Version June 17, 2025
Preprint typeset using LaTeX style openjournal v. 09/06/15

INVERTING AND UP-CHANNELIZING CRITICALLY SAMPLED POLYPHASE FILTER BANKS

Stephen Fay, Mohan Agrawal, Simon Tartakovsky, Jon Sievers, Maya Goss
Version June 17, 2025

ABSTRACT

Polyphase Filter Banks (PFBs) are a common signal-processing method used in astronomical interferometers to channelize digitized time-domain data in real-time. Offline data analysis often requires finer frequency resolution than is practical to achieve in real-time systems. We present an efficient, Fourier-transform based algorithm to invert the effects of PFBs to enable up-channelizing data offline. Data channelized with a PFB is typically fixed-point, of limited bit-depth, and often re-quantized after the FFT. Quantization noise and digital artifacts are greatly amplified by our inverse-PFB. We present two de-noising methods: 1) applying a Wiener filter, which is fast and practical, and 2) a maximum-likelihood conjugate-gradient-based optimization approach, which is slower but more rigorous. We evaluate the correlation loss of re-quantized-and-up-channelized simulated data and find negligible difference between a Wiener filter and optimization without a prior.

1. INTRODUCTION

Polyphase filter banks (PFBs) are widely used by radio telescopes to split incoming electric field samples into finer frequency channels. Hardware constraints often limit the number of channels a PFB can create in real-time. Offline inversion/up-channelization of archived PFB data are increasingly common use-cases where finer frequency resolution is desired, such as in Very Long Baseline Interferometry (VLBI) and localization of Fast Radio Bursts (FRBs). The Array of Long Baseline Antennas for Taking Radio Observations from the Seventy-ninth parallel (ALBATROS), which performs correlations offline, will also need up-channelization to meet its science goals (Chiang *et al.* (2020)). Unfortunately, the "standard" of critically sampled PFB, where the output data rate from the PFB equals the input sampling rate, has a poorly conditioned inverse, with quantization noise leading to large artifacts in a reconstructed timestream. One alternative is to oversample the PFB (Morrison *et al.* (2020)), where the output data rate is some substantial factor larger than the native rate. Modern interferometers can be in the terasample/s range, and so oversampling the PFB can get very expensive very quickly. Here, we present an efficient, Fourier-transform based algorithm to invert the effects of PFBs to enable up-channelizing data offline. We compare two approaches to de-noising quantization induced errors: 1) applying a Wiener filter, and 2) a maximum-likelihood conjugate-gradient-based optimization approach. We evaluate the correlation loss of quantization-and-up-channelization of simulated data and present quantitative results.

Section 2 describes PFBs in a straight-forward and intuitive manner. Section 3 presents a computationally efficient algorithm to invert the PFB. Section 4 addresses the effects of re-quantization noise on the inverted PFB. Section 5 compares two approaches for dealing with re-quantization effects.

2. POLYPHASE FILTER BANK

Given a broad-band signal, our goal is to split it up into a series of narrow-band channels. Picture an FM antenna, and we want a bank of filters where each filter outputs a single radio station. In the early days of radio astronomy, this was done with banks of analog mixers and filters. As the number of desired channels grows, analog solutions rapidly become expensive and unwieldy. Today, virtually all of this "channelization" is done digitally (Thompson *et al.* (2017)).

Naively, one might think that digitally sampling the incoming time series and then taking the Fast Fourier Transform (FFT) over successive frames would be all you would need to do, but in practice that turns out to be a horrible idea. While the FFT works perfectly for a discrete set of frequencies, in the sense that an incoming sine wave ends up in exactly one frequency channel, it performs badly for other frequencies. FFTs can only represent periodic functions, and so they will represent a sine wave with a non-integer number of cycles in the data as the sum of sine waves with integer number of cycles. The effects are 1) reducing power where it should be, and 2) spreading that power into other frequency channel, where it shouldn't be. Sources of radio-frequency interference (RFI) can be many, many orders of magnitude brighter than the celestial signals we're interested in, and if it leaks into other channels, RFI can completely blind a radio telescope (Oppenheim (1999)).

Of course, frequency leakage is nothing new, and people have been using window (aka taper) functions for nearly as long as they have been taking discrete Fourier transforms. A traditional window isn't a great option, because it reduces the frequency resolution of our FFT (see Figure 1). What we would really like is a way to window our data so that the frequency response is uniform within our channel, and zero outside it. We would especially like it if we could do this without increasing the length of the FFTs we have to take, because it gets

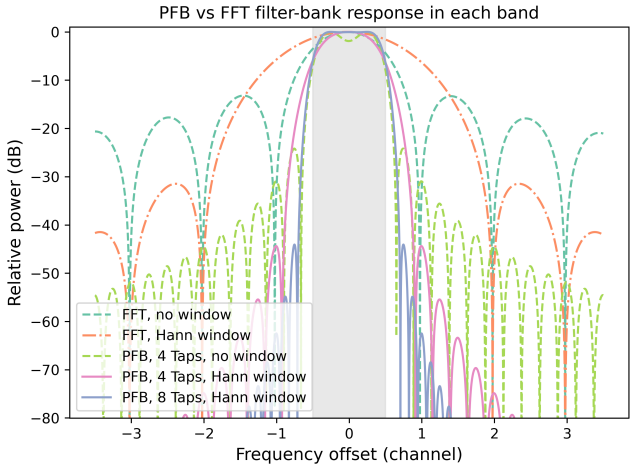

FIG. 1.— Comparison of frequency responses for Polyphase Filter Banks (PFB) and standard Fast Fourier Transform (FFT) filter banks. A PFB with a single tap is equivalent to an FFT filter bank. The plot displays the relative power (dB) as a function of frequency offset for five configurations: FFT without windowing (dashed, teal), FFT with a Hann window (dashed-dot, orange), 4-tap PFB without windowing (dashed, green), 4-tap PFB with a Hann window (solid, pink), and 8-tap PFB with a Hann window (solid, purple). The grey shaded area indicates the nominal pass band of the channel. The results demonstrate that Hann windowing significantly reduces side-lobe levels (spectral leakage), and increasing the number of PFB taps leads to a steeper roll-off and lower side-lobes.

very expensive in a real-time system. The polyphase filter-bank (PFB) is a way to achieve all these goals, at a relatively modest increase in the complexity of the signal processing.

To understand how the PFB works, picture a very long timestream that we want to split into frequency channels of finite bandwidth $\delta\nu$. Ideally, we want each channel to have a perfectly flat response within its band and absolutely zero response outside – a "boxcar" shape in the frequency domain. The convolution theorem tells us that to achieve this boxcar frequency response in the time domain, we must multiply the original time series by a Sinc function. However, a true Sinc function extends infinitely in time, making it impossible to implement. We must therefore approximate it using a Sinc function that is truncated and smoothed by a window function (Rabiner (1996)).

The quality of this approximation – how closely we approach the ideal boxcar shape, especially in terms of side-lobe suppression – depends critically on the length of this windowed Sinc filter. A longer filter yields a sharper, cleaner frequency response. Think of it like trying to identify musical notes in a recording. If you listen to just a brief moment (short time segment), it's hard to distinguish between nearby frequencies - a note and its sharp might sound similar. But if you listen for a longer period (longer time segment), the difference becomes clear. The filter's length relative to the final FFT-input size is defined by the number of taps ($n_{\rm tap}$). A PFB with $n_{\rm tap}$ taps uses an input segment $n_{\rm tap}$ times longer than a simple FFT approach to produce the same number of channels, effectively implementing a filter $n_{\rm tap}$ times longer. This increase in filter length is what provides the superior frequency isolation seen in Figure 1.

Applying this long filter directly and then performing a very long FFT would be computationally prohibitive for real-time systems. The "magic" of the PFB lies in a crucial mathematical insight: this process of long-filtering followed by decimation (keeping only the frequencies we want) can be re-ordered for immense computational savings. It turns out to be exactly equivalent to: 1) multiplying the long input segment by the windowed Sinc, 2) chopping this segment into $n_{\rm tap}$ pieces (or "frames"), 3) adding these pieces together point-by-point (the "polyphase" step), and 4) performing a single, short FFT on the resulting summed segment. This means we gain the frequency-sharpening benefits of a long filter while only needing the computational resources of a short FFT.

This powerful 'decimation-before-FFT' property is a fundamental aspect of Fourier transforms. For example, if we wish to decimate an FFT by a factor of four, it can be achieved by first folding and summing the time series,

$$\sum_{t=0}^{N-1} x[t]e^{-2\pi ikt/N} = \sum_{\tilde{t}=0}^{\frac{N}{4}-1} e^{-2\pi ik\tilde{t}/N} \sum_{s=0}^{3} x\left[\tilde{t}+\frac{sN}{4}\right], \quad (1)$$

with k, N divisible by four. This equation illustrates the core of the PFB's computational efficiency – transforming a long operation into a shorter one via pre-summation.

The entire PFB operation can be expressed concisely using linear algebra, capturing the relationship between the desired filtering and the efficient implementation. This equivalence is written as:

$$(\text{boxcar} * Fx)\downarrow_{n_{\rm tap}} = FSWx. \quad (2)$$

Here, the left side represents the desired operation: convolving FFT'd data (Fx) with a boxcar and decimating ($n_{\rm tap}$), while the right side represents the efficient PFB implementation: it involves applying the Sinc window weights (W), performing the polyphase stacking-and-summing (S), and finally executing the short Fast Fourier Transform (F). This formalism confirms that we achieve the frequency sharpness of an FFT $n_{\rm tap}$ times longer, using only a single FFT of the base length.

For example a PFB with 2048 frequency channels and four taps takes $4 \times 2048 = 8192$ digitized electric field samples, applies the Sinc window, performs the S operation (stacking into 4×2048 and summing to 1×2048), and then FFTs the resulting 2048-point vector. As Figure 1 vividly shows, this significantly improves both the flatness within the band and the attenuation of out-of-band signals compared to standard FFT techniques. Crucially, while increasing $n_{\rm tap}$ improves performance, the computational cost (dominated by the final FFT) remains fixed; only the input data buffer size increases.

The operations W, S, and F in Equation (2) correspond directly to the stages shown in the PFB processing diagram (see Figure 2), where W is the windowing stage, S is the vertical-sum stage, and F is the final FFT applied to each output frame.

Overlapping consecutive input (time-domain) data frames is necessary to balance the ratio of input/output data. A PFB with a ratio of in/out data equal to one is said to be *critically sampled*.[1] When consecutive time-

[1] Technically, if your time domain samples are real, only half as

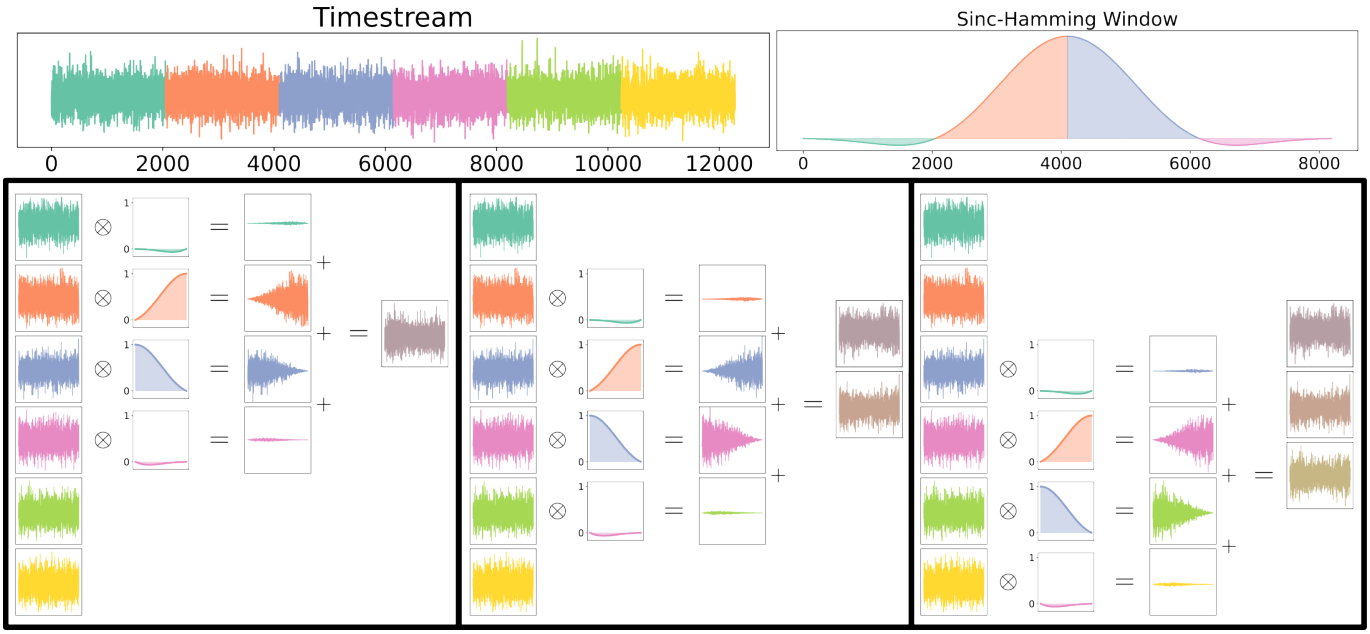

FIG. 2.— The 'Polyphase Structure' part of a four-tap PFB, (i.e. everything before the FFT, or the SW operators of Equation (2)). The filter window is a Sinc function pointwise-multiplied with a taper function; popular taper functions include the Hann or Hamming functions. The Sinc function, $\sin(x)/x$, is the Fourier transform of the Boxcar. The three lower panels demonstrate the operations performed by the PFB on three successive overlapping sets of four frames. A taper window is applied to successive time-stream chunks to suppress frequency leakage. The Sinc and taper windows, while conceptually separate, are combined into a single filter stage. After applying this window to a four-frame-long time-domain data segment, the segment's frames are split, stacked, and summed vertically. This operation is repeated every time the input samples shift out by one frame.

domain frames are overlapped so that output data outnumber input time-domain data, the PFB is said to be *oversampled*. Critically sampled PFBs are common because, compared to oversampled PFBs, they have a smaller digital footprint and are easier to implement on FPGAs.[2] Oversampled PFBs are lossless and there are known, spectrally well behaved invertion methods (Morrison *et al.* (2020)). In this paper we focus on critically sampled PFBs because they present an unsolved challenge relevant to many modern experiments.

In summary, for a PFB with a fixed number of frequency channels, increasing the number of taps enhances both the flatness of the response as well as out-of-band rejection *at marginal additional computational cost.*

3. INVERSE PFB

We present an inverse Polyphase Filter Bank (iPFB) algorithm, which efficiently reconstructs time-domain signals from discrete frequency channels. To understand this reconstruction process we must first delve deeper into the structure revealed by the PFB's operation as a streaming processor. Notably, this inverse operation can be performed even on incomplete spectral data: reconstructing from a subset of frequency channels will recover the original timestream with the missing channels filtered out, effectively allowing selective frequency extraction from the original signal.

3.1. *The PFB as Parallel Convolutions*

The PFB's true nature emerges when we view it not as a single calculation, but as a continuous streaming processor. It operates by repeatedly applying its core transformation to consecutive, overlapping segments of a long input time series. Imagine our input x flowing in. For each step, the PFB grabs a long segment (length $n_{\rm tap} \times n_{\rm chan}$), applies weights derived from the Sinc window, performs the crucial 'stack-and-sum' operation, and finally computes a Fast Fourier Transform (FFT) to get $n_{\rm chan}$ output values. The input window then slides forward by $n_{\rm chan}$ samples, and the entire process repeats.

Let's consider what this means for individual input samples. The 'stack-and-sum' operation acts as a kind of demultiplexer. It effectively splits the input time series x into $n_{\rm chan}$ interleaved sub-streams. The first sub-stream consists of samples $x[0], x[n_{\rm chan}], x[2n_{\rm chan}], \ldots$; the second consists of $x[1], x[n_{\rm chan}+1], x[2n_{\rm chan}+1], \ldots$, and so on.

The windowing step applies its Sinc-derived weights before this 'stack-and-sum' step. When we trace the journey of these individual sub-streams through the repeated PFB steps, a remarkable structure emerges: each of the $n_{\rm chan}$ sub-streams is independently convolved with a short, distinct filter kernel. The length of each kernels is precisely $n_{\rm tap}$, and their coefficient values are drawn directly from the Sinc window weights. In essence, these initial filtering stages—the windowing and polyphase summation—when applied as a streaming process, transform the single input time series into $n_{\rm chan}$ parallel, independently convolved data streams.

The final FFT step takes these $n_{\rm chan}$ convolved sub-

many output samples are required because you can apply a real-to-complex FFT as negative frequencies are the complex conjugates of positive samples. However, this is a technicality; the deeper point is that information is theoretically conserved, but only just, in critically sampled PFBs. As we will see, however, even that statement is only true mathematically. In reality, finite bit-depth and requantization makes the PFB degenerate.

[2] You could also imagine an undersampled PFB, but that is an intrinsically lossy operation, and therefore of limited use. We don't treat those here.

streams at each time step and performs a Fourier transform across them. This mixes the information from the parallel streams to produce the final $n_{\rm chan}$ frequency-channel outputs. Therefore, to invert the PFB, we must first undo this mixing with an inverse FFT, and then—crucially—we must deconvolve each of these $n_{\rm chan}$ parallel sub-streams to recover the original interleaved data.

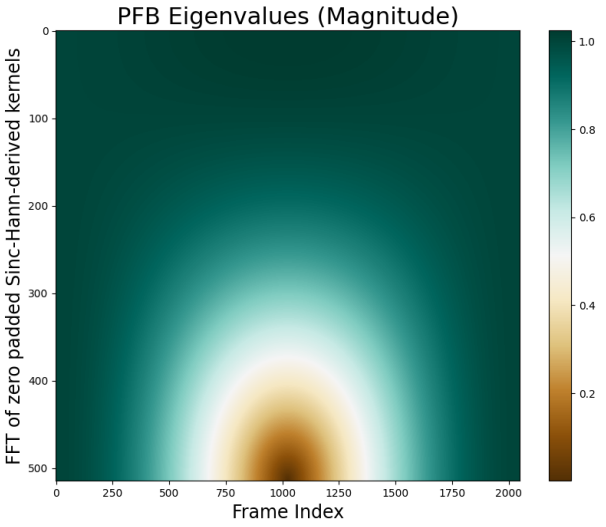

FIG. 3.— Heatmap of the magnitude of the PFB's eigenvalues. The eigenvalues exhibit similar properties irrespective of the number of taps. The PFB, when viewed as a transformation of a long time series into another long time series, is merely the composition of many linear operators, which means we can represent its aggregate effect as a large matrix. The eigenvalues of this PFB matrix are the FFTs of each of the zero-padded polyphase kernels (each kernel is a vertical slice of the stacked Sinc-Hann window segments shown in Figure 2). There are as many distinct kernels as there are samples in one time-domain frame. It is natural to lay out the eigenvalues on a two domensional plot as it is natural to lay out the output baseband of a PFB onto a two dimensional spectrum or waterfall plot. This heatmap shows that the PFB operator has a 'point of degeneracy' and a surrounding area where the eigenvalues approach zero.

3.2. *Inverting Convolutions via Deconvolution*

Fortunately, inverting a convolution (deconvolution) is a well-understood. Convolutions are simpler to negotiate with in the Fourier domain. A convolution operation between some data, z, and a kernel ker, written as:

$$y = \ker * z. \tag{3}$$

is transformed by the Fourier convolution theorem into a simple pointwise multiplication in the Fourier domain, $F\{y\} = F\{ker\} \cdot F\{z\}$.

This immediately tells us how to solve for the original data z. By rearranging the equation in the Fourier domain, we get:

$$F\{z\} = \frac{F\{y\}}{F\{\ker\}}. \tag{4}$$

This means deconvolution simply amounts to division in the Fourier domain. We can then transform back to the time domain to recover z. Alternatively, we can express this entire deconvolution process as another convolution in the original domain:

$$z = F^{\dagger}\left\{\frac{1}{F\{\ker\}}\right\} * y. \tag{5}$$

This highlights that deconvolution is, itself, a convolution, but with an inverse kernel defined by the reciprocal of the original kernel's Fourier transform.

3.3. *PFB Invertibility and Eigenvalues*

The PFB, viewed as an end-to-end operator transforming the input time series into the channelized output, is a linear system. The key to understanding its invertibility lies in its eigenvalues. For a system like this, an eigenvalue represents the gain or scaling factor that the system applies to a specific input mode or frequency component. If an eigenvalue is zero, the system completely filters out that mode – the information is lost, and the system is not perfectly invertible. If an eigenvalue is very small, that mode is heavily attenuated, and attempting to boost it back up during inversion (as implied by the division in Equation (4)) will drastically amplify noise.

Given our understanding of the PFB as $n_{\rm chan}$ parallel convolutions followed by an FFT, we can deduce its eigenvalues. The fundamental modes of the system are tied to the Fourier transforms of those $n_{\rm chan}$ small filter kernels (each of length $n_{\rm tap}$). The magnitude of the Fourier transform of a kernel tells us how much gain it applies at each frequency. Therefore, the eigenvalues of the PFB operator correspond to the Fourier transforms of these $n_{\rm chan}$ distinct, $n_{\rm tap}$-length kernels derived from the Sinc window. Crucially, eigenvalues (Fig 3) vanish at a specific point and are small in the surrounding region. This point of degeneracy is an unavoidable property of PFBs.

This degeneracy is the root cause of the difficulty in inverting the PFB. When we attempt deconvolution using Equation (4), we encounter division by zero (or near-zero) at these degenerate points. This confirms that perfect inversion is mathematically impossible, and practical inversion will be limited by noise amplification in these regions. Reconstructing the original time series requires navigating these 'deaf spots' in the PFB's response.

4. REQUANTIZATION EFFECTS

The inverse-PFB (iPFB) method presented in the previous section is the mathematically correct operation. However, channelized data are typically re-quantized to a few bits, and this causes corruption in the time domain. In this section we characterize this corruption quantitatively.

We model re-quantization as uniform, additive, stationary white-noise in the channelized domain. Propagating uniformly sampled noise through the iPFB, we recover the time-domain induced noise profile. The result is that uniform noise in the channelized domain induces spurious noise spikes in the time domain. This transformation occurs because the iPFB greatly amplifies quantization noise on a specific subset of time-domain samples. The noise profile repeats with a period of one frame; i.e. if the input data frames are 2048 samples wide, the noise spikes repeat over a period of 2048 samples. Figure 4 shows the calculated 'gain' applied to the noise on each frame.

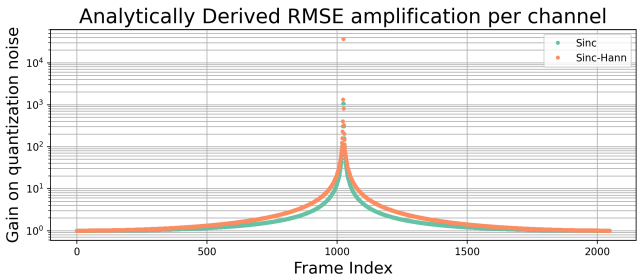

FIG. 4.— The expected gain on the RMS error over the span of a single frame of time-domain data when the iPFB is applied without filtering (Equation (5)). The gain is low at the extremities and large in the middle. This gain profile is identical for each time-domain frame and is derived and explained in detail in Appendix 6.6. Figure 6 shows the noise profile on simulated data.

Numerical experiments with simulated data corroborate results obtained from mathematical analysis of the iPFB-induced noise gain (see Figure 6).

Spurious time-domain noise spikes degrade upchannelized signals by raising the noise floor when samples are processed through longer FFTs. As we will see in the next section, a simple solution proves effective: downweighting noisy recovered time-domain samples with a basic filter.

5. MITIGATING QUANTIZATION EFFECTS

We quantitatively compare two noise mitigation methods: (a) using a Wiener filter and (b) recovering the most likely signal by doing a Chi-squared optimization with an additional time-domain prior.

5.1. *Wiener Filter*

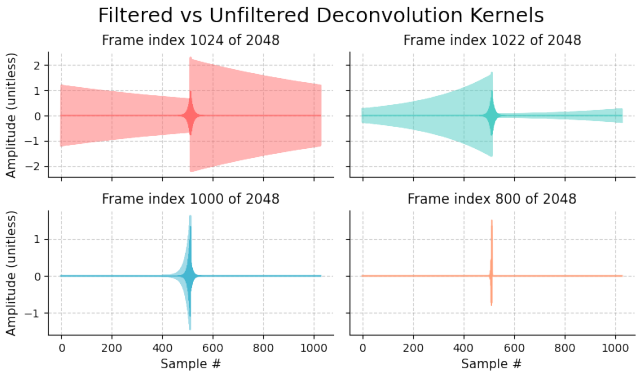

FIG. 5.— IPFB deconvolution kernels with and without Wiener filtering. Transparent curves show unfiltered kernels (Equation (5)); opaque curves show Wiener-filtered kernels (Equation (6)). Each panel represents a different frame index relative to the frame center. Near the frame center, the deconvolution operator becomes singular, creating severe ill-conditioning where small signal differences are computed from many large, noisy values. The central kernel (top left, red) and near-central kernel (top right, teal) exhibit the worst conditioning. Wiener filtering localizes the kernel energy, improving numerical stability. Conditioning improves at 2% from the center (bottom left, blue), though Wiener filtering still provides substantial modification. At 20% from center (bottom right, orange), kernels are well-conditioned and Wiener filtering has minimal effect.

The issue of noise amplification can be mitigated by Wiener filtering the inverse-PFB. The Wiener filter weighs Fourier domain data inversely proportionally to how much they amplify noise. Applying such a filter comes at negligible computational cost.

Wiener Filtering Equation (5) yields

$$z := F^{\dagger}\left\{\frac{f_{\phi}\left(\left|F\{\ker\}^{-1}\right|\right)}{F\{\ker\}}\right\} * y, \tag{6}$$

where the Wiener filter weights take the form

$$f_{\phi}(\beta) = \frac{\beta^{-2}(1+\phi^2)}{\beta^{-2}+\phi^2}. \tag{7}$$

The dimensionless threshold parameter, ϕ, can be chosen to optimize quantization-SNR. However, SNR is not very sensitive to this parameter and for a first pass you can set it to $\phi = 0.1$. See Appendix 6.8 on optimizing the threshold parameter.

Figure 5 illustrates how Wiener filtering modifies the convolution kernels. Figure 6 shows the effects of Wiener filtering on the time-domain quantization RMSE.

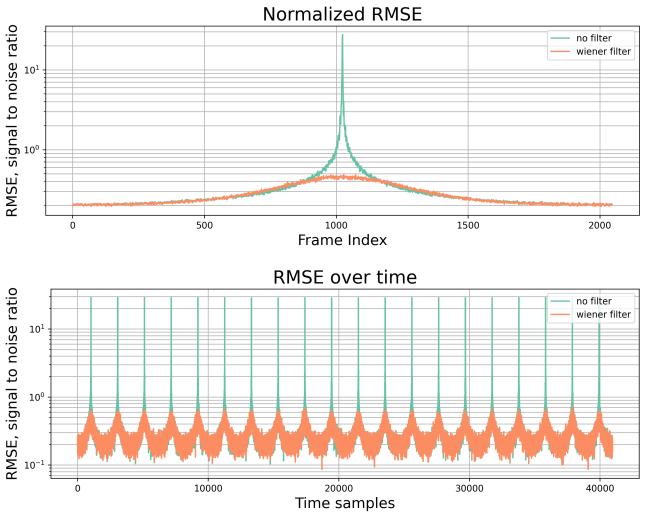

FIG. 6.— Numerically simulated comparison of the Root Mean Squared Error (RMSE) due to requantization with and without Wiener filtering. Above, many frames worth of simulated data is averaged for better statistics, below, many frames worth of data are displayed to show the translational symmetry of this RMSE pattern. The high peaks, which correspond with the noise amplification of Figure 4, are a numerical estimate of the RMSE. The orange lines are the computed RMSE after applying the Wiener filter.

5.2. *Chi-Squared Optimization*

An alternative approach to filtering is to caste the task of recovering time-stream samples as a maximum likelihood optimization problem. While this approach is computationally more intensive and, ultimately, does not significantly beat Wiener filtering, it offers a different and illuminating perspective. We use the conjugate gradient method to approximately solve this optimization problem in few iterations (Shewchuk (1994)).

Given some data d, a linear model A, model parameters x, and a covariant, positive-definite noise matrix N, we write chi-squared,

$$\chi^2 = (d - Ax)^T N^{-1}(d - Ax). \tag{8}$$

Minimizing chi-squared yields the maximum probability model parameters. In our case d are quantized, channelized data, x are the unknown time-domain data, A is the PFB, and we can assume that the quantization noise matrix N, is proportional to the identity matrix. Starting with Wiener-filtered iPFB data as the initial guess, we minimize chi-squared using the conjugate-gradient method, which excels at efficiently solving large second order systems of equations (Shewchuk (1994)).[3]

If time-domain samples are accessible, (for instance, if we program our digital logic to route a subset of these samples), we can incorporate them as an additional term in the chi-squared optimization:

$$\chi^2 = (d - Ax)^T N^{-1}(d - Ax) + (p - x)^T Q^{-1}(p - x). \quad (9)$$

Here, the model parameters are time domain samples x, the linear model is the identity operator, the (diagonal) noise matrix is captured by Q, and prior-known time-domain samples are captured in the vector p.

The interested reader can find an in depth description of what we tried, including a discussion of how we optimized the selection of time-domain samples for a bandwidth-constrained FPGA[4], in Appendix 6.9.

5.3. *Quantization Mitigation Results*

To quantify the success of the Wiener filter and the maximum likelihood optimization, we sum correlated products of simulated data (PFB'd, quantized, inverted, filtered) from two antennas (i.e. time-streams with identical signals but different noise realizations). The sample mean and standard deviation of sums of correlated products give us a signal-to-noise ratio of recovered time-streams that we can compare to floating-point-accurate channelization. This metric is meaningful because interferometers point and see by correlating data from pairs of antennas, and averaging these correlated products is how we beat noise down to make sky maps. Furthermore, this metric is practical because it allows us to compare apples to apples: Wiener filtering and conjugate gradient optimization induce a slight gain on the data, but correlated products are calibrated against sources of known intensity.

	Sample μ	Sample σ	SNR	ENOB
FP	52178 ± 11	1369.4 ± 7.6	38.10 ± 0.21	
No filter	50829 ± 109	13978.5 ± 77	3.64 ± 0.02	2.01
Wiener	49850 ± 10	1334.8 ± 7.4	37.35 ± 0.21	2.77
1% prior	48112 ± 10	1283.7 ± 7.1	37.48 ± 0.21	2.82
3% prior	48245 ± 10	1285.1 ± 7.1	37.54 ± 0.21	2.90
5% prior	48352 ± 10	1286.2 ± 7.1	37.59 ± 0.21	2.90
10% prior	48442 ± 10	1288.5 ± 7.1	37.60 ± 0.21	2.93

TABLE 1
Results of quantization noise mitigation strategies.

Given the sample mean and sample standard deviation of correlated products of up-channelized data from two simulated antennas, the SNR is simply μ^2/σ^2. The simulated data samples two Gaussian random variables, one representing the signal, which is common to both simulated antennas, and two representing two different antenna-specific noise realizations. Each correlated input time-stream uses 2^{25} data-points representing about 1/4 of a second when sampling at 250 MSPS. The standard deviation of the time-domain noise was chosen to have 10x the standard deviation of the signal to mimic realistic conditions. The data was channelized, quantized, then up-channelized using filtering methods discussed above. The first row, FP, represents the theoretical information limit, employing floating-point accurate channelization in the up-channelized domain. The 'No filter' row is the result of up-channelizing using the iPFB without any filter. The 'Wiener' row is the result of applying a Wiener filter on the iPFB. The lower four rows represent data optimized by conjugate gradient descent using priors, the percentage indicates what proportion of time-domain samples were used in the priors. A chi-squared optimization does not outperform a simple Wiener filter without an additional prior. To beat the noise down, up-channelizing and correlating was repeated over 16384 times. The initial channelization frame-size was 2048 samples wide and the up-channelization factor was 32x for a total of 65536 channels. The channelized signals are quantized to 4+4 bits. The quantization interval was chosen to be 0.353 of the RMS of the channelized signals. The pseudo random number generator seeds and code are open sourced for replicability.

6. APPENDIX

6.1. *FFT Aliasing*

In this appendix, we show that a coarse-resolution FFT can be carried out by aliasing pieces of the timestream onto themselves. Mathematically the FFT of $x[t]$ is defined to be

$$\hat{x}[k] = \sum_{t=0}^{N-1} x[t] \exp(-2\pi i k x/N)$$

where N is the total number of data points, t goes from 0 to $N-1$, and $x[t]$ are the data we are Fourier transforming. We evaluate this sum for every integer k from 0 to $N-1$. If we instead want a subset of the frequencies, $k = k' n_{tap}$ where k' is an integer ranging from 0 to $N/n_{tap} - 1$, we can rewrite the FFT as:

$$\sum_{t=0}^{N-1} x[t] \exp(-2\pi i k' n_{tap} t/N). \quad (10)$$

We can always break apart the sum into pieces, because every $n_{\rm tap}^{\rm th}$ mode is $N/n_{\rm tap}$ periodic, then add those pieces together:

$$\sum_{t=0}^{\frac{N}{n_{\rm tap}}-1} \exp\left(\frac{-2\pi i k' t}{N/n_{\rm tap}}\right) \sum_{r=0}^{n_{\rm tap}-1} x\left[t + r\frac{N}{n_{\rm tap}}\right]. \quad (11)$$

We carry out the sum on time domain data before taking the shorter FFT.

[3] When running gradient descent it's practical to inverse-FFT the quantized, channelized data and use that as d; the noise matrix conveniently remains proportional to the identity under this transformation.

[4] Field Programable Gate Arrays are often used in astronomical interferometric readout systems at this stage of the signal processing.

An alternative, possibly more intuitive, way of thinking about this is to notice that if we want the $k = 0, 4, 8...$ frequencies (assuming $n_tap = 4$), then we're multiplying our full timestream by sine waves with 0,4,8... periods and then summing. Each of these sine waves is four copies of it's first quarter, which means we can sum first and then multiply by one quarter of the sine wave.

6.2. *PFB convolution derivation*

Binning and averaging the output of an FFT is the same thing as down-sampling the convolution of a boxcar with FFT'd data. The convolution theorem enables us to turn convolutions into Fourier domain point-wise multiplication.

$$(\text{boxcar} * Fx) \downarrow_{n_{\text{tap}}} = (F[\text{sinc} \cdot x]) \downarrow_{n_{\text{tap}}} = FSW \quad (12)$$

TODO...

6.3. *Forward PFB Code Snippet*

A python code snippet to PFB a timestream, x. This code is not in any way machine-optimized.

```
import numpy as np
from numpy.fft import rfft
NTAP = 4
NFRAME = 4096
N = 1000
x = np.random.randn(NFRAME * N)
sinc = np.sinc(NFRAME * NTAP)
hann = np.hanning(NFRAME * NTAP)
window = sinc * hann
pfb_data = []
for i in range(len(x) / NFRAME - NTAP + 1):
    x_segment = x[i * NFRAME : (i+NTAP) * NFRAME]
    x_windowed = x_segment * window
    x_stacked = x_windowed.reshape((NTAP,-1))
    x_stacked_summed = np.sum(x_stacked, axis=0)
    x_pfbd = rfft(x_stacked_summed)
    pfb_data.append(x_pfbd)
```

Or, equivalently...

```
kernels = window.reshape((NTAP, NFRAME))
x_stacked_summed = np.zeros((N - NTAP + 1,NFRAME))
for ix in range(NFRAME):
    x_stacked_summed[:,ix] = np.convolve(
        kernels[:,ix],
        x[ix::NFRAME],
        mode='valid'
    )
pfb_data = rfft(x_stacked_summed, axis=1)
```

Or, equivalently... [in final version make sure you get conjugate and rolling correct, also in ipfb snippet]

```
pad = np.zeros(N-NTAP, NFRAME) # zero padding
mat_eig = rfft(np.vstack(kernels, pad), axis=0)
x_reshaped = x.reshape((N, NFRAME))
pfb_data = irfft(
    mat_eig * rfft(x_reshaped, axis=1),
    axis=1
)
```

6.4. *Inverse PFB Code Snippet*

A python code snippet to undo PFB'd data:

```
import numpy as np
from numpy.fft import rfft,irfft
xout = irfft(spec, axis=1)
xout = rfft(xout, axis=0)
xout /= mat_eig
xout = irfft(xout, axis=0).flatten()
```

Above, `mat_eig` is a precomputed eigenvalue matrix and `spec` is a 2D array of PFB'd data.

[cite numpy]

6.5. *Inverse PFB Time Complexity*

The computational time complexity of Equations (4) and (5), and their Wiener filtered counterparts (**??**) and (**??**) as a function of the number of samples, n, are worth examining. Equation (4) is limited by FFTs, which grow in time complexity like $O(n \log n)$, whereas you might think that Equation (5) grows like $O(n^2)$. However, the convolution kernels $F^\dagger\{1/F\{\text{ker}\}\}$ are compactly supported, so they are actually FIR filters, with linear, $O(n)$, time complexity. Unlike FFTs, which demand significant memory resources to handle large datasets, FIR filters can operate on smaller, sequential chunks of data, making them more practical for buffered processing. Moreover, each (of the 2048) decoupled deconvolution is itself easily parallelizable.

6.6. *Inverse PFB Quantization Error Gain*

Figure 4 shows the amplification of the expected root-mean squared error. Quantization noise can be modeled as a random variable sampling from a uniform distribution in the PFB domain, and as sampling from a normal, complex distribution in the eigen basis (or "polyphase domain"[5]). Quantization noise is amplified, by the reciprocal of PFB's eigenvalues (see Figure 3) in the polyphase domain. A conjugate Fourier transform is applied to each column of data to transform the data back into the time domain; this operation averages the error amplification over that column. The small eigenvalues only effect central columns, which is why the quantization noise is spurious in the time domain; it's also why the noise spike profile is periodic, with a period of one frame. Therefore, Figure 4 is merely a column-wise average of the absolute value of the reciprocal of the eigenvalues shown in Figure 3.

6.7. *Alternatives to Sinc-Hann PFB*

Given the near-zero eigenvalues of the PFB, we considered modifying the weights of the Sinc-Hann filter–without changing the polyphase structure (Figure **??**) of the PFB–to see if we could find something without small eigenvalues. After sinking some time into this problem by doing gradient descent on window weights with a variety of loss functions, and other experiments unworthy of description, we concluded that such an effort was futile. To prevent others from trying the same thing, here is a

[5] The polyphase domain the data is in after the polyphase structure of the PFB has been applied, but before the Fourier transform; see figure **??**.

short proof that any sensible PFB window weights will have zero eigenvalues for an arbitrary number of taps. In other words, it's not possible to craft a critically sampled PFB filter that both suppresses spectral leakage and isn't degenerate.

First we define any sensible PFB window array, (W_k), to be (a) symmetric about it's center, so that $W_k = W_{N-k}$, and (b) to go to zero on the boundary, so that $W_1 = W_N = 0$. Note that this is quite a loose requirement on the window because it has to suppress spectral leakage. Now, suppose the number of taps, $n_{\rm tap}$, is even, then we claim that there will be zero eigenvalues in the central column in the polyphase domain. The central column eigenvalues are the Fourier transform of a zero-padded array whose non-zero elements are symmetric. The middle frequency of this Fourier transform is therefore the dot-product of a symmetric vector with the alternating sequence (1, -1, 1, -1, ...). Because there are an even number of taps and therefore non-zero elements, the symmetric terms recieve inverted signs and cancel each other, the total sum, proving that there is a zero eigenvalue. In practice, there will be many zero eigenvalues around this area. In the case of $n_{\rm tap}$ odd, the only adjustment to the proof is that we pick the zero'th (or the final column) column instead of the middle column. The first term in the FT is $W_0 = 0$, the other terms are even and symmetric, and the previous argument applies. QED.

6.8. *Optimal Wiener Threshold*

The Wiener filter,

$$f_\phi(\beta) = \frac{\beta^{-2}(1+\phi^2)}{\beta^{-2}+\phi^2}, \tag{13}$$

(equivalent to Equation (7) with) has a threshold parameter, $0 < \phi << 1$, which deserves some attention. If the Wiener threshold is too small, it won't effectively downweigh troublesome eigenvalues[6], on the other hand, if it's too large, it will bias clean values too. To determine the optimal threshold parameter we assess the expected mean squared error in our data. We can assume the signal (sky plus instrument noise but not quantization noise) samples a complex Gaussian distribution of standard deviation σ_s in the polyphase domain. In this domain, the quantization noise approximately samples a Gaussian whose standard deviation we denote σ_n. Then, the mean-squared error of Wiener filtered data (still in the polyphase domain), has two components,

$$\mathrm{MSE}(y) = \frac{\sigma_n^2}{\beta^2}\left(\frac{1+\phi^2}{\phi^2+\beta^{-2}}\right)^2 + \sigma_s^2\phi^4\left(\frac{1-\beta^{-2}}{\phi^2+\beta^{-2}}\right)^2. \tag{14}$$

Here β represents the absolute reciprocal of PFB eigenvalues[7] (i.e. the reciprocal of values in Figure 3); β ranges from 1 to infinity. The first component relates to quantization noise. For $\beta \sim 1$, it grows like β^2, as it would without the Wiener filter, but then it tapers off thanks to the filter. The first term shrinks with ϕ. The second term relates to how much the filter distorts real data, and it grows with ϕ. Finding the optimal threshold parameter amounts to no more than optimizing MSE in the most convenient way possible. One way to do this is to minimize the sum of the expected errors over all eigenvalues, or values of β. The optimal threshold parameter will, of course, depend on the signal-to-quantization-noise ratio σ_s/σ_n.

[6] Indeed, in the limit of $\phi \to 0$, Equation (13) reduces to the constant function $f_0(\beta) = 1$, and has no effect. By setting $\phi = 0$, one can easily obtain the expression for the MSE of an unfiltered quantized iPFB from Equation (14).

[7] Equivalently, the absolute value of eigenvalues of the iPFB.

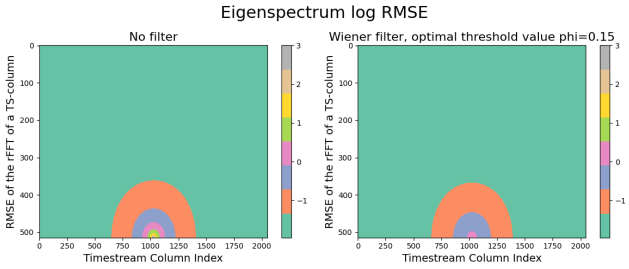

FIG. 7.— Eigenvalues of the iPFB with verses without a Wiener filter. TODO: write more caption here...

6.9. *Time domain prior selection*

TODO: write this appendix.

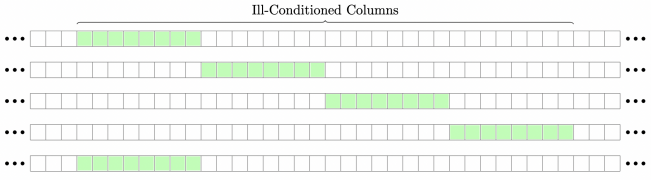

FIG. 8.— Selection of prior...MORE TODO... The boxes represent our time stream from left to right. The data points which we are explicitly salvaging are highlighted in green. Although this is an image of the raw time stream and not the PFB'd signal, we reshape it to lay it out as a stack of rows of length $n_{\rm frame}$ for a more geometrically intuitive understanding.

6.10. *Conjugate Gradient Descent*

TODO: Write this section

6.11. *Optimal 15-level Requantization Delta and QSNR*

I wrote this into a chapter of thesis. Can copy paste if think it's a good idea.

6.12. *Hamming/Hanning/Hann disambiguation*

The Hamming window, $0.54 + 0.46\cos(2\pi n/N)$, for $-N/2 \le n \le N/2$, has a frequency response with very low immediate side-lobes (below -40 dB), but falls slowly thereafter. The Hann window, $\sin^2(\pi n/N)$, for $0 \le n \le N$, has a frequency response which falls off much faster asymptotically but has slightly worse immediate side-lobes than the Hamming window (only about -30 dB). The Hamming window is named after the famous computer scientist and mathematician Richard Hamming, the Hann window is named after Julius von Hann, they bear no relation. The Numpy package confusingly misnames the Hann window "hanning".

REFERENCES

H. C. Chiang, T. Dyson, E. Egan, S. Eyono, N. Ghazi, J. Hickish, J. M. Jauregui-Garcia, V. Manukha, T. Menard, T. Moso, J. Peterson, L. Philip, J. L. Sievers, and S. Tartakovsky, "The array of long baseline antennas for taking radio observations from the sub-antarctic," (2020), arXiv:2008.12208 [astro-ph.IM].

I. S. Morrison, J. D. Bunton, W. van Straten, A. Deller, and A. Jameson, Journal of Astronomical Instrumentation **09**, 2050004 (2020), https://doi.org/10.1142/S225117172050004X.

A. R. Thompson, J. M. Moran, and G. W. Swenson, *Interferometry and synthesis in radio astronomy* (Springer Nature, 2017).

A. V. Oppenheim, *Discrete-time signal processing* (Pearson Education India, 1999).

L. R. Rabiner, *Multirate digital signal processing* (Prentice Hall PTR, 1996).

J. R. Shewchuk, "An introduction to the conjugate gradient method without the agonizing pain," (1994)