Draft version September 13, 2022
Typeset using LaTeX **twocolumn** style in AASTeX63

Large-scale power loss in ground-based CMB maps

Sigurd Naess[1]

[1] *Institute of Theoretical Astrophysics, University of Oslo, Norway*

ABSTRACT

1. INTRODUCTION

SKN: Should cite Naess (2019). Should say something about discussion about this in the literature being limited to high-contrast regions, and cite XGLS etc.

CMB telescopes observe the sky by scanning their detectors across it while continuously reading off a series of samples from the detectors. Typically the signal-to-noise ratio of each sample is small, but by combining a large number of samples with knowledge of which direction the telescope was pointing at any time, it's possible to reconstruct an image of the sky. There are several ways of doing this, with the most common being maximum-likelihood, filter+bin and destriping. These all start by modelling the telescope data as

$$d = Pm + n \tag{1}$$

where d is the set of samples read off from the detectors (often called the time-ordered data), m is the set of pixels of the sky image we want to reconstruct, n is the noise in each sample (usually with significant correlations), and P is a pointing matrix that encodes how each sample responds to the pixels in the image.

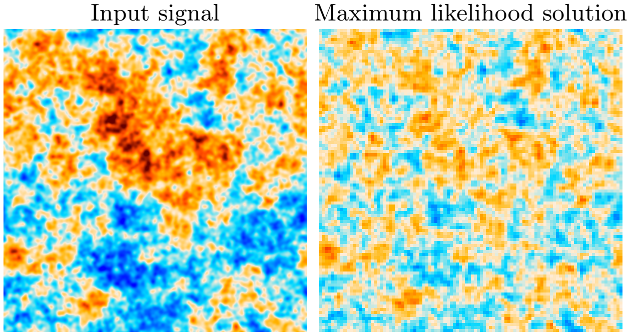

Figure 1. Preview of the model error bias I will discuss in the following sections. Despite the standard expectations that maximum-likelihood mapmaking is optimal and unbiased, the maximum-likelihood solution (**right**) for a simple toy example is strongly power-deficient on large scales compared to the input signal (**left**). As we shall see this bias is not unique to maximum-likelihood methods, and can be triggered by several subtle types of model error.

Given the model, it's possible to either directly solve for an unbiased map (as in maximum likelihood mapmaking or destriping), or to measure and correct for the bias in a biased estimator (as in filter+bin mapmaking). However, this fails when the model does not actually describe the data, and this turns out to be the norm rather than the exception. The goal of this paper is to demonstrate the unintuitive and surprisingly large (O(1)) effect even seemingly inconsequential model errors can have when imaging the CMB. The full scope of these errors appears to be largely unappreciated, and I fear that most ground-based CMB analyses so far suffer from uncorrected bias at low multipoles in total intensity due to these effects. The bias usually manifests as a power deficit at large scales, as illustrated in figure 1.

2. SUBPIXEL ERRORS

Subpixel errors may be both the most common and most important class of model errors in CMB mapmaking. For efficiency reasons P is always[1] chosen to use simple nearest-neighbor interpolation, where the value of a sample is simply given by the value of the pixel nearest to it. This means that P can be implemented by simply reading off one pixel value per sample, and its transpose P^T consists of simply summing the value of the samples that hit each pixel. However, this comes at the cost of there being a discontinuous jump in values as one scans from one pixel to the next, as illustrated in figure 2. Hence, the closest the model can get to a smooth curve is a staircase-like funciton that hugs it, leaving a residual full of discontinuous jumps (the blue curve in the figure).

Discontinuous residuals are not necessarily problematic. The trouble arises when this is coupled with a likelihood[2] where some modes have much less weight than others. Ground-based CMB telescopes suffer from atmospheric emission that acts as a large source of noise on large

[1] I am not aware of any published CMB analysis that has done something else.

[2] The equivalent for filter+bin is a filter that impacts some modes more than others (which is the whole point of a filter), and for destriping it's a baseline amplitude prior with those properties.

scales. This leads to time-ordered data noise power spectra similar to the one sketched in figure 3, with a white noise floor at short timescales (high frequency) transitioning to a steep rise of several orders of magnitude as one moves to longer timescales (low frequency). In this case long-wavelength modes have orders of magnitude lower weight in the likelihood than short-wavelength modes. Put another way, they are orders of magnitude *cheaper to sacrifice* when the model can't fully fit the data.

2.1. *1D toy example*

To illustrate the interaction between a nearest neighbor pointing matrix's subpixel errors and a likelihood where large scales have low weight, let us consider a simple 1D case where 100 samples scan uniformly across 10 pixels:

```
npix  = 10
nsamp = 100
pix   = np.arange(nsamp).astype(float)*npix/
    nsamp
```

A standard nearest-neighbor pointing matrix for this looks like:

```
P = np.zeros((nsamp,npix))
for i, p in enumerate(pix):
  P[i,int(np.round(pix[i]))%npix] = 1
```

We build the inverse noise matrix/filter/baseline-prior F by projecting an inverse fourier spectrum into pixel space:

```
freq   = np.fft.rfftfreq(nsamp)
inv_ps = 1/(1+(np.maximum(freq,freq[1]/2)/0.03)
    **-3.5)
F = np.zeros((nsamp,nsamp))
I = np.eye(nsamp)
for i in range(nsamp):
  F[:,i] = np.fft.irfft(inv_ps*np.fft.rfft(I[i
      ]), n=nsamp)
```

The signal itself consists of just a long-wavelength sine wave:

```
signal = np.sin(2*np.pi*pix/npix)
```

With this in place, we can now define our map estimators.

2.1.1. *Binning*

The *binned map* is simply the mean value of the samples in each pixel, with no weighting:

```
map_binned = np.linalg.solve((P.T.dot(P)), P.T.
    dot(signal))
```

2.1.2. *Maximum-likelihood*

The maximum-likelihood solution of equation 1 for the sky image m is

$$\hat{m} = (P^T N^{-1} P)^{-1} P^T N^{-1} d \tag{2}$$

where N is the covariance matrix of the noise n. In our toy example, $N^{-1} = F$, so the *maximum-likelihood map* is

```
map_ml = np.linalg.solve((P.T.dot(F).dot(P)),P.
    T.dot(F.dot(signal)))
```

2.1.3. *Filter+bin*

As the same suggests, filter+bin consists of filtering the time-ordered data, and then making a binned map. We'll use F as our filter, so the *filter+bin map* is

```
map_fb = np.linalg.solve(P.T.dot(P), P.T.dot(F)
    .dot(signal))
```

The filter+bin map is biased by design, so to interpret or debias it one needs to characterize this bias. There are two common approaches to doing this: Observation matrix and simulations.

Observation matrix—The observation matrix approach (cite BICEP here) recognizes that the whole chain of operations observe, filter, map together make up a linear system, and can therefore be represented as a matrix, called the *observation matrix*. Building this matrix is heavy, but doable for some surveys. Under the standard assumption that the observation step is given by equation 1, the observation matrix is given by

```
obsmat = np.linalg.inv(P.T.dot(P)).dot(P.T.dot(
    F).dot(P))
```

and using it, we can define a debiased filter+bin map

```
map_fb_deobs = np.linalg.solve(obsmat, map_fb)
```

Simulations—Alternatively, and more commonly, one can characterize the bias by simulating the observation of a set of random skies, passing them through the filter+bin process, and comparing the properties of the input and output images. The standard way of doing this is by assuming that equation 1 describes the observation process, and that the bias can be described by a *transfer function*: a simple independent scaling of each fourier mode. Under these assumptions, we can measure and correct the bias as follows.

```
nsim = 1000
sim_ips = np.zeros(npix//2+1)
sim_ops = np.zeros(npix//2+1)
for i in range(nsim):
  sim_imap = np.random.standard_normal(npix)
  sim_omap = np.linalg.solve(P.T.dot(P), P.T.
      dot(F).dot(P).dot(sim_imap))
  sim_ips += np.abs(np.fft.rfft(sim_imap))**2
  sim_ops += np.abs(np.fft.rfft(sim_omap))**2
tf = (sim_ops/sim_ips)**0.5
map_fb_detrans = np.fft.irfft(np.fft.rfft(
    map_fb)/tf, n=npix)
```

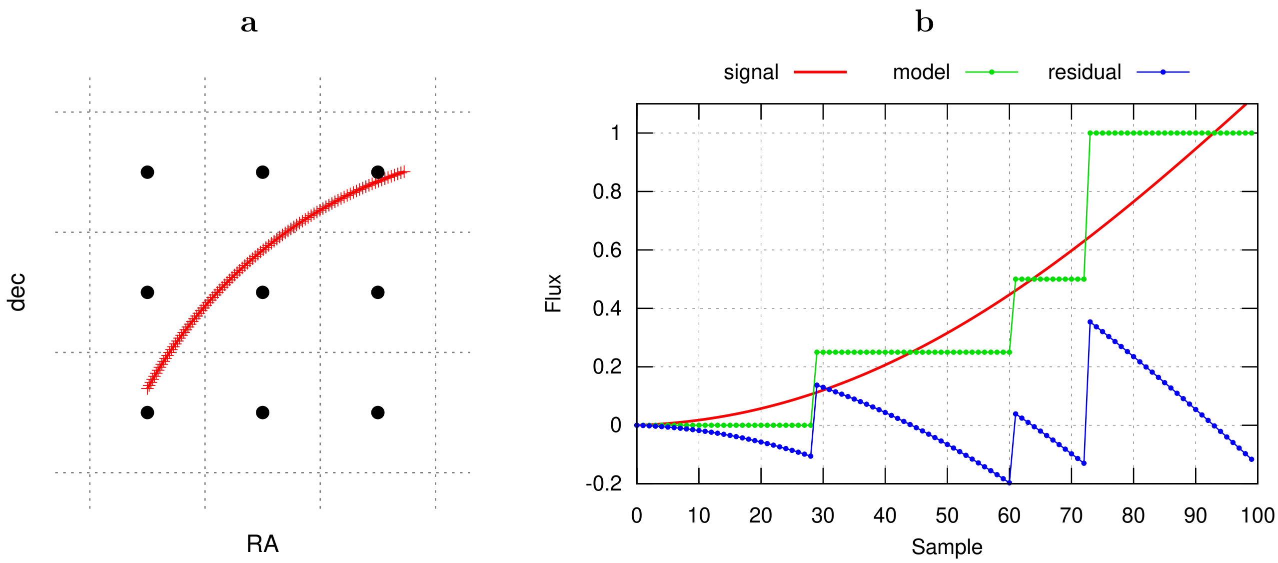

Figure 2. **a**: Example path (red) of a detector across a few pixels. The area closest to each pixel center (black dots) is shown with dotted lines. In the nearest neighbor model, the value associated with each sample is simply that of the closest pixel, regardless of where inside that pixel it is. **b**: Example detector signal (red) for the same path. The closest matching model (green) leaves a jagged residual (blue) that has power on all lengthscales despite the signal itself being very smooth. For comparison, if our model were a constant zero, then the residual would just be the signal itself (red), and hence smooth. **If smooth residuals are much cheaper in the likelihood than jagged ones, then a zero model will be preferred to one that hugs the signal as tightly as possible** like the green curve.

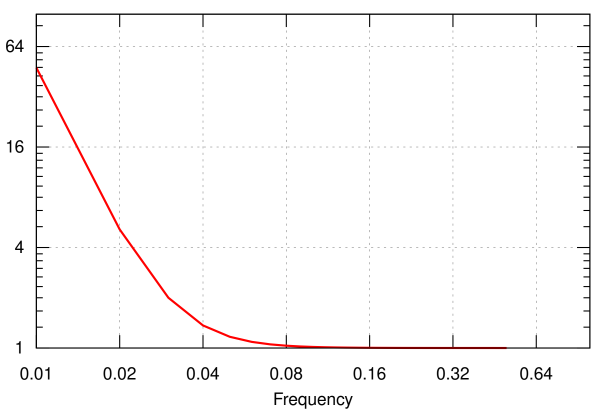

Figure 3. The noise model/inverse weights/inverse filter used in the subpixel bias demonstration in figures 4 and 5. It is a simple Fourier-diagonal 1/f + white noise spectrum typical for ground-based CMB observations. The frequency axis is in dimensionless units in this toy example, but for real telescopes the transition from white noise is typically a few Hz, corresponding to multipoles of hundreds on the sky.

2.1.4. *Destriping*

Destriping splits the noise into a correlated and uncorrelated part, and models the correlated noise as as a series of slowly changing degrees of freedom to be solved for jointly with the sky image itself. The data is modeled as

$$d = Pm + Qa + n_w \tag{3}$$

where n_w is the white noise with diagonal covariance matrix N_w, and Q describes how each correlaetd noise degree of freedom a maps onto the time-ordered data, typically in the form of seconds (ground) to minutes (space) long baselines. Given this, the maximum-likelihood solutions for a and m are

$$\begin{aligned} Z &= I - P(P^T N_w^{-1} P)^{-1} P^T N_w^{-1} \\ a &= (Q^T N_w^{-1} Z Q + C_a^{-1})^{-1} Q^T N_w^{-1} Z d \\ m &= (P^T N_w^{-1} P)^{-1} P^T N_w^{-1} (d - Qa) \end{aligned} \tag{4}$$

where C_a is one's prior knowledge of the covariance of a. Destriping allows a speed/optimality tradeoff in the choice of baseline length and prior, and approaches the maximum likelihood map when the baseline length is a single sample and $C_a + N_w = N$. We implement this limit below, but explore other choices in section **??**. In our toy example $N_w = I$, so $C_a = F^{-1} - I$.

```
iCa    = np.linalg.inv(np.linalg.inv(F) - I)
Z      = I-P.dot(np.linalg.solve(P.T.dot(P), P.T))
a      = np.linalg.solve(Z+iCa, Z.dot(signal))
map_ds = np.linalg.solve(P.T.dot(P), P.T.dot(
    signal - a))
```

SKN: refs and citations from https://arxiv.org/pdf/1103.2281.pdf

2.1.5. *Results*

Figure 4 compares the recovered 1D sky "images" for the different mapmaking methods for this toy example. All methods are expected to have a small loss of power at

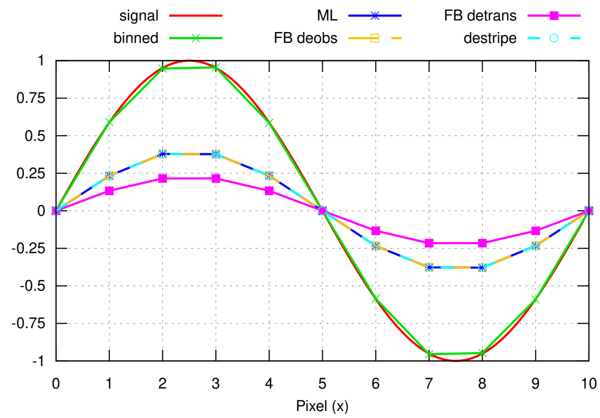

Figure 4. Demonstration of large loss of power in long-wavelength mode caused by the poor subpixel treatment in the standard nearest-neighbor pointing matrix. Figure 3 shows the noise model/inverse weights/inverse filter used in the various methods. **signal**: The input signal, a smooth long-wavelength mode, sampled at 10 samples per output pixel. **binned**: Simple binned map (the unweighted average per pixel). Very suboptimal in the presence of correlated noise, but unbiased. **ML**: Maximum-likelihood map. 2/3 of the signal is lost despite the naive expectation of biaslessness for this estimator. **FB deobs**: Filter+bin map debiased using an observation matrix. Identical to ML. **FB detrans**: Filter+bin map debiased by deconvolving a transfer function measured from simulations. Even more biased than the others due to ignoring mode coupling. **destripe**: Destriper in the maximum-likelihood limit (1-sample baselines with optimal baseline prior). Identical to ML.

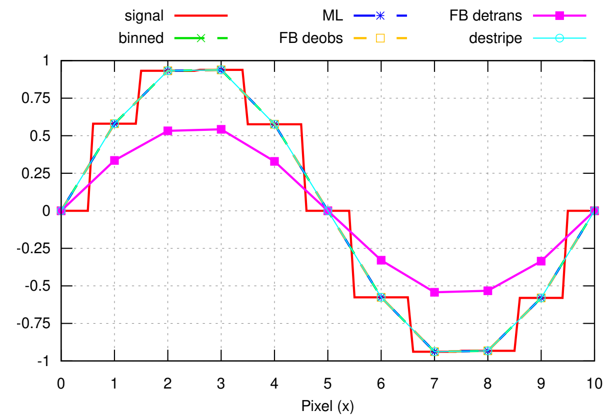

Figure 5. Like figure 4, but with the input signal having the same nearest-neighbor pixelization as the models. In this case all models except FB detrans are unbiased.

small scale (called the "pixel window") due to averaging the signal within each pixel, but this effect is well-known, easy to model, and not our focus here. We deconvolve it using the following function before plotting.

```
def dewin(x): return np.fft.irfft(np.fft.rfft(x
    )/np.sinc(freq),n=len(x)).real
```

After pixel window deconvolution the binned map (green) closely matches the input signal (red). The same can not be said for the other estimators. Maximum likelihood, filter+bin with observation matrix debiasing and destriping (which are all equivalent in the limit I consider here) are strongly biased, with the signal only being recovered with 1/3 of its real amplitude.

The situation is even worse for for filter+bin with simulation-based debiasing, as this suffers from an additional bias due to assuming that the Fourier modes are independent.[3]

2.1.6. *Explanation*

To see why inaccuracies in modelling the signal at sub-pixel scales can bias the largest scales in the map, let us consider the example in figure 2, where a detector measures a smooth, large-scale signal while moving across a few pixels. With a nearest-neighbor pointing matrix it is impossible to model this smooth signal: the model for each sample is simply that of the closest pixel, regardless of where inside that pixel it is. The model therefore looks like a staircase-like function in time domain.

Given that we can't exactly match the signal, what is the best approximation? Let us consider two very different alterantives. **Model A**: The value in each pixel is the average of the samples that hits it, making the model curve trace the smooth signal as closely as it can. This is the green curve in figure 2, and has the sawtooth-like residual shown with the blue curve. **Model B**: The value in each pixel is zero, and the residual is simply the signal itself. Model B seems like a terrible fit to the data, but under a reasonable noise model for a ground-based telescope, like the one shown in figure 3, it will actually have a higher likelihood (lower $\chi^2 = r^T N^{-1} r$ where r is the residual) than model A. The reason is that while model B has a much larger residual than model A, model B's residual is smooth and hence has most of its power at low frequency in time domain where N^{-1} is very small. Meanwhile, model A's residual extends to all frequencies due to its jagged nature, including the costly high frequencies. The actual maximum-likelihood solution will be intermediate between these two cases, sacrificing some but not all of the large-scale power in the model to make the residual smoother.

To demonstrate that the bias really is caused by subpixel errors, I repeated the simulation with a signal that follows the same nearest-neighbor behavior as the data model, thus eliminating subpixel errors. The result

[3] This additional bias disappears if the simulations have exactly the same statistical properties as the real signal we wish to recover.

is shown in figure 5. The bias has disappeared in all methods except filter+bin with transfer-function based debiasing (which has an additional source of bias due to its assumption of a Fourier-diagonal filter response).

2.2. *2D toy example*

The 1D toy example is useful for understanding the origin of the bias, but its unrealistic observing pattern makes it insufficient for exploring optimality/bias trade-offs in the different methods. I therefore made a larger toy example where a single detector scans at constant speed across a square patch, sampling $N_{\rm scan} = 400$ equi-spaced rows with $N_{\rm scan}$ equi-spaced samples per row, followed by a column-wise scann the same patch, leading to a total cross-linked data set $N_{\rm samp} = 2N_{\rm scan}^2 = 3200$ samples long. This equi-spaced grid of samples was chosen to make it easy to simulate a signal directly onto the the samples without needing the complication of pixel-to-sample projection. These samples will then be mapped onto a square grid of pixels with a side length of $N_{\rm side} = 100$ using the different mapmaking methods.

For the signal I draw realizations from a CMB-like $1/l^2$ power spectrum with a Gaussian beam with standard deviation $\sigma = 3$ pixels. Here l is the pixe-space wavenumber, which I evaluate in the higher resolution sample space as

```
ly = np.fft.fftfreq(N_scan)[:,None] * N_side/
    N_scan
lx = np.fft.fftfreq(N_scan)[None,:] * N_side/
    N_scan
l  = (ly**2+lx**2)**0.5
```

With this I can define the signal power spectrum $C_l = l^{-2}$ and beam $B_l = \exp(-l^2\sigma^2/2)$, and draw signal realizations as

```
signal_map = np.fft.ifft2(np.fft.fft2(np.random
    .standard_normal((N_scan,N_scan)))*Cl**0.5*
    Bl).real
signal_tod = np.concatenate([signal_map.reshape
    (-1), signal_map.T.reshape(-1)])
```

The last step here takes into account that the simulated scanning pattern covers the field twice, first horizontally and then vertically.

For the noise I use a simple 1/f spectum $N(f) = 1 + (f/f_{\rm knee})^\alpha$, with `f = np.fft.rfftfreq(N_samp)` and the knee frequency $f_{\rm knee}$ corresponding to 1/30th of the side length, $f_{\rm knee} = 0.5 \cdot 30/N_{\rm scan} = 0.0375$ (6.7 pixel wavelength), and with an atmosphere-like exponent $\alpha = -3.5$.

The pixel coordinates of each sample are

```
pix_pat1 = (np.mgrid[:N_scan,:N_scan]*N_side/
    N_scan).reshape(2,-1)
pix_pat2 = pix_pat1[::-1]
pix = np.concatenate([pix_pat1,pix_pat2],1)
```

which I use to build the nearest-neighbor pointing matrix

```
iy, ix = np.floor(pix).astype(int)%N_side
P_nn   = scipy.sparse.csr_array((np.full(N_samp
    ,1),(np.arange(N_samp),iy*N_side+ix)),shape
    =(N_samp,N_pix))
```

2.2.1. *Cases*

I will investigate 5 classes of nearest-neighbor maps:

1. **bin**: Simple binned map, which I expect to be unbiased but very noisy
2. **ML**: Maximum-likelihood map. Ideally unbiased and optimal, but will deviate from this due to model errors.
3. **ML wX**: Maximum-likelihood maps with a “whitened” noise model, which overestimates the white noise power by a factor 10^X, $X \in \{1, 2, 3\}$, reducing the overall correlatedness of the noise model. I expect the bias to be proportional to the overall dynamic range of the noise model, so making the nosie model whiter should reduce bias. The cost will be suboptimal noise weighting, leading to a noisier map, but it might be worth it.
4. **DS** X: Destriping map with a baseline of $X \in \{4, 16, 64\}$ samples and no amplitude prior. This is probably the most common type of destriping. The baseline lengths can be compared with the noise knee wavelength of 6.7 pixels ≈ 27 samples.
5. **DS+** X: Like DS X, but uses the correlated part of the maximum-likelihood noise model as an amplitude prior (C_a in equation 4).
6. **FB**: Filter+bin mapmaking where a transfer function is measured using simulations based on the data model, and this is deconvolved when measuring the power spectrum. This is the standard approach for filter+bin mapmaking. In this case only the power spectrum is debiased, not the map, so I don’t show an example map for this case.

Since all the mapmaking methods are linear, the signal and noise can be mapped separately. I make 400 signal-only and noise-only data realizations, and map them using each method, computing the mean signal and noise power spectra.

2.2.2. *Results*

Example maps are shown in figure 6, while the bias and noise are quantified in figures 7 and 8 respectively. As expected the binned map is unbiased but extremely noisy. The maximum-likelihood map is low-noise, but

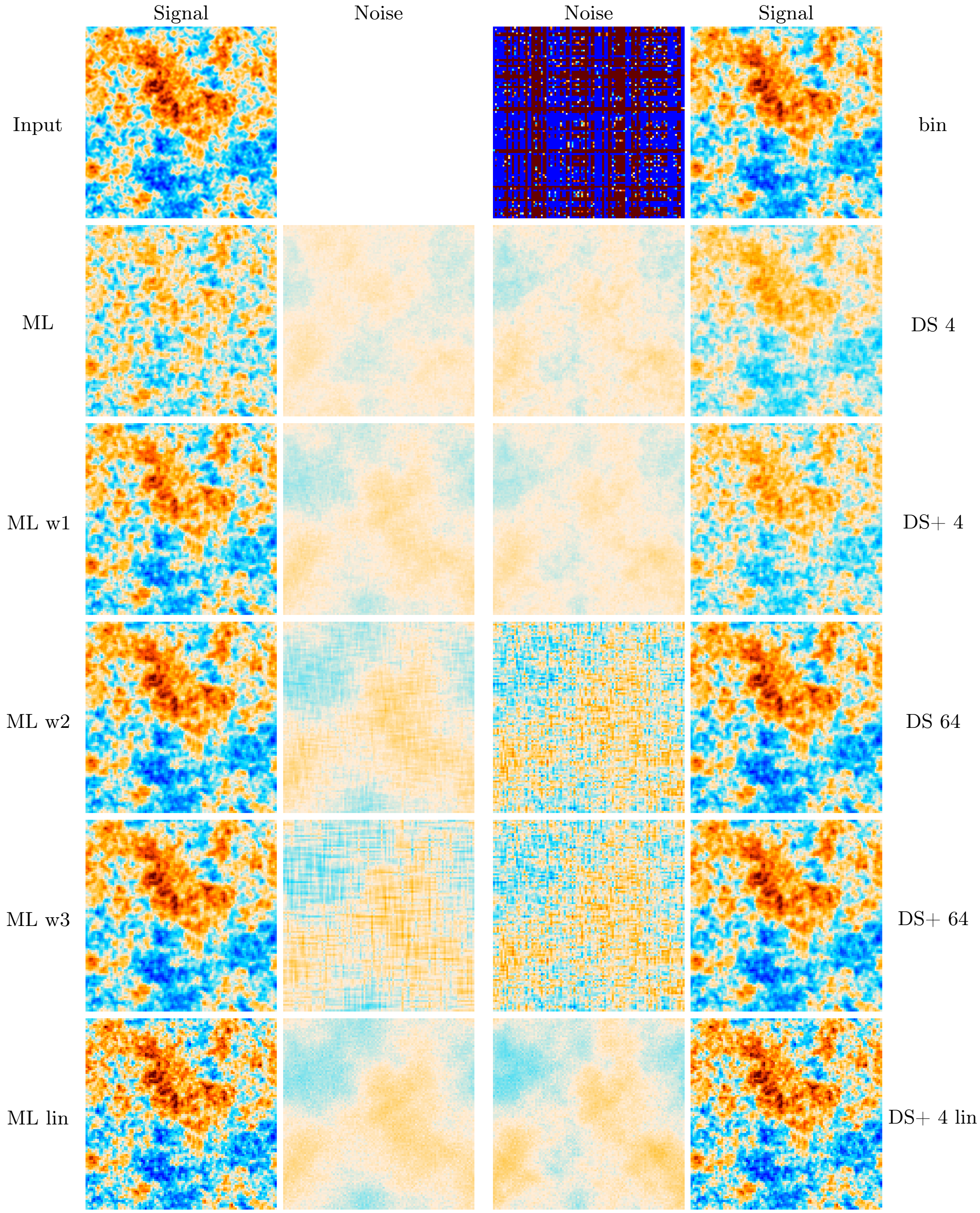

Figure 6. Example signal and noise maps for the different mapmaking methods. The top left map is the input signal, which is directly evaluated at 4x higher resolution than what is used for reconstructing the output maps. All output maps were built assuming a nearest neighbor pointing matrix, except for the last row where a bilinear pointing matrix was used. The binned map is bias-free but has uselessly high noise. Maximum likelihood (ML) and destriping with short baselengths (with (DS+) and without (DS) baseline amplitude prior, which only matters for the shortest baseline) are all low-noise but biased on large scales. This bias goes away as the noise model is artificially whitened (ML wX) or the baseline length is increased (for destriping), but this comes at a cost of increased noise on all scales. Bilinear mapmaking (last row) instead avoids the bias by eliminating most of the model error. It can be difficult to judge how significant the bias and noise levels are from these images. See figures 7 and 8 for easier to interpret comparisons of the signal and noise power spectra respectively.

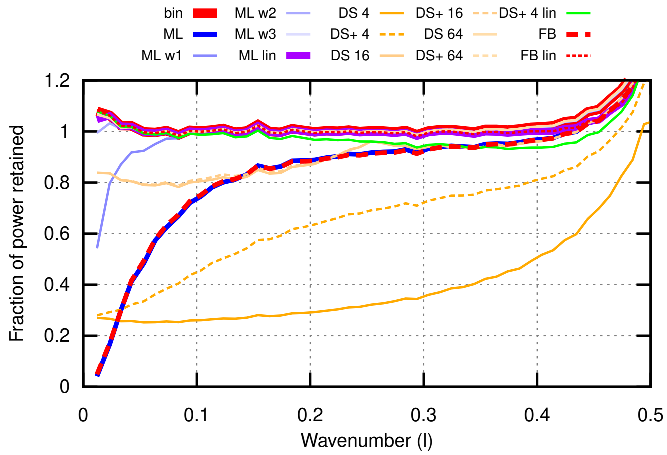

Figure 7. Comparison of the subpixel bias of different mapmaking methods described in section 2.2.1 and discussed in section 2.2.2. Standard maximum-likelihood (thick blue) is strongly biased, as are all but the (very noisy) longest destriping baseline. This bias disappears (ML) or is greatly reduced (DS) when switching to bilinear interpolation in the pointing matrix. See figure 8 for the corresponding noise spectra. I interpret upturn at $l > 0.4$ (close to the Nyquist frequency) as aliasing, which is expected and not relevant for the biases we consider here.

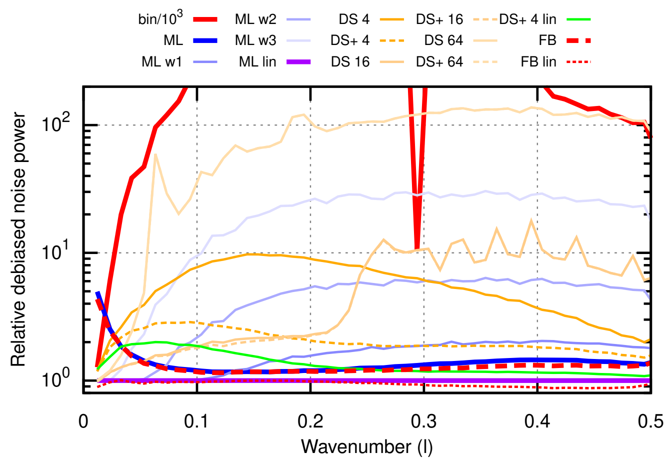

Figure 8. Comparison of the optimality for the methods defined in section 2.2.1, measured as the the debiased noise power spectrum for each method relative to that of bilinear maximum-likelihood mapmaking. See figure 7 for the bias of each method. Note the logarithmic vertical axis, and that binned case was divided by 10^3 to bring it partially inside the plot bounds. It's clear that naive bias mitigation methods like artificially whitening the noise model or increasing the baseline length are too costly to be practical.

measurably biased on all scales, with a power deficit of a few percent at the smallest scales which grows to almost 100% on the largest scales. The deficit appears to fall proportionally with the whitening, with ML w1 and w2 being respectively 10x and 100x as close to unbiased. Sadly this comes at the cost of 40% and 350% higher noise power respectively. These numbers may differ for real-world cases, but this still seems like a very expensive bias mitigation method. All but the longest-baseline destriped maps are also strongly biased, with the shortest baseline being considerably worse than ML for almost all scales. Much like we saw with the ML variants, the less biased destriping versions come at a high cost in noise. Finally, the filter+bin map is biased even after simulation-based debiasing due to the simulations not capturing the subpixel behavior of the real data. Both the bias and noise levels are the same as maximum-likelihood in this example.

To test whether the observed biases are truly caused by subpixel errors, I repeat the simulations with only one sample per pixel ($N_{\rm scan} = N_{\rm side}$). As expected this results in an unbiased power spectrum for all methods.[4]

2.2.3. *Effective mitigation of subpixel errors*

There will be no subpixel errors in the limit of infinitely small pixels, so it's tempting to simply reduce the pixel size to solve the problem. This does work, but since subpixel errors are proportional to $|\nabla s \cdot \vec{\Delta}|$, where s is the true, smooth signal on the sky and $\vec{\Delta} = [\Delta_x, \Delta_y]$ is the pixel shape, the improvement is only first order in the pixel side length. A much more feasible solution is to instead improve the subpixel handling in the pointing matrix. Going from nearest neighbor to bilinear interpolation is enough to practically eliminate subpixel errors. We can implement this in the toy example by defining

```
pix_left = np.floor(pix).astype(int)
ry, rx   = pix-pix_left
iy1,ix1  = pix_left % N_side
iy2,ix2  = (pix_left+1)% N_side
weights  = np.concatenate([
    (1-ry)*(1-rx), (1-ry)*rx,
    ry    *(1-rx), ry    *rx])
samps    = np.tile(np.arange(N_samp),4)
inds     = np.concatenate([
  iy1*N_side+ix1, iy1*N_side+ix2,
  iy2*N_side+ix1, iy2*N_side+ix2])
P_lin  = scipy.sparse.csr_array((weights,(samps,
    inds)),shape=(N_samp,N_pix))
```

Bilinear mapmaking results in a different pixel window than the standard `pixwin_nn = sinc(ly)[:,None]*sinc(lx)[None,:]` of nearest neighbor mapmaking. I measure this empirically using simulations of simple binned mapmaking, and find it to have the separable form `pixwin_lin =`

[4] Even filter+bin, which ignores mode coupling, ends up having an unbiased power spectrum because the simulations used to build the transfer function followed the same distribution as the real signal.

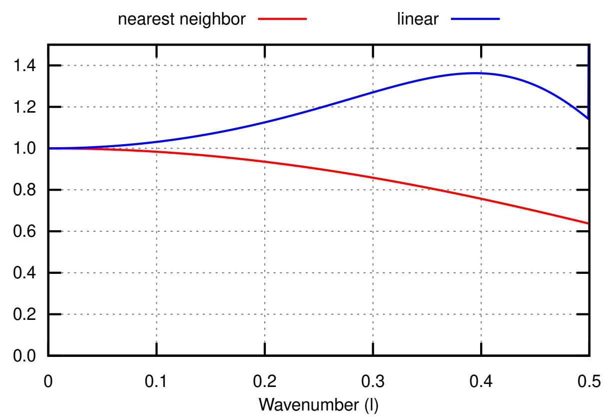

Figure 9. Comparison of the 1D pixel window for nearest neighbor mapmaking (red, sinc(l)) and linear mapmaking (blue). The 2D pixel window is the outer product of the 1D one along each axis. The pixel windows model the response of the fourier coefficients to binned (unweighted) mapmaking. Square it to get the effect on the power spectrum.

`linwin1d(ly)[:,None]*linwin1d(lx)[None,:]`, with `linwin1d` being plotted in figure 9.

Since each sample in bilinear mapmaking gets contributions from the four closest pixels, the pointing matrix is about four times slower than the standard nearest neighbor version. However, as shown in figures 7 and 8 this cost is worh it, with bilinear maximum-likelihood mapmaking being bias-free and even lower noise than standard ML. The bias is also greatly reduced for destriping, but not eliminated.

3. DON'T THINK YOU'RE SAFE JUST BECAUSE YOU AVOID SUBPIXEL ERRORS

By no means all model errors have the strong impact on large scales that subpixel errors do. For example, gain errors typically have a scale-independent response while pointing errors broaden the beam, damping small scales but not large ones. It is, in fact, quite difficult to come up with other mechanisms by which model errors can cause large-scale power bias. One example I found requires a multi-detector system large relative gain errors combined with a noise model where the noise is dominated by a long-wavelength common mode (strong correlation between detectors at long wavelengths). This is implemented in the program `common_mode_test.py` with the output shown in figure 10. In this example the large scales are biased *high* and also subject to a phase shift, while the small scales have negligible bias.

While the commmon-mode/bias-interplay works as a demonstration, it is a much less robust error mechanism than subpixel errors, and requires quite extreme parameters to produce appreciable bias. In my tests it also tends to disappear as more data is added and crosslinking increases, and I have not been able to provoke large-scale bias using this method for full-scale realistic datasets (unlike subpixel errors!).

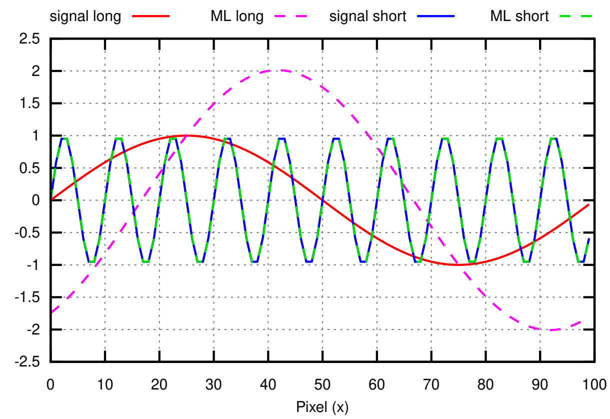

Figure 10. Demonstration of large-scale bias in a multi-detector system due to an interaction between strong large-scale detector correlations in the noise model and large relative gain errors between the detectors. **signal long**: An input long-wavelength signal, with the same pixelization as the output to avoid subpixel bias. **ML long**: Corresponding maximum-likelihood map, which exhibits both an amplitude and phase error. **signal short** and **ML short**: The same, but for a short-wavelength mode. Here the bias is negligible, despite the model's gain errors being scale-independent.

It would be convenient if the difficulty in finding model error mechanisms that bias large scales meant that subpixel errors are the only ones one needs to worry about, but sadly that does not seem to be the case. In my practical experience with ground-based CMB mapmaking there have been cases of large-scale power loss where subpixel errors were responsible for less than half of the total. In these cases the difficulty in finding these mechanisms becomes a curse rather than a blessing, since it makes it very difficult to track down the source of the bias. What makes model error bias especially insidious is that it is completely invisible to any simulation that does not explicitly include that particular type of model error. It is therefore very easy to trick oneself into believing that one has an unbiased analysis pipeline while there are in fact O(1) biases remaining.

REFERENCES

Naess, S. K. 2019, JCAP, 2019, 060–060, How to avoid X'es around point sources in maximum likelihood CMB maps, http://dx.doi.org/10.1088/1475-7516/2019/12/060