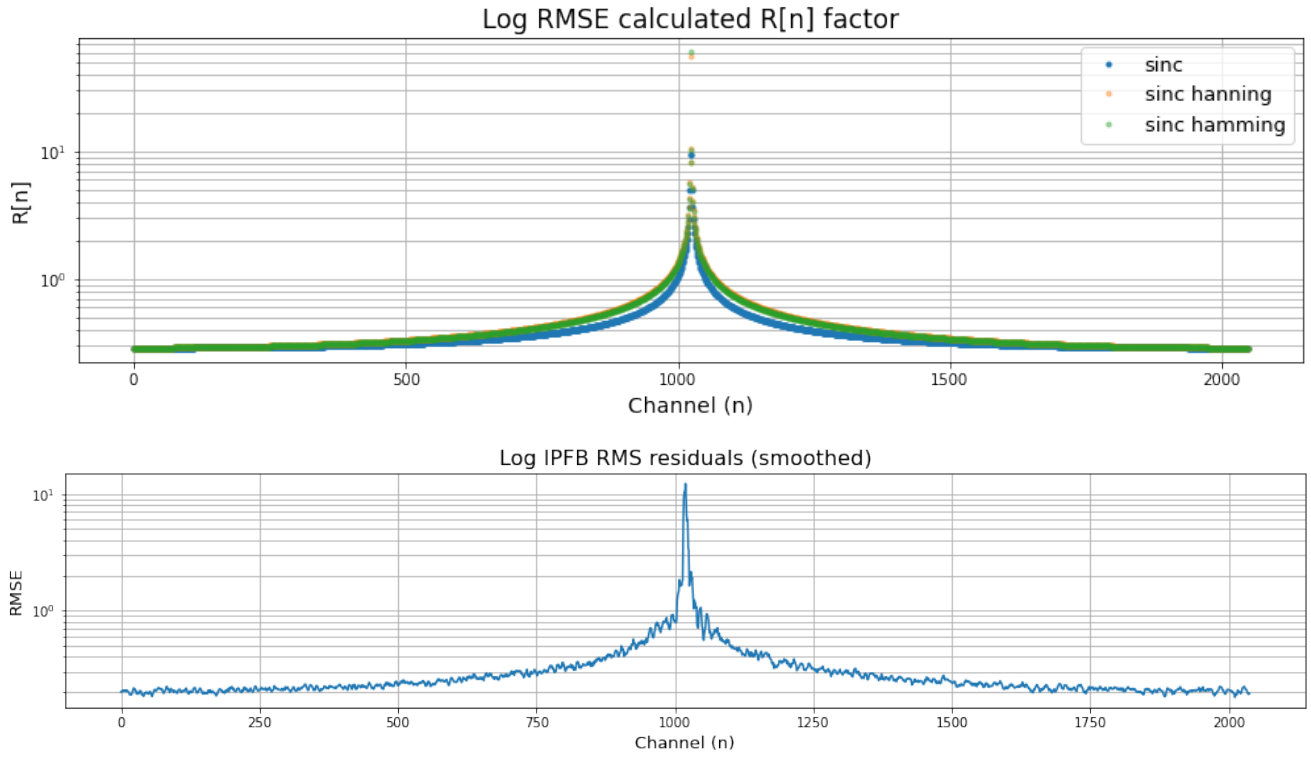

Figure 2: Top: $R[n]$ as described above (9), calculated for three windows. Bottom: the RMSE on data points from each channel for the virgin IPFB with quantization interval $\Delta = 0.5$, obtained via simulation. The fact that the two plots are the same up to a factor of Δ confirms our above derivation of $R[n]$.

3.2 Why do some eigenvalues go to zero?

Let $\omega[n]$ be a sinc hanning window of length $4b$ indexed from $-2b$ to $2b$ (or indeed a sinc / sinc-hamming). These windows are symmetric $\omega[n] = \omega[-n]$. Now consider the DFT of a real valued array $\big[\omega[k-2b], \omega[k-b], \omega[k], \omega[k+b], 0, 0, 0, 0, 0, \cdots, 0\big]$, (which corresponds to the k'th column of eigenvalues from 1.)

$$\mathcal{F}[\omega[k-2b], \omega[k-b], \omega[k], \omega[k+b], 0, 0, \cdots, 0](k) = \omega[k-2b] + \omega[k-b]e^{-2\pi i \frac{k}{4b}} + \omega[k]e^{-2\pi i \frac{2k}{4b}} + \omega[k+b]e^{-2\pi i \frac{3k}{4b}}$$

The value of k ranges from 1 to b (we can ignore $k > b/2$ because the signal is real, hence FFT symmetric). As we see from figure 1, the values that are causing trouble by being small are the ones at the bottom, i.e. when $k = b/2$. These zeros correspond to symmetric w-chunks. By symmetry $\omega[b/2 - 2b] = \omega[b/2 + b]$ and $\omega[b/2 - b] = \omega[b/2]$, therefore the above sum is

$$\omega[-3b/2] - \omega[-b/2] + \omega[b/2] - \omega[3b/2] = 0$$

3.3 Can we modify the PFB window so as to avoid this?

3.3.1 No: a mathematical argument.

Assuming we don't break the symmetry of the sinc-hanning window ω, i.e. that $\mathring{\omega}[n] = \mathring{\omega}[-n]$, where $\mathring{\omega}[n]$ is the modified window. Let $\phi[n] := \mathring{\omega}[n] - \omega[n]$. Then we require that the derivative $\phi'[n]$ be very steep in the middle of each of the four segments. If we consider that any eigenvalue who's absolute value is below 0.1 is "bad" (these are the eigenvalues responsible for magnifying the quantization error.) And we want there to be maximum 20 channels of bad eigenvalues (out of 2048 = 1% of channels). Then we require (10)

$$10 \cdot \phi'[k] > 0.1 \tag{10}$$

for $k = b/2$, $k = 3b/2$. Now consider the fourier decomposition of $\phi(k)$

$$\phi[k] \approx \sum_{s=1}^{8000} a_s \sin(sk/16000) \tag{11}$$

We must keep the side-lobes below 10^{-7} so that there is minimal leaking. Each term a_s contributes *approximately* to a side-lobe as a delta function at the index of it's frequency. Therefore we require that $|a_s| < 10^{-7}$ for each s.

$$\begin{aligned} 10^{-2} \overset{(10)}{<} |\phi'[k]| &\leq 10^{-7} \sum_{s=1}^{8000} \frac{s}{4 \cdot 8000} \frac{2}{\pi} \\ &\approx 10^{-7} \frac{8000^2}{4 \cdot 8000} \frac{1}{\pi} \\ &\approx 10^{-5} << 10^{-2} \end{aligned} \tag{12}$$

In practice if you use this approach the side-lobes reach up to 10^{-4} before there is any noticeable improvement, and there isn't a way of making the zero eigenvalues disappear entirely without side-lobes which go the whole way up to 10^{-2}.

3.3.2 No: by trial and error.

Here is what was tried:

- Gradient descent to optimize the window (a vector of length 8192) with a variety of loss functions that penalize low eigenvalues and large side-lobes.
- GD in the fourier space of the window, with constraints on the fourier coefficients.
- Using GD to optimize a chebyshev (symmetric with constant term 1) filter.
- Modifying the eigenvalues themselves and re-constructing a window from them. This is a little bit non-trivial because the eigenvalue space has ($2\times$) more degrees of freedom than the window space (since they're complex.)
- Manual tinkering of the window: multiplying by polynomials, breaking the symmetry, applying hand crafted distortions to the domain before feeding it through a sinc-hanning function. (this was perhaps unsurprisingly the least successful approach)

For every loss function the gradient descent approach would preserve the symmetry of the window. Even when seeded with an asymmetric window (usually the starting point was the sinc-hanning window.)

See `gradient_descent.py`, `gradient_descent_2.py` and `loss_functions.py` in the GitHub repository, (other failed attempts are scattered in jupyter notebooks, not all pushed.)

4 Correcting the inverse.

4.1 Weiner Filter

4.2 Constructing the most likely inverse-pfb with limited extra bandwidth (about 3% extra).

We examine the expected errors in each channel from the 'virgin' IPFB described in the previous section. Using the notation of equation (4)

$$h^{(r)} = [h^{(r)}[1], h^{(r)}[2], h^{(r)}[3], \cdots, h^{(r)}[T]]$$

Quantizing $h^{(r)}[t]$ is equivalent to adding uniform noise with standard deviation to both real and imaginary components. The standard deviation for each component is $\sigma = \Delta/\sqrt{12}$ where Δ is the quantization interval.

$$H^{(r)}[\xi] = \sum_{t=0}^{T} h^{(r)}[t] e^{-2\pi i t\xi/T}$$

The uncertainty on $h^{(r)}[t]$ induces an uncertainty in it's fourier transform $H^{(r)}[\xi]$. We write $H^{(r)}[\xi] + \hat{x}[\xi]$ where $\hat{x}[\xi]$ is a Gaussian random variable.

$$\hat{x}[\xi] \sim \frac{1}{\sqrt{2\pi\sigma_0^2}} \exp\left(-\frac{1}{2}\frac{y^2}{\sigma_0^2}\right) \qquad \text{normal dist in } y \qquad \sigma_0 = \Delta/\sqrt{6}$$

As before the error carries forward

$$\langle g^{(r)}[t]\rangle = \left\langle \mathcal{F}^{-1}[(H^{(r)}[\xi] + \hat{x}(\xi))/W^{(r)}(\xi)]\right\rangle = \frac{1}{\sqrt{T}}\sum_{\xi} e^{2\pi i t\xi/T} H^{(r)}[\xi]/W^{(r)}[\xi]$$

$$\langle |g^{(r)}[t]|^2\rangle - \langle g^{(r)}[t]\rangle^2 = \left\langle \sum_{\xi=0}^{T-1} |\hat{x}[\xi]|^2/|W^{(r)}[\xi]|^2 \right\rangle \frac{1}{T} = \sigma_0^2 \left\langle \frac{1}{|W^{(r)}[t]|^2}\right\rangle =: \sigma^2_{(r)} \tag{13}$$

We also have another T equations, one for each frequency ξ

$$\frac{1}{\sqrt{T}}\sum_{t} e^{-2\pi i t\xi/T} g^{(r)}[t] = (H^{(r)}[\xi] + \hat{x}[\xi])/W^{(r)}[\xi] \tag{14}$$

$$g^{(r)}[t] = \mathrm{iDFT}\big[(H^{(r)}[\xi] + \hat{x}[\xi])/W^{(r)}[\xi]\big] \tag{15}$$

Suppose that we know one fifth of the values of the discrete time series $g^{(r)}[t]$ (for instance that $g^{(r)}[0], g^{(r)}[4], g^{(r)}[9], ...$ are known.)
we cast it as a chi-squared problem (16). This we can minimize approximately and efficiently using conjugate gradient iterations.

$$\chi^2 = (d - Am)^T N^{-1}(d - Am) + (p - m)^T Q^{-1}(p - m) \tag{16}$$

Where d is the data (output of forward PFB), m is the input to the PFB which we are solving for, A is a matrix representing the forward PFB, N^{-1} is a diagonal quantization noise matrix (with entries $1/\sigma^2$), p is the prior (i.e. our preserved data-points) and Q^{-1} is a diagonal matrix which represents the uncertainties in the prior.

Note on Q^{-1}: for ease of notation we represent p as the ground truth with a bit of quantization noise σ_0 (it's a factor of $\sqrt{2}$ smaller than σ because the quantized terms are real with zero imaginary component), so that Q^{-1} is filled on the diagonal with $1/\sigma_0^2$ at those places where we have preserved the term, and it's filled with 0 in those places where we have not preserved the data point (most of them) representing infinite uncertainty.

4.3 RMSE of simulated data with carefully selected 3% extra.

From the output of the PFB we select and preserve every kth term from the q middle-most channels. The numbers k and q are constrained by how much extra space you have. If k is large we can also afford to have a large q. We need q to be thick enough to cover the worst channels. And we need k to be small enough that it sufficiently fixes the contributions from each channel.

Figure 3 shows the root mean squared errors for $k = 6$ and q such that exactly 3% of the data is preserved. The results are very promising.

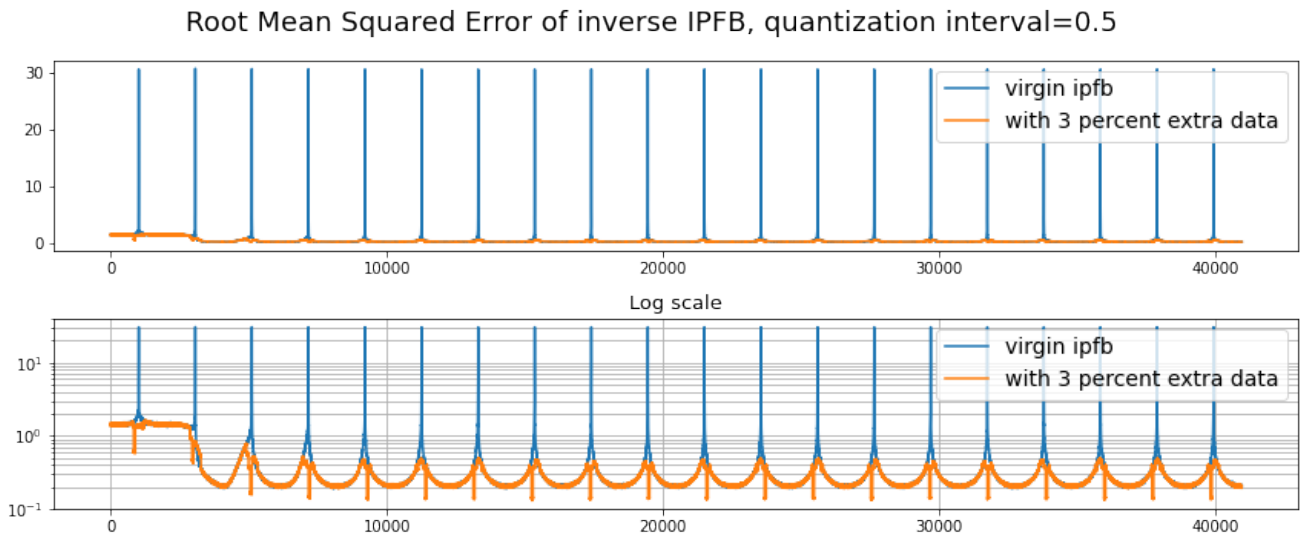

Figure 3: Root mean squared error estimates calculated by simulating data, which is Gaussian Noise with sigma=1.0. The data is first passed through the forward IPFB. It then has both real and imaginary parts quantized with quantization interval 0.5 i.e. a quantization noise sigma of $0.5/\sqrt{12}$ on each component. Then inverted with the method discussed in section 3, and improved upon with 3% additional as described in this section (section 4.2). Every 6'th data point is preserved from the middle 364 channels (out of 2048).

4.3.1 Why does this work?

One way to understand why this works so well is by thinking of a different solution (by constructing a pseudo-inverse PFB matrix from the set of linear equations + the data that we have) that works in theory but not in practice, and then understanding that our chi-squared method will perform better than this solution.

If we consider (14) as a linear system of equations we can substitute the known values of $g^{(r)}$ and remove the need for equations containing small values of $W^{(r)}[\xi]$ (those error magnifying prone equations) effectively removing problematic eigenvalues as suggested in figure 4.

[change this paragraph, don't over-generalize, sometimes solving a system of equations w/ pseudo inverse is a good idea] In this case the pseudo inverse In practice solving a linear system of equations by computing a pseudo-inverse matrix is not a good idea. And our previous method of inverting the PFB is already quite good, except for these noise spikes.

Looking at our eigenvalues plot we can see that the most problematic $\widehat{w}^{(r)}$'s belong to the middle columns. These correspond to the mid frequency-range output channels.

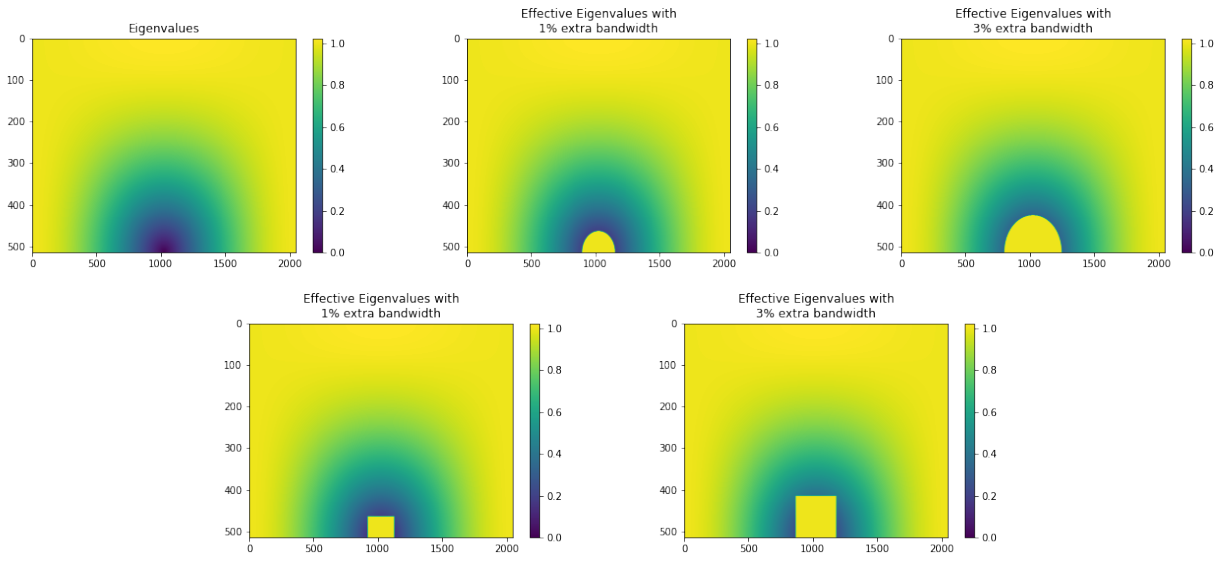

Figure 4: Top from left to right: sinc-hamming window eigenvalues, eigenvalues with 1% smallest blanked, eigenvalues with 3% smallest blanked out.
Bottom: Eigenvalues with 1% blanked out in an implementable way. Eigenvalues with 3% blanked out in an implementable way.
We take our intuition that knowing one extra data-point is roughly equivalent to replacing the worst eigenvalue of it's corresponding column, from the inverse matrix method described in the appendix, because we expect chi-squared minimization to perform better than numerically computing a pseudo-inverse matrix.

4.4 Conjugate Gradient method for optimizing large quadratic systems of equations.

[Little intro to conj grad with some pointers / citations to conj grad stuff. Show plot of how well it performs, and mention edge-effects.]

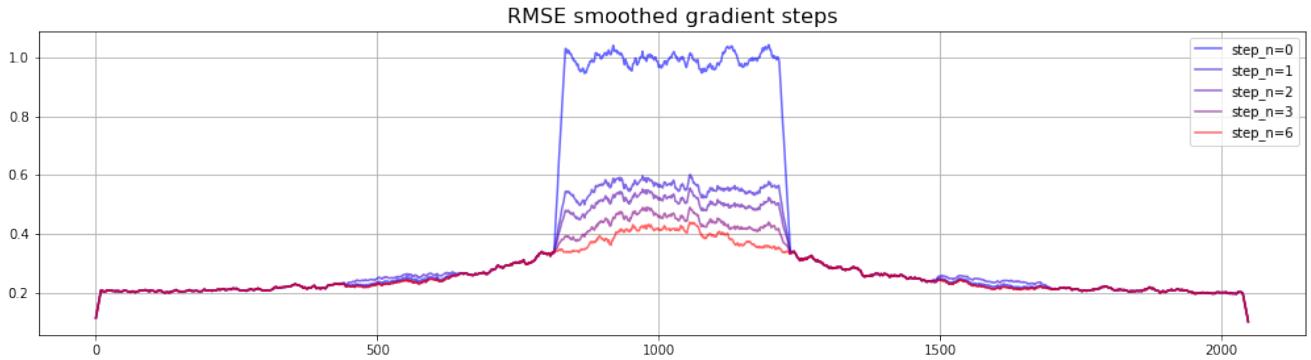

Figure 5: Estimated RMSE during conjugate gradient descent optimization of chi-squared. It converges very fast. The initial guess x_0 is a virgin IPFB with the badly behaved regions set to zero (hence the jump in RMSE.)

Missing figures, put the Wiener filter thing in here. Noisy time-s

5 Discussion

6 Conclusion

7 Appendix

7.1 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

$$\mathcal{F} \equiv \frac{1}{\sqrt{N}} \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 & 1 \\ 1 & e^{-2\pi i/N} & e^{-2\pi i \frac{2}{N}} & \cdots & e^{-2\pi i \frac{N-2}{N}} & e^{-2\pi i \frac{N-1}{N}} \\ 1 & e^{-2\pi i \frac{2\cdot 1}{N}} & e^{-2\pi i \frac{2\cdot 2}{N}} & \cdots & e^{-2\pi i \frac{2(N-2)}{N}} & e^{-2\pi i \frac{2(N-1)}{N}} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & e^{-2\pi i \frac{N-1}{N}} & e^{-2\pi i \frac{2(N-1)}{N}} & \cdots & e^{-2\pi i \frac{(N-2)(N-1)}{N}} & e^{-2\pi i \frac{(N-1)^2}{N}} \end{pmatrix} \tag{17}$$

Taking the fourier transform of a signal (/ array / vector), with `scipy.fft.fft()` is equivalent to multiplying the signal by the above matrix (17). 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 $\{\pm 1, \pm i\}$.

7.1.1 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.

7.2 Convolution theorem

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

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

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

Proof.

$$\begin{aligned}\mathcal{F}[g * h(t)](x) &= \mathcal{F}\left[\int_{-\infty}^{\infty} g(\xi)h(t-\xi)d\xi\right] \\ &= \int_{-\infty}^{\infty} dt e^{-2\pi i x t} \int_{-\infty}^{\infty} d\xi g(\xi)h(t-\xi) \\ &\overset{fubini}{=} \int_{-\infty}^{\infty} d\xi e^{-2\pi i x \xi} g(\xi) \int_{-\infty}^{\infty} dt e^{-2\pi i x(t-\xi)} h(t-\xi) \\ &= G(x) \cdot H(x)\end{aligned} \tag{18}$$

Similarly

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

□

7.2.1 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.

$$B(\xi) = \begin{cases} 1 & \text{if } |\xi| \leq a/2 \\ 0 & \text{if } |\xi| > a/2 \end{cases}$$

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.

7.3 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.

7.4 Convolution Theorem for Discrete Periodic signals.

For arrays $g, h \in \mathbb{C}^N$ the following equation is true:

$$\mathcal{F}\big[(g * h)[t]\big][\xi] = G[\xi]H[\xi] \tag{19}$$

Where $\mathcal{F}$ is the discrete fourier transform, and $G = \mathcal{F}g$ and $H = \mathcal{F}h$.

Proof.

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

□

7.5 Chi squared revisited

In response to physics stack exchange question.

So your measurements are a bunch of normal distributions or random variables (in y)

$$v_n \sim \frac{1}{\sqrt{2\pi\sigma_n^2}} \exp\left\{-\frac{1}{2}\frac{(y-\mu_n)^2}{\sigma_n^2}\right\}$$

What is the probability (or the 'relative weight') of the real being y? Well it's going to be proportional to

$$\text{relative weight}(y) = \Pi_n \frac{1}{\sqrt{2\pi\sigma_n^2}} \exp\left\{-\frac{1}{2}\frac{(y-\mu_n)^2}{\sigma_n^2}\right\}$$

Once you're persuaded yourself the above formula makes sense, notice that to find the most likely value of y all we have to do is maximize it. Since the logarithm function is monotone increasing on $\mathbb{R}^+$ the maximum of this distribution will remain the same if we just take the log of everything

$$\ln(\text{relative weight})(y) = -\frac{1}{2}\sum_n \frac{(y-\mu_n)^2}{\sigma_n^2} - \sum_n \ln\Big(\sqrt{2\pi\sigma_n^2}\Big)$$

To maximize this we can simplify our lives and discard the constant terms, it's equivalent to minimizing

$$\chi^2(y) = \sum_n \frac{(y-\mu_n)^2}{\sigma_n^2}$$

Here y and μ_n can be vectors too but we will keep things simple and assume they are just real numbers (also σ_n). Taking the derivative wrt to y to find extreme of χ^2 we find

$$\frac{d}{dy}\chi^2(y) = 2\sum_n (y-\mu_n)/\sigma_n^2 = 0$$

$$\Rightarrow y_{\text{optimal}} = \left(\sum_n \frac{\mu_n}{\sigma_n^2}\right) / \left(\sum_n \frac{1}{\sigma_n^2}\right)$$

We know that this corresponds to the optimal value because there is only one extreemum. If all the sigmas are the same the formula simplifies to

$$y_{\text{opt}} = \frac{\sum_n \mu_n/\sigma^2}{N/\sigma^2} = \frac{1}{N}\sum_n \mu_n = \text{mean}\{\mu_n\}_{n=1}^N$$

which is just the mean. Turning our attention to the general case, we wish to find the uncertainty in y_{opt}. We found y_{opt} in terms of the means μ_n but we can also sneakily write y_{opt} as a distribution in the following way.

$$y_{\text{opt}} \sim \frac{\sum_n v_n/\sigma_n^2}{\sum_n 1/\sigma_n^2}$$

The variance of the sum is the sum of the variance, thus

$$\text{var}(y_{\text{opt}}) = \left(\sum \frac{1}{\sigma_n^2}\right)^{-1} \cdot \sum_n \frac{\text{var}(v_n)}{\sigma_n^2} = \left(\sum \frac{1}{\sigma_n^2}\right)^{-1}$$

and our final standard deviation is

$$\sigma_{\text{opt}} = \sqrt{\left(\sum \frac{1}{\sigma_n^2}\right)^{-1}}$$

The result you are looking for (or at least, that you where looking for eight years ago) is

$$y_{\text{opt}} \pm \sigma_{\text{opt}} \longleftrightarrow \left(\sum_n \frac{\mu_n}{\sigma_n^2}\right) / \left(\sum_n \frac{1}{\sigma_n^2}\right) \pm \sqrt{\left(\sum_n \frac{1}{\sigma_n^2}\right)^{-1}}$$

7.6 Figures

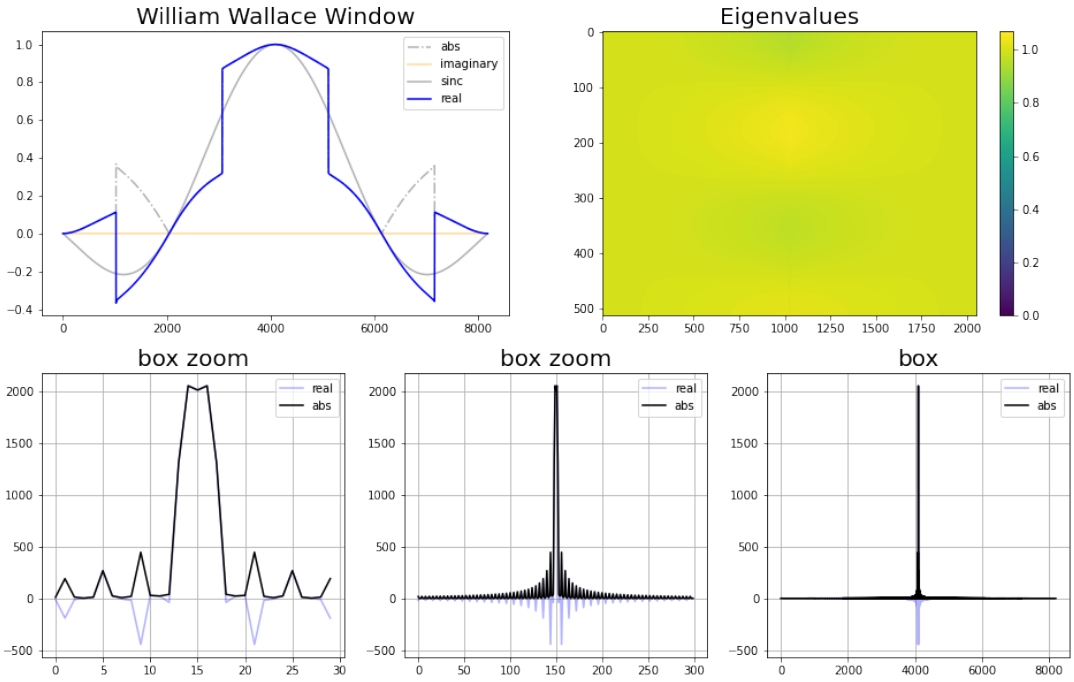

Figure 6: Top left is a plot of a window which has a desirable eigenvalue spectrum (right) but a terrible side-lobe problem (bottom).

7.7 Inverting the PFB for different values of n_{tap}

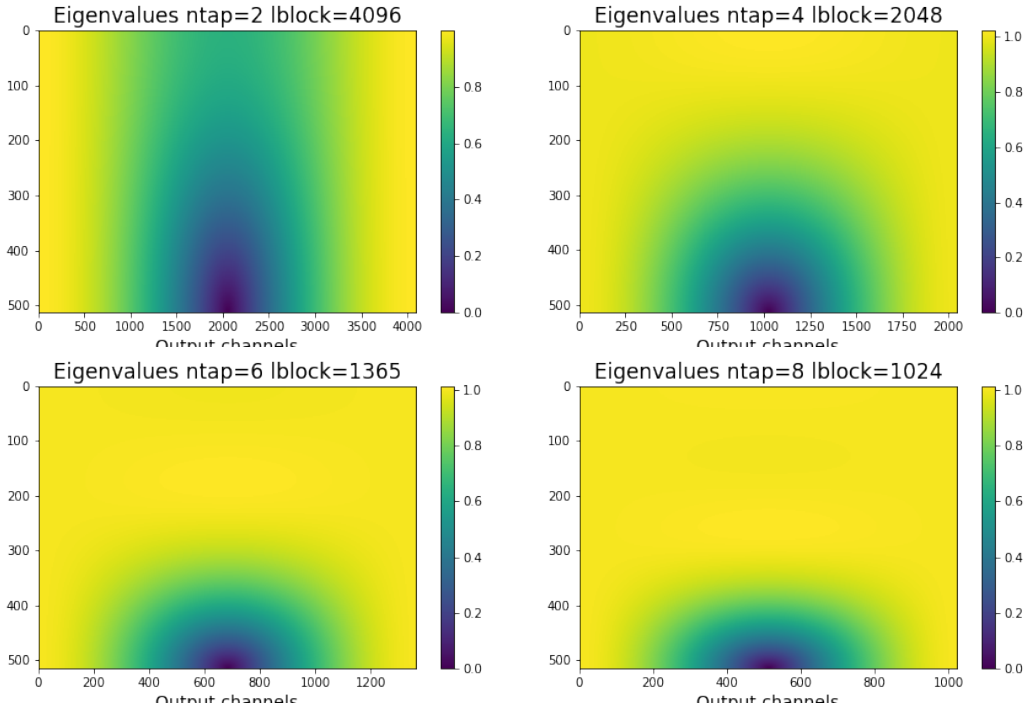

Figure 7: *Link to the code that generates these plots.* Using less taps makes the signal harder to invert.