Computational techniques for efficient Bayesian analysis of the Cosmic Microwave Background

Thesis submitted for the degree of Philosophiae Doctor

by

Dag Sverre Seljebotn

Institute of Theoretical Astrophysics
University of Oslo

2017

Preface

In 2009 ESA launched the Planck space observatory. For four years it scanned the microwave frequencies in all directions around us. Goal: Gather the best measurements to date of the Cosmic Microwave Background (CMB). The CMB, originating from roughly 380,000 years after the Big Bang, is our richest source of data for learning about the universe as a whole, and enables us to check cosmological hypotheses and estimate cosmological parameters. CMB measurements of ever-increasing sensitivity and resolution has turned modern cosmology from a theory-driven science to a data-driven science.

As the amount of available CMB data explodes, the statistical modelling and computational algorithms need to follow suit. This thesis advances the state of the art in two areas: 1) New algorithms for *CMB component seperation*; and 2) novel approaches for computing the *Spherical Harmonic Transform*.

As seen by Planck and other telescopes, the CMB radiation is contaminated by radiation from matter in the Milky Way on the same microwave frequencies. Eriksen et al. (2008) developed an elegant framework for *component seperation*; providing estimates of the pure CMB disentangled from this *foreground radiation*. As it stood, the method could only be applied to the data from Planck if the data was first downgraded to lower resolution, throwing away more than 90% of the data in the process. This thesis presents a novel algorithm for multi-resolution component seperation that enables analysis of Planck data at full resolution. The estimates produced are still exact, faithfully propagating the instrumental noise in every pixel to the final parameter condidence intervals – what has changed is that the estimates are computed between twice as fast (full sky coverage) to 1000s of times faster (mask applied)[1].

The computational bottleneck of the CMB component seperation problem is the *Spherical Harmonic Transform* (SHT); the spherical analogue to the much more popular Fourier transform. This thesis implements an experimental research code for approximate SHTs that beat the speed record for computing the SHT by a factor of 2x–6x, and presents a novel strategy for computing the SHT on the GPU[2]. The author also contributed to the development of the Libsharp SHT library, which is included with the very popular HEALPix library, used by thousands of researchers every day[3].

[1] Paper II, Paper VI
[2] Paper III, Technical Report I
[3] Paper IV

Acknowledgements

This thesis is dedicated to my wife Åshild and my children Eskil and Astrid.

Thanks to my supervisor Hans Kristian Eriksen for a great time. If I had only followed your advice more often I would have gotten twice as much done in half the time.

Thanks to Martin Reinecke for many discussions about spherical harmonic transforms, Ingunn Wehus for helping me get hold of and understand Planck data, Kent-André Mardal and Mikolaj Szydlarski for being my guides in the world of linear solvers, Jeff Jewell for chasing many crazy ideas with me, Mark Tygert for being enthusiastic about my work on his algorithm, and Xing Cai for co-supervising.

Thanks to my office-mates and co-group-members over the years; Phil Bull, Unni Fuskeland, Eirik Gjerløw, Kristin Mikkelsen, Sigurd Næss, Benjamin Racine and Tone Ruud, for all the things I have learned from you and for making my time at the institute enjoyable.

Contents

Part I

Synopsis

Chapter 1

A primer on CMB analysis

1.1 Cosmology and the CMB

As one looks out into the universe, one observes electromagnetic radiation (light) that has travelled for a very long time. Looking past all of the stars and galaxies, way in the back, one can see a wall of electromagnetic radiation in the microwaves. This radiation originates from roughly 380,000 years after the Big Bang, at the moment where the universe cooled sufficiently for electrons to bond together with protons to form hydrogen, thus marking the start of a translucent universe. This *Cosmic Microwave Background* radiation (CMB) was discovered by accident in 1964 by Arno Penzias and Robert W. Wilson (Penzias and Wilson, 1965) while they were testing new radio equipment for Bell Labs. The remarkable thing they noted about the CMB is that it was exactly the same no matter where on the sky they pointed their antenna. This discovery established the Big Bang theory, and Penzias and Wilson were awarded the 1978 Nobel Prize in Physics.

While the most important feature of the CMB is simply its existence, it was theorised that it should have tiny fluctuations originating from the uneven distribution of matter in the universe. The NASA-funded Cosmic Background Explorer (COBE) Differential Microwave Radiometers (DMR) launched in 1989 and made the first observation of these so-called *anisotropies* (see figures 1.1, 1.2). The intensity of the CMB is given in Kelvin; and while the main background is at 2.73 Kelvin, the anisotropies on top are at the $100\mu K$ level. Thus observing the anisotropies requires highly sensitive instruments.

After COBE there has been a large number of CMB experiments, each pushing the boundaries in sensitivity and resolution, leading to an explosion in the available data. Most of the experiments focus on smaller patches of sky, using telescopes on the ground or in balloons. This thesis will primarily focus on full sky missions, of which there are three: After COBE, NASA funded the Wilkinson Microwave Anisotropy Probe (WMAP) which launched 2001, and ESA funded the Planck space telescope which launched in 2009. Figure 1.2 shows the gradual increase in resolution from these three missions. As the resolution and sensitivity is increased, new computational techniques must be

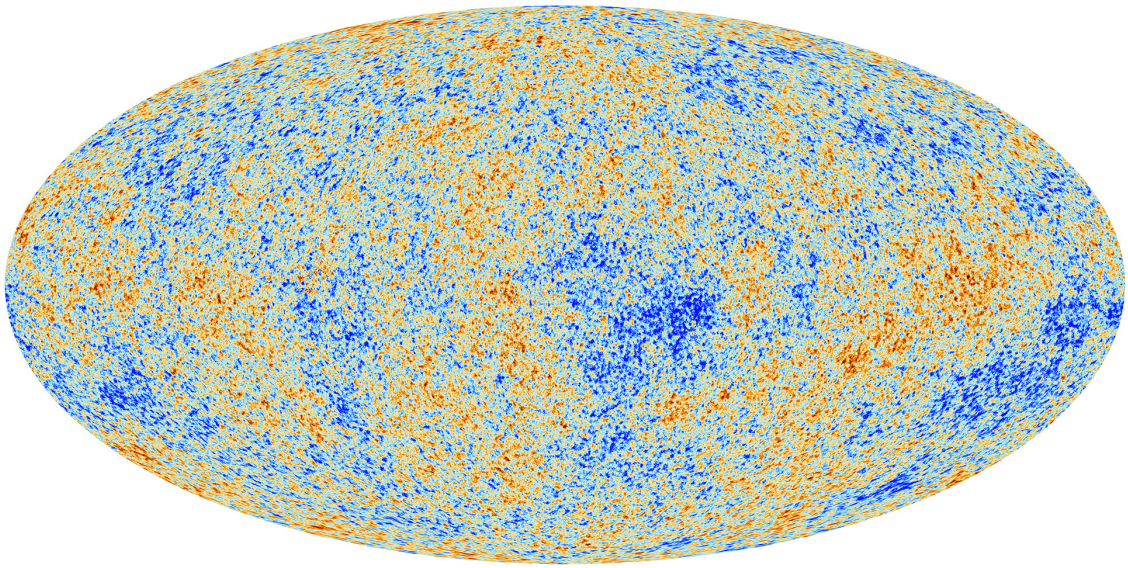


Figure 1.1: An estimate of the CMB anisotropies over the full sky, made by the Planck team. The anisotropies are simply the difference between the measured temperature in a direction and the mean CMB temperature of 2.725 K. The measurements are in every direction around us in the universe, i.e. a sphere. This so-called "map", and every other spherical map in this thesis, is displayed in the Mollweide projection. The Mollweide projection gives an accurate representation of areas and scales, at the expense of having inaccurate angles and shapes. How to make this estimate of the CMB and the limitations inherent in this image is covered later in this thesis. This image is taken from a ESA/Planck 2013 press release.

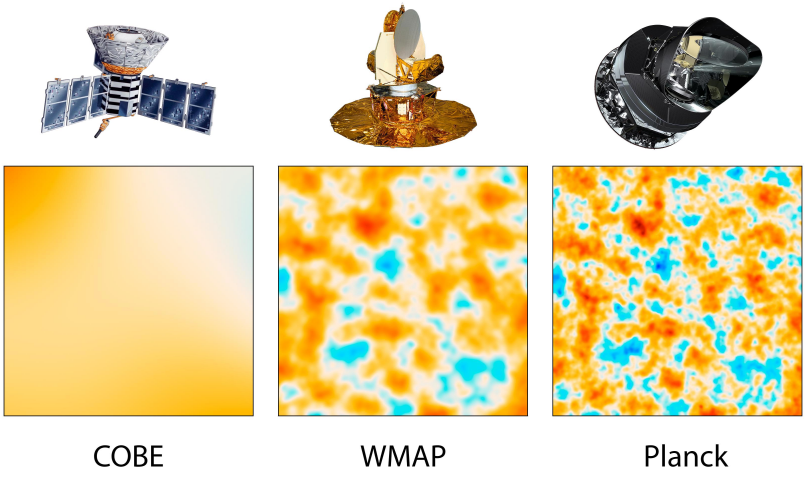


Figure 1.2: Comparison of resolution of temperature anisotropies from fullsky CMB missions. The patches cover 10 square degrees of sky. Illustration by NASA/JPL-Caltech/ESA.

.

developed for analysis of the data. The data recorded by Planck is still being analysed, and this effort is the backdrop of this thesis.

It is believed that Planck has retrieved all of the information that is available in the *temperature anisotropies*. However, there is additional information in the polarization of the radiation (that is, the orientation of the photons as they hit the detectors), and here the race is still very much on to make the best observations first. If sufficiently sensitive observations are made, one should be able to see a signature of the so-called *gravity waves*, which would be a huge scientific discovery.

The importance of the anisotropies is not to give us a concrete physical map of the early universe. Rather, the insights comes from studying the statistical pattern of the anisotropies. How strong are large waves compared to the smaller waves in general? From such studies much can learn much about the physical laws that govern the universe at large. If our universe had been fundamentally different from the one we live in, there would have perhaps have been stronger large-scale waves and weaker small-scale waves, or vice versa.

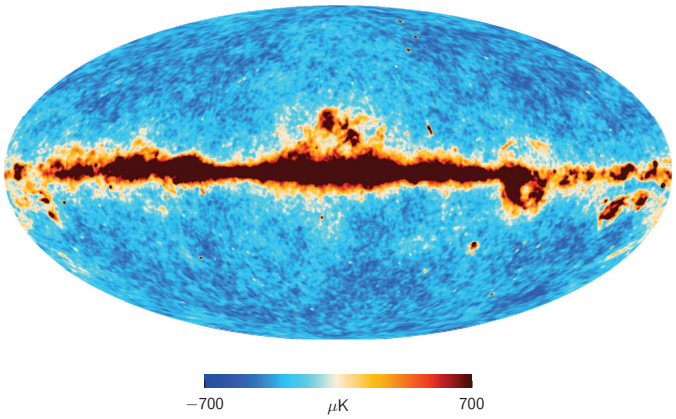
The systematic way to study the CMB is through its *power spectrum*, which is a way of estimating the intensity of each wavelength. Different *cosmological models* with different *cosmological parameters* will predict different power spectra, and so proposed cosmological models can be tested against the CMB data, and the CMB data used to estimate associated parameters. Some of the cosmological parameters tell us something about the fundamental nature of the universe at the time the CMB was generated: How much regular matter was present, the existence and quantity of *dark matter* and *dark energy*, and the specific time the universe went from opaque to translucent. Then, after the CMB photons started passing freely through the universe, they have travelled for a long time through space; this further processes the power spectrum, and makes it possible to learn both how long the photons have travelled, and properties of the space-time through which they went. Our best estimate of the age of the universe, 13.8 billion years, comes directly from the CMB observations made by Planck. We return to this topic in section 1.5.

1.2 The microwave sky

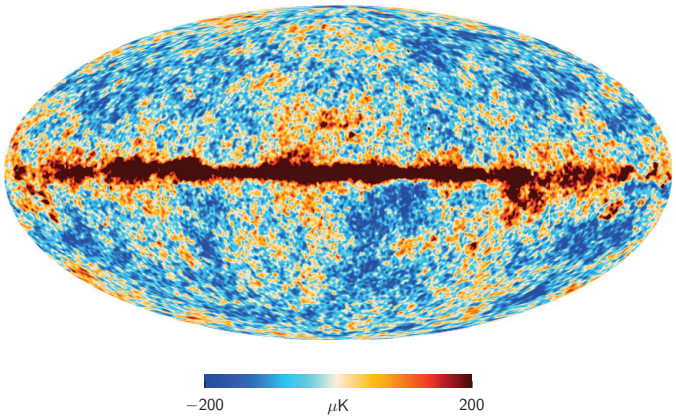
If we possessed a perfect photo of the CMB, we could simply read off its power spectrum directly (as detailed in the next section on *spherical harmonic transforms*) and then see how well different cosmological models fits and, assuming a model, translate the spectrum to cosmological parameters.

However, we do not possess such a perfect photo. In this section we explain how to go from data to cosmology, with emphasis on the "Commander" Gibbs sampling approach (Jewell et al., 2004; Wandelt et al., 2004; Eriksen et al., 2008).

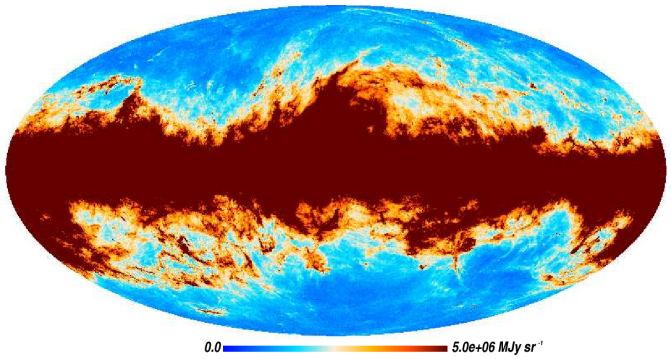
Figure 1.3 shows some full sky images taken by the Planck space observatory on different frequencies. It is evident that there are other sources of microwave radiation beside the CMB. In particular the horizontal band present on all frequencies is radiation from matter within our own galaxy. Figure 1.4 displays estimates of radiation from different types of sources (*components*) estimated



(a) 30 GHz map, Planck LFI (Planck Collaboration VI, 2015)

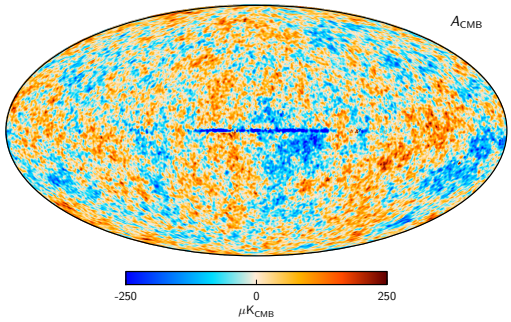


(b) 70 GHz map, Planck LFI (Planck Collaboration VI, 2015)

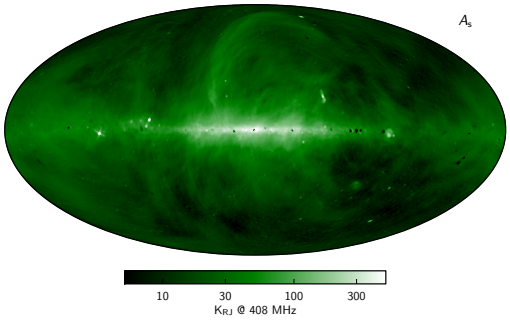


(c) 857 GHz, Planck HFI (Planck Collaboration VIII, 2015)

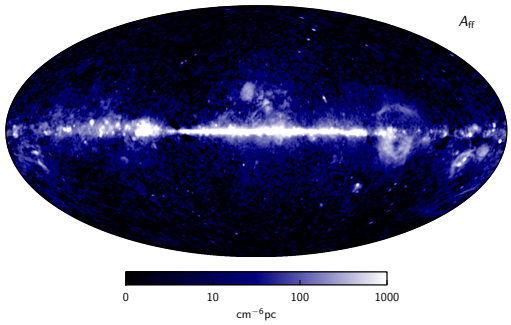
Figure 1.3: Three of the nine fullsky images of microwave radiation produced by the Planck space observatory, in the Mollweide projection. The orientation is such that the Milky Way galactic plane is along the horizontal axis. Comparing with 1.4, The 30 GHz map is a sum of CMB, *free-free*, *spinning dust* and *synchrotron* emissions, the CMB is dominant in the 70 GHz map outside of the galactic plane, and *thermal dust emissions* dominate in the 857 GHz map. Panel (e) in 1.4 lists all the 9 frequencies that Planck scanned. In addition to the *intensity* components shown here, each frequency also has two *polarization* components, see section 1.6.



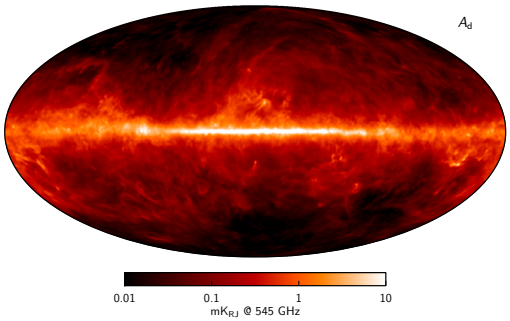
(a) CMB



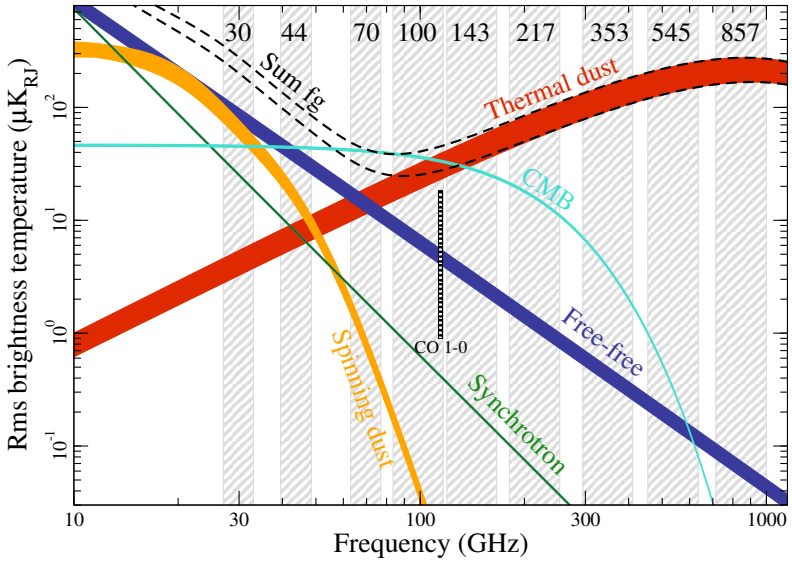
(b) Synchrotron emission



(c) Free-free emission



(d) Thermal dust emission



(e) Spectral energy density (SED) of each component

Figure 1.4: Figures from Planck Collaboration X (2015) illustrating component seperation. Panels (a) to (d): Estimates of some of the components of the microwave sky. Other components present in the model are *spinning dust emissions* and *CO emission lines*. The sum of these, plus instrumental noise and beam convolution effects, make up the observed microwave sky shown in figure 1.3. The CMB is obviously mis-estimated in the galactic band, this is taken into account during further analysis as described in section 1.3. Panel (e): Illustration of the rough energy density on different frequencies by different sources of microwave raditation. The functions are sometimes known from physics, but parameters must be fitted to the data and varies a little bit between different positions on the sky, hence bands instead of lines in this figure.

by Planck Collaboration X (2015); of these, the CMB originates at the edge of the observable universe, while the remainder mainly originates from the Milky Way.

We will not dwell on the how the galactic radition arises, but simply consider them input given by astrophysicists to the statistical models. The important feature is that each of the components emits different levels of energy on different frequencies, as indicated in panel (e) of figure 1.4. The fundamental behaviour of this "spectral energy density" (SED) is sometimes known from physics, although for some components one or more parameters has to be estimated.

For instance consider thermal radiation from dust clouds (component (d) in the figure). Depending on the nature of the dust particle, the SED slope, shown by the green band in (e), will the a bit different. Therefore Planck Collaboration X (2015), in addition to the energy density, also estimate a different slope parameter for each position on the sky for some of the components. Naturally this is also an approximation, but it is much better than assuming a single average parameter for the full sky.

Comparing figures 1.3 and 1.4, notice that the CMB component can be seen clearly in the 70 GHz channel, and also faintly as part of the background in the 30 GHz channel. In the latter case there is also a strong presence of *synchrotron emissions*, *free-free emissions* and *spinning dust emissions*. In contrast, the 857 GHz channel is almost a direct image of the *thermal dust emissions*. Planck Collaboration X (2015) describes each of these components in further detail.

One could now look at the power spectrum of the 70 GHz channel away from the galactic plane, and use that to make crude estimates of cosmological parameters. Our estimates will however be much better if we use all the data we have to reconstruct the underlying CMB. Consider that while the 857 GHz image in figure 1.3 contains no information about the CMB directly, it does contain indirect information about the CMB because it provides a good map of where dust is located around us. What we learn about dust in the high-frequency channels we can extrapolate to the other channels, so that they in turn become more informative about the CMB. A major part of this thesis is dedicated to developing numerical algorithms for such *component separation*.

For successful component seperation we need to understand both the physical effects processing the CMB, and the data gathering process, as illustrated in figure 1.5. Below we will focus on three aspects: Foregrounds, instrumental noise, and beam convolution.

Instrumental noise and *beam convolution* are best understood by describing how the images of the microwave sky were captured. The images in figure 1.3 specifically comes from the Planck space observatory, which scanned the sky from the L_2 Lagrange point from 2010 to 2013. It spins around its own axis so that its detectors points in different directions, covering all of the sky over the course of half a year. The resulting data can be viewed in two ways. First you have the detector values as a function of time, so called *Time-Ordered Data* (TOD). Combining these with where the detectors pointed at the given time one gets the spherical images or *maps*. Thus the instrumental noise is fundamentally connected to the timestamp of each detector value, not the

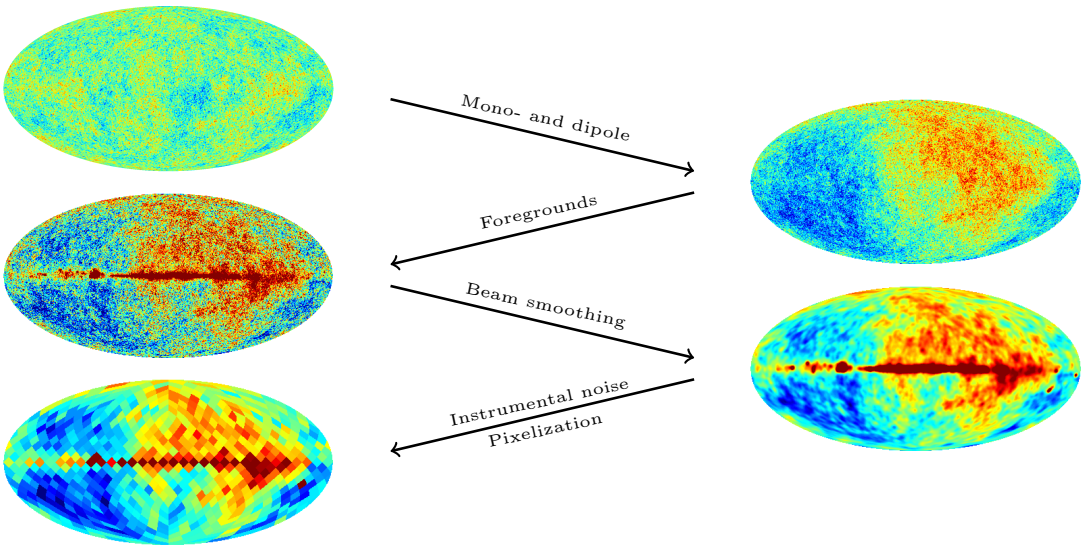


Figure 1.5: Schematic illustration of how the underlying CMB anisotropies are processed on its way to our data files. The Commander approach, described in chapter 2, is to model this processing of the signal, then use Bayesian statistical technqiues to recover the statistical properties the original CMB. The mono- and dipole is not discussed much in this thesis as they are currently removed in a pre-processing step.

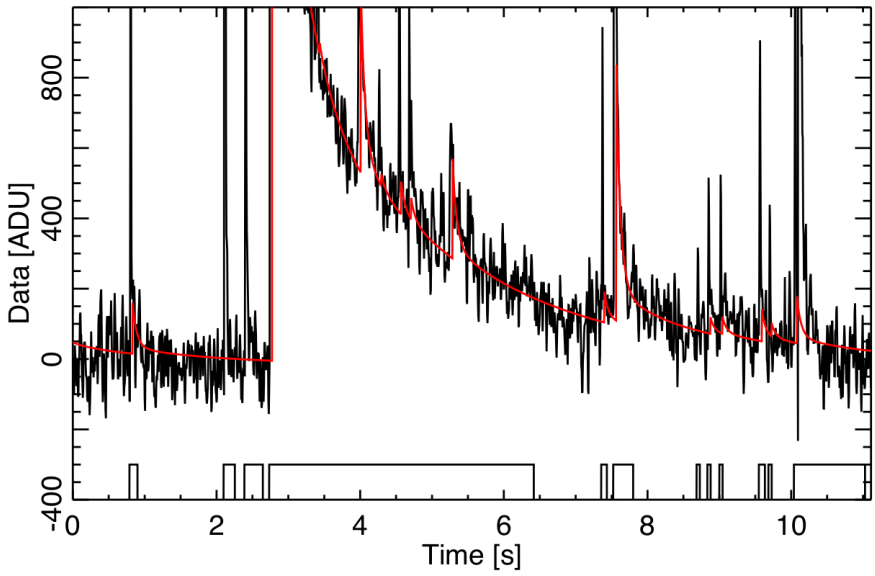


Figure 1.6: Example of Time Ordered Data (TOD), that is, detector value as a function of time. The figure is taken from Planck Collaboration X (2014). This example is from a single bolometer at the 143 GHz channel. The spikes come from energetic particles hitting the bolometer and causing a sudden increase in temperature, it then takes some time to cool down after each particle hit. These spikes are fitted to templates (red line); then these templates are a) subtracted and b) flags that some data should be thrown away (lines at the bottom of the plot). Some of the remaining variation on top of the red lines is instrumental/electrical noise; the properties of this noise is estimated from the TOD, but it cannot be subtracted. Below all these sources of noise, there is a response to varying levels of microwave radiatation as the detector points in different directions on the sky. Since the microwave radiation stays the same in a given direction, while noise is independent of the direction of the instrument, the noise is reduced by scanning the same spot many times.

.

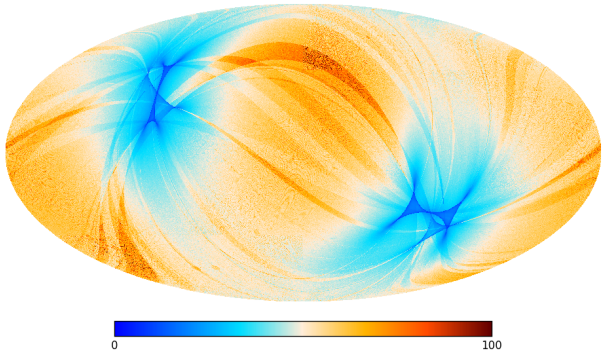


Figure 1.7: The standard deviation of the instrumental noise in each pixel for one of several 143 GHz maps produced by Planck Collaboration VIII (2015). Units in μK. This so-called "RMS map" is a primarily a function of the scanning strategy of the instrument; the two blue regions are aligned with our solar system and have been scanned a lot of times. Another contributing factor is data filtering, as shown in figure 1.6. Finally there is variation in instrumental noise properties over time.

.

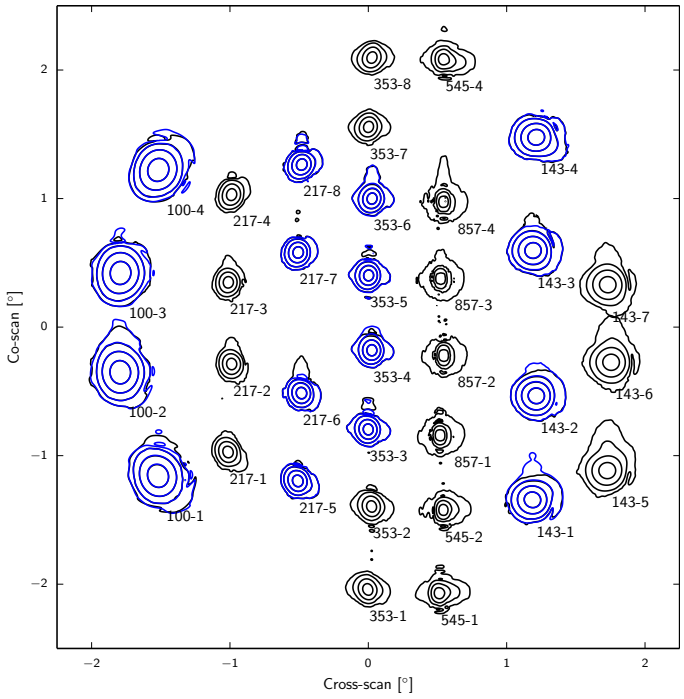


Figure 1.8: Beams reconstructed from scans of Mars, Saturn and Jupiter. Figure taken from Planck Collaboration VII (2015). The countours are logarithmic at -3, -10, -20, and -30 dB from the peak. Note in particular that the higher frequencies (857 GHz) have smaller beams than on lower frequencies (100 GHz). Note that in the final map data, each pixel will have been scanned many times, with the beam in several different orientations, which gives a certain averaging effect. In analysis of the kind done in this thesis, we usually aproximate these beam profiles with radially symmetric beams. Beam convolution can then be done in spherical harmonic domain. An alternative would have been to make use of the code of Mitra et al. (2011) which computes an effective beam for each pixel, but this is computationally more expensive.

.

position on the sky; unlike 2D images taken in a single instant.

Planck contains two different types of detectors: The *High Frequency Instrument* (HFI) using bolometers, and the *Low Frequency Instrument* (LFI) using radiometers. Each has their own unique noise characteristics. For instance, bolometers has a problem with getting hit by relativistic particles, which deposits some energy that temporarily inflate measurements. In general instrumental noise is correlated in time. Much effort is spent analysing and filtering the data in time-domain (see figure 1.6). After projection of the time-ordered data to the spherical maps we in general assume i) that correlations between different pixels can be neglected, ii) that the noise contribution in each pixel is Gaussian, distributed with a known standard deviation per pixel given in the so-called *RMS map*, as seen in figure 1.7.

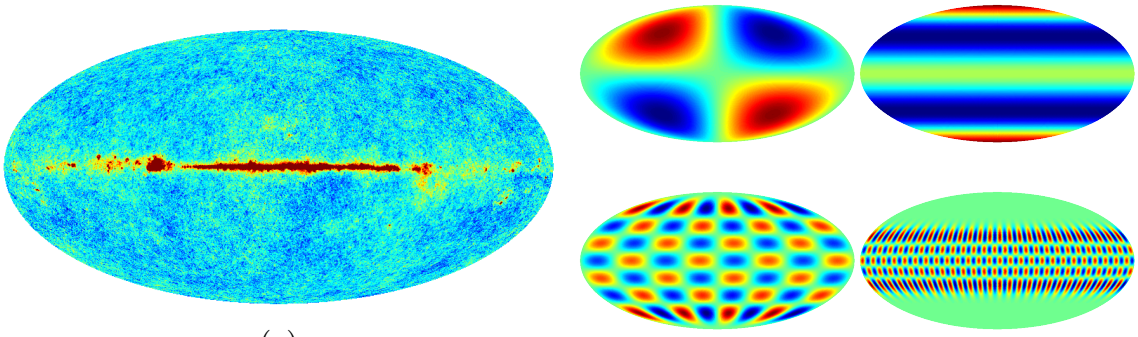
The final effect to take into consideration is the Point Spread Function, or, as it is commonly referred to in the CMB community, the *beam*. When a detector points in a given direction it doesn't observe in an infinitely small point, but rather integrates over a "blob" of a given size. Each detector has a specific beam associated with it; the ones used in the HFI part of Planck can be seen in figure 1.8. In the final maps, each pixel is a sum of many observations within the same pixel, and each observation will have the detector (and thus beam) oriented in a different way. Ideally one would try to observe each pixel in as many orientations as possible, to get a nice symmetric beam. In the case of Planck pixels tend to only be observed in two main orientations, leading to somewhat slanted effective beams. Symmetric beams must often be assumed anyway for computational reasons.

The important point to take away about beams at this point is that they are by nature different for every frequency. While the 30 GHz channel of figure 1.3 was observed with a beam with a Full Width Half Max (FWHM) of 32 arc minutes, the 857 GHz channel has a much smaller beam of only 3.67 arc minutes. This translates directly into higher resolution maps for the higher-frequency channels.

1.3 CMB extraction

So, how does one move from the observed microwave sky of figure 1.3 to an estimate of the CMB such as the one in figure 1.1? There are multiple techniques in use, with different philosophies and trade-offs. The official Planck releases do not even pick a single estimate, but present four different ones, named NILC, SEVEM, SMICA, and Commander (see chapter 5).

The focus of this thesis has been to improve the Commander method, first described by Jewell et al. (2004), Wandelt et al. (2004) and Eriksen et al. (2004), and subsequently developed in Eriksen et al. (2008). The main approach is simply to translate what was just laid out in the previous section as accurately as we can into a statistical model; then use Bayesian statistics and Monte Carlo Markov Chain (MCMC) methods to recover posterior distributions for each cosmological parameter. A strength of this approach is that it should in principle give the most correct answer in the end. The uncertainty in each pixel due to instrumental noise, and any other uncertainties one cares to introduce



(a) (b)

Figure 1.9: A spherical image (a) can be represented as a sum of spherical harmonic basis functions $Y_{\ell m}(\hat{n})$ (b). Plotted here are the real parts of $Y_{2,1}(\hat{n})$, $Y_{4,4}(\hat{n})$, $Y_{10,6}(\hat{n})$, and $Y_{30,4}(\hat{n})$. Higher ℓ corresponds to more waves / higher frequencies / smaller scales.

such as in educated guesses in the model, are faithfully propagated all the way through to the the posterior distribution of the final cosmological parameters.

The downside of such modelling is that if the assumptions one make about the physics are too wrong, one can end up with biased estimates. The Bayesian framework leads to imposing an astrophysical prior on the emissions from synchroton, dust, and so on; both implicitly by stating their existence, and explicitly by putting some statistical priors on their properties. Whether introducing such astrophysical knowledge from the outside and letting them affect the final parameter estimate is an advantage or disadvantage may be a matter of perspective, and of how confident we are in our knowledge of astrophysics and of instrumental effects.

1.4 Spherical harmonic transforms

The key to linking cosmological models with CMB observations is to look at the angular power spectrum of the CMB. How strong are the longer waves versus the shorter waves? Usually one associates the power spectrum with the Fourier transform, but those are only defined for Euclidian spaces, not on spheres. The corresponding tool for analysing spherical images is the *spherical harmonic transform* (SHT). Assume that we have a real field on the sphere $f(\hat{n})$, with $\hat{n}$ a unit vector on the sphere. Then we can represent the same field using the spherical harmonic coefficients $a_{\ell m}$,

$$f(\hat{n}) = \sum_{\ell=0}^{\infty} \sum_{m=-\ell}^{\ell} a_{\ell m} Y_{\ell m}(\hat{n}), \tag{1.1}$$

where $Y_{\ell m}(\hat{n})$ are the spherical harmonic basis functions. See figure 1.9 for examples of such basis functions. Computing f in a set of pixel positions $\hat{n}_i$, given $a_{\ell m}$, is known as *spherical harmonic synthesis*. The opposite action of *spherical harmonic analysis* is given by

$$a_{\ell m} = \int f(\hat{n}) Y^*_{\ell m}(\hat{n}) d\Omega \tag{1.2}$$

where $d\Omega$ indicates integration over the sphere. In one sense these transforms are analogous to Fourier transforms; representing an image with a set of orthogonal basis functions that allow us to directly inspect and filter on frequencies.

The *angular power spectrum* is defined as the statistical variance of these coefficients, averaged over m,

$$C_\ell = \mathrm{Var}(a_{\ell m}). \tag{1.3}$$

Assuming that the mean is zero, which is usually the case, a simple estimator is the *observed power spectrum*,

$$\hat{\sigma}_\ell^2 = \frac{1}{2\ell+1} \sum_{m=-\ell}^{\ell} a_{\ell m}^* a_{\ell m}. \tag{1.4}$$

Cosmological theory predicts a certain power spectrum C_ℓ for the CMB, and it is this that allows us to build a bridge from CMB observations to cosmology, and which makes SHTs so important to CMB analysis.

In contrast to Euclidian spaces, there is no grid on the sphere such that all grid points have the same distances to their neighbors. Therefore several spherical grids are in active use, each with individual strengths and weaknesses. Most commonly used by far in the CMB community is the HEALPix grid (Górski et al., 2005), which has the advantage that one can define pixel borders such that all pixels have the same area, as well as a convenient nesting property where each such pixel can be divided into four child pixels. The trade-off paid is that quadrature rules for equation (1.2) can only be approximate. For other grids there exists quadrature rules that perfectly recover $a_{\ell m}$.

The *discrete Fourier transforms* are convenient to work with numerically as a simple basis change from $\mathbb{R}^n$ to $\mathbb{R}^n$. No equivalent concept exists for SHTs, and one must work with approximations to the continuous transforms. This follows from the irregular spacing of grid points on the sphere, which ensures that some parts of the grid will always see higher frequencies than other parts. One may either use more pixels than spherical harmonic coefficients, and a spherical harmonic signal will survive a round-trip to pixel domain, but not vice versa – or, one may use more spherical harmonic coefficients than pixels, and have the opposite problem.

The currently known algorithms for SHTs are slow compared to the FFT. One part of this thesis is the development of improved computational techniques and code for computing SHTs; we will return to this in chapter 4.

1.5 Cosmological parameters

To recap, we have covered observations of the microwave sky, how the CMB can be extracted from observations of the microwave sky, and how we can get hold of a power spectrum which tells how strong the large waves are relative to shorter waves in the CMB. In order to link this to astrophysical insights, what remains is a cosmological model that *predicts* a CMB power spectrum, such as the one displayed in figure 1.10.

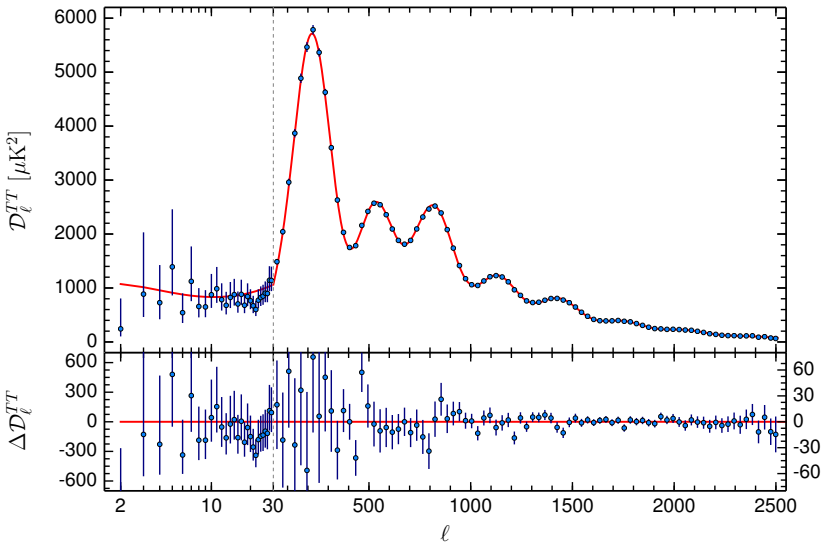


Figure 1.10: The Planck 2015 CMB power spectrum. The blue dots, with 1σ uncertainties, shows estmates of how strong each frequency is in the full-sky CMB shown in figure 1.1. The red line is the one predicted by the best-fit cosmological model, after 5 parameters have been fit. Figure from Planck Collaboration XIII (2015).

Cosmology is the science of how our universe — or all of space-time — has evolved since the Big Bang, some 13.8 billion years ago. According to Einstein's Theory of General Relativity (GR), the presence of energy and matter is coupled to space and time itself, and as the universe expands, space and energy and time evolve together. The current picture is that right after the Big Bang there was a very short period of exponential expansion (known as "inflation"), followed by a deceleration in the expansion of the universe, while about 5 billion years ago the expansion of the universe started to accelerate again.

The simplest model that currently best fits the available data is known as the ΛCDM model, where the energy of the universe is currently present in about 5% regular matter, about 26% cold dark matter (CDM), which is something we do not know what is and that we cannot see, but which interacts through gravity with regular matter; and finally about 70% dark energy (Λ), which is again something we do not know what is, but which has to be plugged in for the equations to match the data. The model is however dynamic; the same model predicts that right after Big Bang the majority of the energy was present in photons, not matter nor dark energy.

The ΛCDM model depends on six different parameters, including, e.g., the age of the universe. Given such parameters, it is possible to set up a set of differential equations that govern how matter and space-time evolves. Software such as CAMB (Lewis et al., 2000) can quickly solve these differential equations, leading directly to a prediction of what the CMB power spectrum should look like. The red line in figure 1.10 is such a power spectrum predicted from the parameters that best fit the data from Planck.

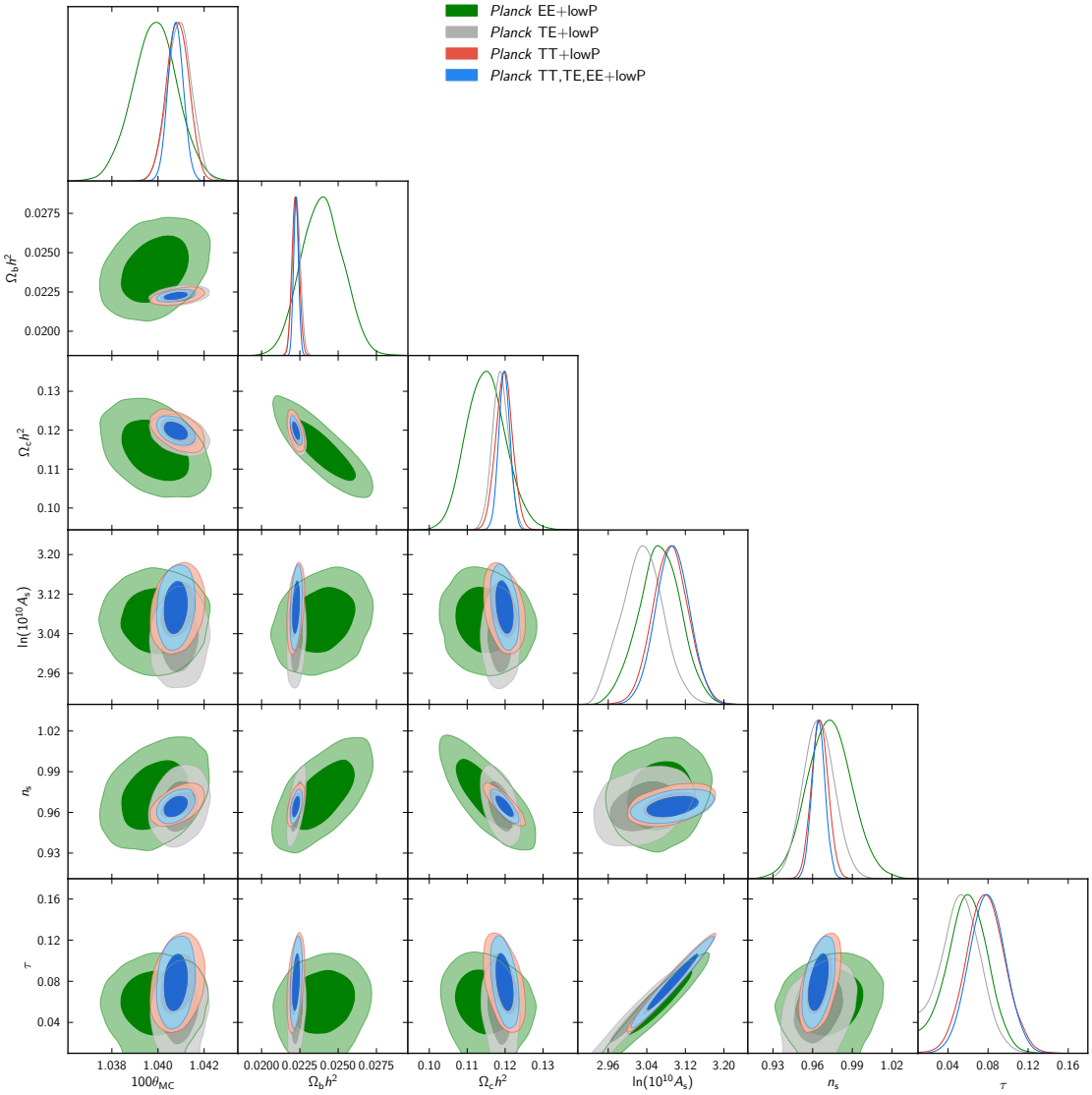


Figure 1.11: Constraints for the 6 cosmological parameters fitted in Planck 2015. Figure from Planck Collaboration XIII (2015).

We do not know up front what the values of the six parameters are, that is something we want to learn from the data. A convenient way to fit the parameters to the data is by using Monte Carlo Markov Chain (MCMC); CosmoMC (Lewis and Bridle, 2002) is a popular code for this purpose. Figure 1.11 summarizes such an MCMC run that fits the ΛCDM model to recent Planck data. In addition to best fit values and confidence intervals, we see whether our belief about a parameter should be strongly correlated with or independent from what we believe about another parameter.

1.6 Polarization

So far we have only discussed the radiation intensity of the microwave sky. However, there is additional data available. The photons arriving at the detectors has an orientation in a specific plane, and by using polarization filters one can detect this orientation. It turns out that different cosmological and astrophysical effects leaves signatures in the polarization.

Concretely, instead of representing the radiation on each position on the sky as a single temperature scalar T, we represent the radiation with a tuple (T, Q, U), where Q and U indicate the amplitude of polarization in different directions. These quantities are again related to the E power spectrum and the B power spectrum, which, given parameters, can be predicted by CAMB just like the temperature power spectrum. There are also "cross power spectra", so that we in total have six power spectra: TT, EE, BB, TE, TB, EB.

The E-mode power spectrum is actively used to constrain cosmological parameters. In figure 1.11 the different colors indicate which power spectra were used to constrain the parameters; note that TT+TE+EE (blue) has tighter constraints than TT alone (red).

The smaller scales ($\ell > 200$) of the B-mode power spectrum can be used to measure *lensing*, that is, to what degree the light has been warped on its way from the CMB background to us by the mass of clusters of galaxies. Several ground based telescopes, such as South Pole Telescope (SPT), Atacama Cosmology Telescope (ACT), Background Imaging of Cosmic Extragalactic Polarization (BICEP) and POLARBEAR, have observed this effect.

It has been predicted that the inflationary era have caused a signature of *gravitational waves* in the CMB. If it is there, it will take the form of a peak in the CMB BB power spectrum on large scales (low ℓ). None of the current telescopes are sensitive enough to detect this peak[1]. Detection of such a peak would be a major scientific discovery, and the chase for this discovery is the primary motivation behind the next generation of CMB telescopes.

Polarization has not been treated much in this thesis, but most of the results discussed generalize in a straightforward manner from working with only a temperature map to working with three scalar maps instead. For spherical harmonics there are special "spin-weighted" transforms which are used to transform (E, B) in pixel domain to (Q, U) in spherical harmonic domain; these are

[1] Much fuss was made in 2014 when BICEP2 claimed detection of gravitational wave signatures in the CMB, but this was later shown to be radiation from foregrounds (BICEP2/Keck Collaboration and Planck Collaboration, 2015).

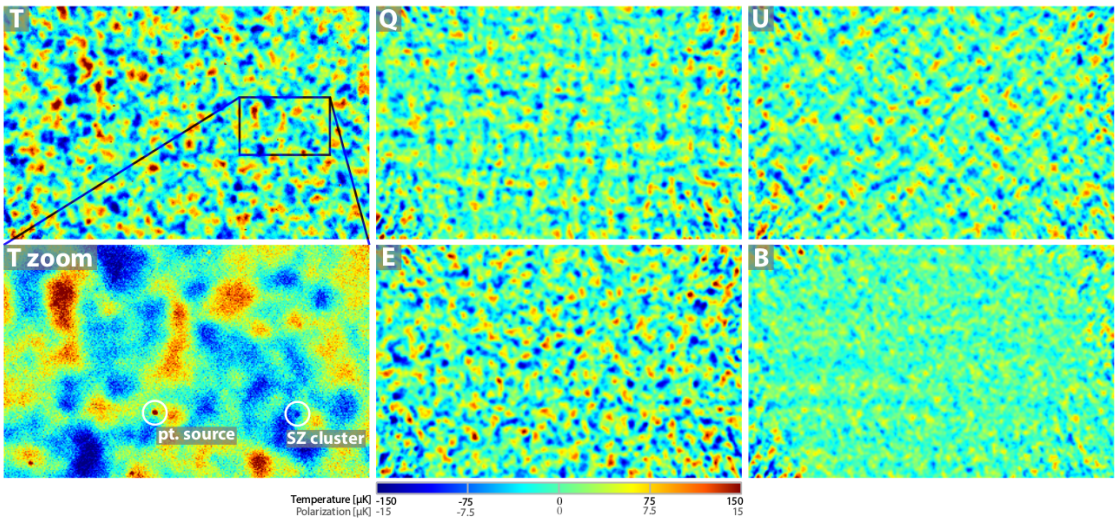


Figure 1.12: A view on the polarization in the CMB, as seen by the ACT Polarimeter (figure taken from Næss et al., 2014). The upper left pane (T) displays a small patch of the CMB sky as displayed in figure 1.1. The panes marked Q and U displays the polarization intensity in the way it is observed by telescope. The Q and U maps are orientation-specific and not very useful in this form; during analysis they are transformed into rotationally invariant E- and B-maps. The +-pattern in Q and ×-pattern in U is typical when there is more E-power than B-power.

computationally very similar to the regular SHTs. The *libsharp* code (Paper IV) has excellent support for spin-weighted SHTs, while *Wavemoth* (Paper III) has no such support.

Chapter 2

Bayesian CMB analysis with Commander

The idea behind the `Commander` code is to simply take our physical knowledge of the microwave sky as described in section 1.2, summarize that in a statistical model, and then solve whatever computational challenges that crops up.

Sections 2.1 through 2.3 provide an informal summary of the "Commander 1" paper by Eriksen et al. (2008), which built on the foundation laid by Eriksen et al. (2004), Jewell et al. (2004) and Wandelt et al. (2004).

A weakness of the Commander 1 approach is that it requires single resolution for all sky maps. This worked well enough for data from the WMAP mission, but in the case of Planck, the 30 GHz channel has a beam of nearly 33 arcminutes, compared to only 3.7 arc-minutes for the 857 GHz channel. Section 2.4 describes extension of the Commander framework to a joint analysis across different resolutions, dubbed "Commander 2". Finally, section 2.5 discusses priors on component amplitudes, including an example on using a Conditional Auto-Regressive (CAR) prior not previously published.

2.1 Basic Gibbs sampling

Review of Eriksen et al. (2008)

Let us first consider a single observation of the microwave sky on a single frequency sky, and that the CMB is the only physical component. That is, either we live in a universe where radiation from dust, synchrotron, etc. does not exist; or those components have been satisfactorily removed in a pre-processing step. Our model for the observed data then reads

$$\mathbf{d} = \mathbf{YBs} + \mathbf{n}, \tag{2.1}$$

where $\mathbf{d}$ denotes the observed data as a set of pixels on the sphere; $\mathbf{s}$ is the CMB component we seek and $\mathbf{n}$ is instrumental noise in each pixel. The instrumental noise is of course unknown, but we assume that we know its statistical

properties, and specifically that it is Gaussian with zero mean and a known diagonal covariance matrix $\mathbf{N}$.

Note that $\mathbf{s}$ is in fact a physical continuous field and has no relation to the pixelization chosen. Because we are primarily interested in its power spectrum, it is best to discretize $\mathbf{s}$ in spherical harmonic domain. This is done simply by saying that $\mathbf{s}$ is a vector of spherical harmonic coefficients $s_{\ell m}$ for ℓ less than or equal some L, above which we assume that all coefficients are zero; then argue that this truncation does not impact the solution for $\mathbf{s}$ on smaller scales. The signal $\mathbf{s}$ has a covariance matrix which we will denote $\mathbf{S}$. We will assume that the CMB is statistically isotropic and Gaussian, in which case $\mathbf{S}$ is diagonal in spherical harmonic domain with the CMB power spectrum C_ℓ on its diagonal.

As earlier mentioned, the telescope does not observe infinitely small points in the sky, but rather measures how much radiation is present within a beam (a "cone", although with varying density) that is moved across the sky. This has the effect of smoothing out the signal. As long as this smoothing effect is symmetric, which we typically assume, then the smoothing action is simply a diagonal matrix in spherical harmonic domain, and is written as $\mathbf{B}$ above. We could in principle also have applied it in pixel domain, and write $\mathbf{BY}$ rather than $\mathbf{YB}$, but the computational overhead would be significant.

Finally, $\mathbf{Y}$ represents the spherical harmonic synthesis, that is, the projection of the spherical harmonic coefficients to the observed pixels. Note that this projection does not need to be across the full sky – often one will apply a *mask* across some parts of the sky, and the corresponding pixels are then missing from $\mathbf{d}$ and $\mathbf{n}$ and the corresponding rows missing from $\mathbf{Y}$. For ground-based telescopes only patches of the sky are observed, so that most of the sky is masked out. For full-sky observations one may still wish to mask out particular regions of the sky where the modelling is not trusted to be an accurate representation of the truth.

Now, what we are interested in is not in fact an estimate of the CMB signal $\mathbf{s}$ as such, but the cosmological parameters, which we denote θ. These can be fit to the CMB signal by going through the computed power spectrum function $C_\ell(\theta)$, shown in figure 1.10.

So, what do we know about the universe we live in given the observed data? The Bayesian answer to that question is that we want to compute the *posterior distribution* for the power spectrum,

$$p(C_\ell|\mathbf{d}) \propto |\mathbf{S}(C_\ell) + \mathbf{N}|^{-1/2} e^{-\frac{1}{2}\mathbf{d}^T(\mathbf{S}(C_\ell)+\mathbf{N})^{-1}\mathbf{d}} \tag{2.2}$$

Note that while the density for $\mathbf{d}$ is Gaussian, but the posterior, as a function of C_ℓ, is very much non-Gaussian. We now hit a computational issue: The $\mathbf{S}$ matrix is diagonal in spherical harmonic domain, and the $\mathbf{N}$ matrix is diagonal in pixel domain, but their sum $\mathbf{S} + \mathbf{N}$ is dense in either domain. This means that simply storing the covariance matrix, at $O(N_{\text{pix}}^2)$, would require several petabytes of storage, and that running regular routines for dense linear algebra, at $O(N_{\text{pix}}^3)$, is a computational impossibility. While solving linear systems such as $(\mathbf{S}(C_\ell) + \mathbf{N})^{-1}\mathbf{d}$ is still possible by using iterative methods, computing the determinant $|\mathbf{S}(C_\ell) + \mathbf{N}|$ is firmly out of reach.

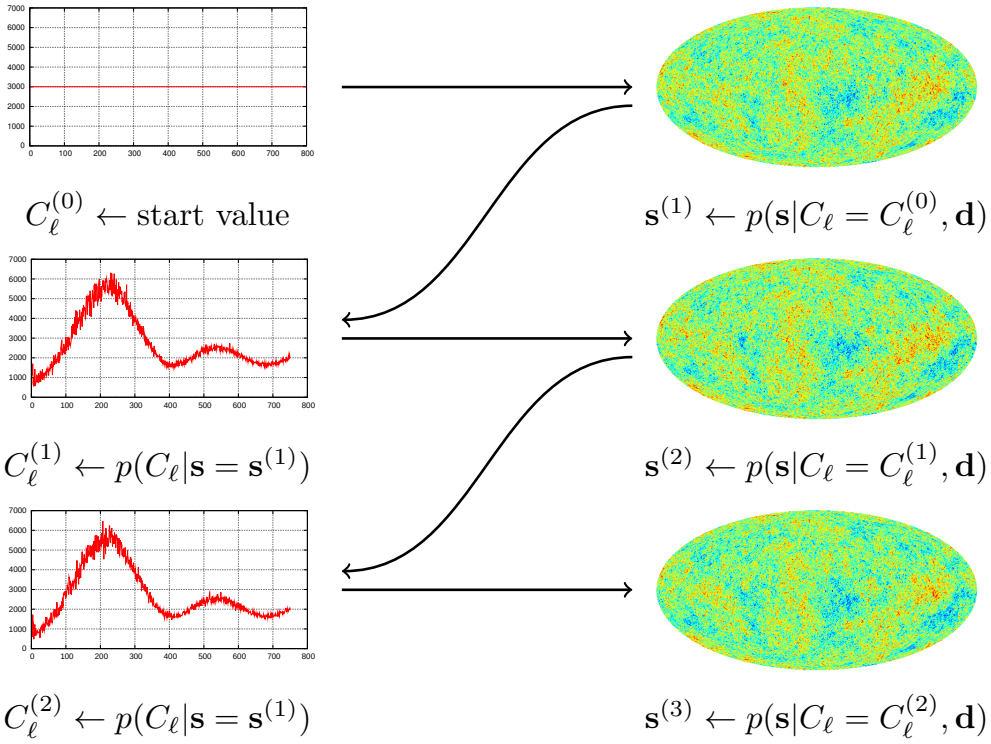


Figure 2.1: Gibbs sampling. We alternative between sampling $\mathbf{s}$ given C_ℓ and C_ℓ given $\mathbf{s}$. Notice that the different CMB signals are slightly different near the center (galactic plane), since a mask has been applied so that the signal is not constrained by data in this region. Outside of the mask the signal is almost completely constrained, variations between each sample is only on the order of the noise in each pixel.

We can, however, draw samples from the posterior by applying *Gibbs sampling*, a form of *Monte Carlo Markov Chain* (MCMC). The idea is to introduce the true CMB signal $\mathbf{s}$ as a proxy between the C_ℓ and the data, and alternate between sampling $\mathbf{s}$ and C_ℓ.

The distribution of $\mathbf{s}$ conditional on a cosmological model C_ℓ is given by

$$\begin{aligned} p(\mathbf{s}|\mathbf{d}, C_\ell = C_\ell^{(0)}) &\propto e^{-\frac{1}{2}(\mathbf{d}-\mathbf{YBs})^T\mathbf{N}^{-1}(\mathbf{d}-\mathbf{YBs})^T} e^{-\frac{1}{2}\mathbf{s}^T\mathbf{S}(C_\ell)^{-1}\mathbf{s}} \\ &\propto e^{-\frac{1}{2}\mathbf{s}^T(\mathbf{S}(C_\ell)^{-1}+\mathbf{B}^T\mathbf{Y}^T\mathbf{N}^{-1}\mathbf{YB})^{-1}\mathbf{s}} \end{aligned} \tag{2.3}$$

As a function of the signal $\mathbf{s}$ the distribution is Gaussian and the troublesome determinant can be elided for most purposes. Drawing a sample from this distribution is computationally tractable using methods described in the next section.

The distribution of C_ℓ conditional on a true CMB signal $\mathbf{s}$ is given by

$$p(C_\ell|\mathbf{d}, \mathbf{s} = \mathbf{s}^{(1)}) = p(C_\ell|\mathbf{s} = \mathbf{s}^{(1)}) \propto |\mathbf{S}(C_\ell)|^{-1/2} e^{-\frac{1}{2}\mathbf{s}^T\mathbf{S}(C_\ell)^{-1}\mathbf{s}}.$$

Note that if we already know the true CMB signal $\mathbf{s}$, the observed microwave data $\mathbf{d}$ can tell us no more about C_ℓ, allowing us to simplify the posterior. Since $\mathbf{S}$ is a diagonal matrix, this distribution can be written as an inverse Gamma distribution.

So, we have a distribution for $\mathbf{s}$ conditional on C_ℓ and for C_ℓ conditional on $\mathbf{s}$, creating somewhat of a circular dependency. This is where the Gibbs sampling algorithm comes into play. We start the process with some arbitrary power spectrum $C_\ell^{(0)}$, and then alternate between sampling $\mathbf{s}$ given C_ℓ and C_ℓ given $\mathbf{s}$, as illustrated in figure 2.1. According to MCMC theory, this process will converge to the true distribution, so that the $(C_\ell^{(i)}, \mathbf{s}^{(i)})$ samples (after an initial *burn-in* phase) comes from the joint posterior, and represents our knowledge given the data.

2.2 Constrained realizations of the CMB

Review of Eriksen et al. (2008)

A main focus of this thesis is the computational aspects of drawing samples of the true CMB signal $\mathbf{s}$ from $p(\mathbf{s}|\mathbf{d}, C_\ell)$. We will we refer these as *(data- and model-)constrained realizations*. According to equation (2.3), $p(\mathbf{s}|\mathbf{d}, C_\ell)$ is a Gaussian density, with mean vector and covariance matrix given by

$$\begin{aligned} \mathrm{E}(\mathbf{s}|\mathbf{d}, C_\ell) &= (\mathbf{S}^{-1} + \mathbf{B}\mathbf{Y}^T\mathbf{N}^{-1}\mathbf{Y}\mathbf{B})^{-1}\mathbf{Y}^T\mathbf{N}^{-1}\mathbf{d}, \\ \mathrm{Var}(\mathbf{s}|\mathbf{d}, C_\ell) &= (\mathbf{S}^{-1} + \mathbf{B}\mathbf{Y}^T\mathbf{N}^{-1}\mathbf{Y}\mathbf{B})^{-1}. \end{aligned}$$

If we had less data the natural way to proceed computationally is to form the covariance matrix and then employ the Cholesky factorization. In our case this approach is not computationally tractable. Instead our computations revolve around the following linear system:

$$\mathbf{A}\mathbf{x} \equiv \left(\mathbf{S}^{-1} + \mathbf{B}^T\mathbf{Y}^T\mathbf{N}^{-1}\mathbf{Y}\mathbf{B}\right)\mathbf{x} = \mathbf{b} \tag{2.4}$$

with

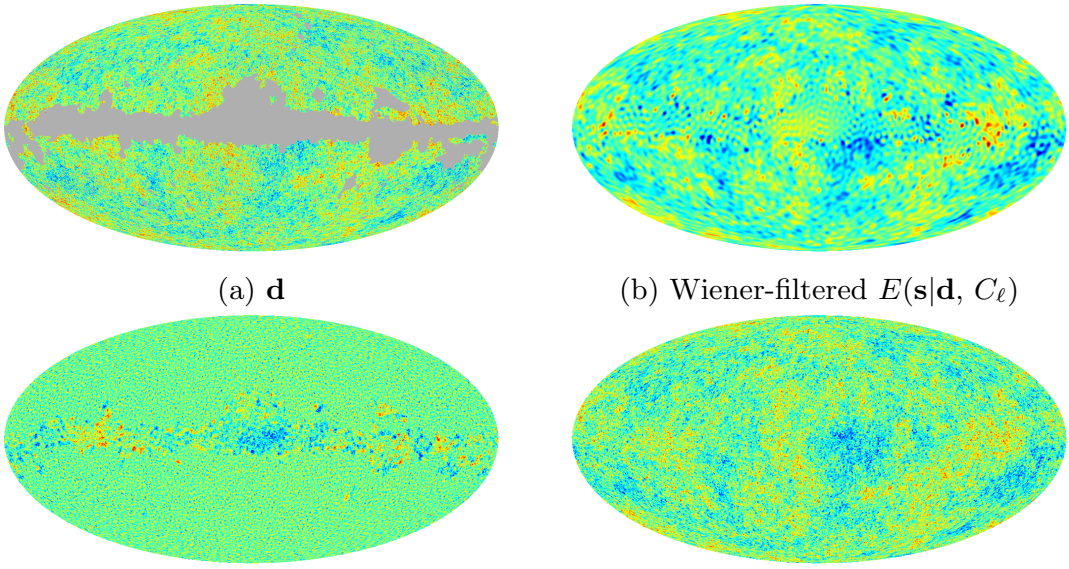
$$\mathbf{b} \equiv \mathbf{B}^T\mathbf{Y}^T\mathbf{N}^{-1}\mathbf{d} + \mathbf{B}^T\mathbf{Y}^T\mathbf{N}^{-1/2}\omega_1 + \mathbf{S}^{-1/2}\omega_2.$$

We will deal with two cases:

- If we let ω_1 and ω_2 be zero, the solution $\mathbf{x}$ will be the mean $\mathrm{E}(\mathbf{s}|\mathbf{d}, C_\ell)$, known as the *Wiener-filtered map*. This is particularly relevant in a component seperation setting.
- If we let ω_1 and ω_2 be vectors with coefficients drawn from the standard Gaussian distribution, the solution $\mathbf{x}$ will be a sample from $p(\mathbf{s}|\mathbf{d}, C_\ell)$.

Figure 2.2 illustrates the difference. Both results can be verified by simply computing $\mathrm{E}(\mathbf{x})$ and $\mathrm{Var}(\mathbf{x})$ in the expression above.

The matrix $\mathbf{A}$ of equation (2.4) is still huge; the improvement over conventional methods for working with the Gaussian distribution is that we do not need to factorize it (the linear algebra equivalent of taking the square root), but only need to solve a linear system. We can then apply *iterative solvers*, which do not require access to the elements of the matrix, but solve the system by performing a number of matrix-vector products $\mathbf{A}\mathbf{x}$. Such products can be efficiently computed as $\mathbf{A}$ consists only of diagonal matrices, scaling as $O(L^2)$, and spherical harmonic transforms, scaling as $O(L^3)$. A direct dense factorization would in contrast have a memory use of $O(L^4)$ and require $O(L^6)$ arithmetic operations.



(a) $\mathbf{d}$ (b) Wiener-filtered $E(\mathbf{s}|\mathbf{d}, C_\ell)$

(c) Random realization (d) Sample $\mathbf{s}$ from $p(\mathbf{s}|\mathbf{d}, C_\ell)$

Figure 2.2: (a) The observed CMB sky, where foregrounds have been removed using other methods, and a mask applied to regions we do not trust. (b) The *Wiener-filtered map* is the most likely signal. It looks "washed out" as it lacks power; this is analogous to how 0 is the single most likely value for a univariate standard Gaussian. (c) Random realization added to the Wiener-filtered map. In the region within the mask the signal is not constrained by data and consists of entirely random fill-in from the prior $p(\mathbf{s}|C_\ell)$. In the data-constrained region the realization is at the level of the instrumental noise in each pixel. (d) The sum of (b) and (c) is a realization of the underlying CMB signal, constrained by the data and a cosmological model C_ℓ.

2.3 Modelling foreground components

Review of Eriksen et al. (2008)

So far we have considered a single CMB component and an observation of the microwave sky at a single frequency. In the final model, we need to incorporate the foreground components as well. We do this by direct extension; in addition to $\mathbf{s}_{\text{cmb}}$, we also want to use Gibbs sampling to sample from the posterior of a thermal dust emission component $\mathbf{s}_{\text{dust}}$, a synchrotron emission component $\mathbf{s}_{\text{synch}}$, and so on. In general, we will refer to these as $\mathbf{s}_k$.

As discussed in section 1.2, the key to seperating the different components of the microwave radiation is to look at the frequency response of each component as seen in figure 1.4 (e). The spectral energy density (SED) of most components is assumed to be proportional to ν^β for some component-specific β. Note that β varies across the sky. For instance, figure 2.3 displays β for the low-frequency foregrunds as seen by Planck.

In order to avoid degenerate results it is important to provide sky maps on at least as many frequencies as we have components in the model. Thus a multi-component analysis is implicitly always a multi-frequency analysis too. To simplify notation we stack together the vectors for different sky maps and

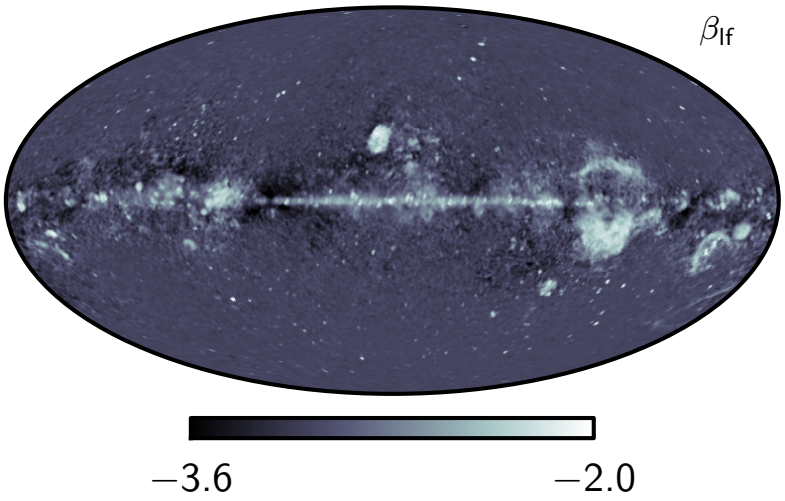


Figure 2.3: Example *spectral index map* taken from Planck Collaboration X (2015). This map is fitted to the joint WMAP, Planck and 408 MHz data assuming a single combined low-frequency component representing all of synchrotron, free-free and spinning dust. The spectral energy density is taken to be proportional to ν^{β}; see figure 1.4 (e).

.

components and write

$$\mathbf{d} \equiv \begin{bmatrix} \mathbf{d}_{30\mathrm{GHz}} \\ \mathbf{d}_{70\mathrm{GHz}} \\ \vdots \end{bmatrix} \qquad \mathbf{n} \equiv \begin{bmatrix} \mathbf{n}_{30\mathrm{GHz}} \\ \mathbf{n}_{70\mathrm{GHz}} \\ \vdots \end{bmatrix} \qquad \mathbf{s} \equiv \begin{bmatrix} \mathbf{s}_{\mathrm{cmb}} \\ \mathbf{s}_{\mathrm{dust}} \\ \vdots \end{bmatrix}.$$

An extension of the single-component model of equation (2.1) can now be written

$$\mathbf{d} = \mathbf{P}\mathbf{s} + \mathbf{n}, \tag{2.5}$$

where $\mathbf{P}$ is a block matrix that projects all of the components stacked in $\mathbf{s}$ to all of the sky maps stacked in $\mathbf{d}$:

$$\mathbf{P} = \begin{bmatrix} \mathbf{P}_{30\mathrm{GHz,cmb}} & \mathbf{P}_{30\mathrm{GHz,dust}} & \cdots \\ \mathbf{P}_{70\mathrm{GHz,cmb}} & \mathbf{P}_{70\mathrm{GHz,dust}} & \cdots \\ \vdots & \vdots & \end{bmatrix}.$$

We return to defining the projection operator of each component below. Further, we assume that each component $\mathbf{s}_k$ is assigned a Bayesian prior $p(\mathbf{s}_k)$ that is Gaussian, so that $\mathrm{Var}(\mathbf{s}) = \mathbf{S}$ forms a block-diagonal matrix of the prior covariance matrices $\mathbf{S}_k$. This prior is discussed further in section 2.5; as a simple starting point, one may consider C_ℓ on the diagonal of $\mathbf{S}_{\mathrm{cmb}}$, while the other components have infinite-variance priors, so that $\mathbf{S}_k^{-1} = \mathbf{0}$. Finally, $\mathbf{N} = \mathrm{Var}(\mathbf{n})$ is also a block-diagonal matrix of $\mathbf{N}_\nu$, since each frequency map is observed by seperate detectors with independent instrumental errors.

From here on we can proceed in the same way as in section 2.2. The joint posterior of the component vectors is a Gaussian,

$$p(\mathbf{s}|\mathbf{d}, \mathbf{S}) \propto e^{-\frac{1}{2}\mathbf{s}^T(\mathbf{S}^{-1}+\mathbf{P}^T\mathbf{N}^{-1}\mathbf{P})^{-1}\mathbf{s}}. \tag{2.6}$$

We explore the posterior by solving the linear system

$$\mathbf{A}\mathbf{x} \equiv \left(\mathbf{S}^{-1} + \mathbf{P}^T\mathbf{N}^{-1}\mathbf{P}\right)\mathbf{x} = \mathbf{P}^T\mathbf{N}^{-1}\mathbf{d} + \mathbf{P}^T\mathbf{N}^{-1/2}\omega_1 + \mathbf{S}^{-1}\omega_2, \tag{2.7}$$

where ω_1 and ω_2 are once again standard Gaussian variates to draw a sample from the posterior, or zero to find the posterior mean. Chapter 3 contains the computational details.

The projection of the CMB component is the same as the single-component projection in equation (2.1), so that

$$\mathbf{P}_{\nu,\mathrm{cmb}} = \mathbf{Y}_\nu \mathbf{B}_\nu.$$

For the foregrounds, the choice of projection $\mathbf{P}_{\nu,k}$ depends on whether we are working in a single resolution setup, dubbed "Commander 1", or a multi-resolution setup, dubbed "Commander 2". We will now proceed with the single-resolution setup of Eriksen et al. (2008), and return to multi-resolution in section 2.4.

Assuming a common beam $\mathbf{B}$ for all data $\mathbf{d}_\nu$, we then define the foreground components $\mathbf{s}_k$ as vectors of pixels after beam convolution. The CMB component $\mathbf{s}_{\mathrm{cmb}}$ is still defined as a vector of spherical harmonic coefficients before beam convolution. This choice of basis lets us use a simple diagonal matrix as the projection operator,

$$\mathbf{P}_{\nu,k} = \mathbf{Q}_{\nu,k}.$$

This *mixing matrix* $\mathbf{Q}_{\nu,k}$ defines the response of component k on frequency ν in each pixel; in most cases this is a diagonal matrix with $(\nu/\nu_{0,k})^{\beta_k}$ on the diagonal, for some reference frequency ν_0 for the component, and where β_k is a map that varies from pixel to pixel. Finally, it should be mentioned that sometimes templates are fitted to the maps in addition to diffuse foregrounds; e.g., for point sources, or for the mono- and dipole. In this case the component vector $\mathbf{s}_k$ is template amplitudes, and $\mathbf{P}_{\nu,k}$ contains the templates (Eriksen et al., 2008).

The missing piece at this point is the spectral index maps. One of the advantages of the Gibbs sampling framework is its flexibility. Additional effects in the model, whether physical or instrumental in nature, can simply be appended to the Gibbs cycle and sampled over[1]. So, as the β-maps are unknown, we append β-sampling as additional steps in the Gibbs cycle. If we have two foreground components, "dust" and "synch", a Gibbs cycle might look like this (ignoring C_ℓ and other model parameters):

$$\begin{aligned}
\mathbf{s}_{\mathrm{cmb}}^{(n+1)}, \mathbf{s}_{\mathrm{dust}}^{(n+1)}, \mathbf{s}_{\mathrm{synch}}^{(n+1)} &\leftarrow p(\mathbf{s}_k | \beta_k = \beta_k^{(n)}) \\
\beta_{\mathrm{dust}}^{(n+1)} &\leftarrow p(\beta_{\mathrm{dust}} | \mathbf{d}_{30\mathrm{GHz}}, \mathbf{d}_{70\mathrm{GHz}}, \ldots, \mathbf{s}_k = \mathbf{s}_k^{(n+1)}, \beta_{\mathrm{synch}} = \beta_{\mathrm{synch}}^{(n)}) \\
\beta_{\mathrm{synch}}^{(n+1)} &\leftarrow p(\beta_{\mathrm{synch}} | \mathbf{d}_{30\mathrm{GHz}}, \mathbf{d}_{70\mathrm{GHz}}, \ldots, \mathbf{s}_{\mathrm{k}} = \mathbf{s}_k^{(n+1)}, \beta_{\mathrm{dust}} = \beta_{\mathrm{dust}}^{(n+1)})
\end{aligned}$$

[1] The only problem being that if one introduces parameters that are too strongly correlated in the posterior, convergence will grind to a standstill.

When sampling such β-maps the idea is to take each band and subtract what we, at that point, assume is the contribution from the CMB and the *other* foreground components. The remainder is assumed to be the foreground of interest in isolation, only contaminated by instrumental noise. Furthermore we have conditioned on the amplitude of the component, and only sample β in isolation. Further details can be found in Eriksen et al. (2008).

2.4 Multi-resolution component separation

Paper II, Paper VII

While the basis and projection operator chosen for the signal components $\mathbf{s}_k$ in the previous section are convenient for implementation and computation, they require a common resolution on all input data. In reality, observations of the microwave sky have very different resolutions. For the analysis in the Planck 2015 release the resolution spanned from 3.7 arc minutes to 1 degree, measured in the full-width half-max (FWHM) of the beam.

In Commander 2 we change this so that the analysis is inherently multi-resolution. Recall that in the previous section we defined the projection operator as

$$\mathbf{P}_{\nu,k} = \mathbf{Q}_{\nu,k} \quad \text{(single-resolution)},$$

with foreground components $\mathbf{s}_k$ defined in pixel domain after beam convolution. We now instead define $\mathbf{s}_k$ as being the coefficients of the spherical harmonic expansion of the real, unconvolved foreground emissions, and define projection to the sky as

$$\mathbf{P}_{\nu,k} = \mathbf{Y}_\nu \mathbf{B}_\nu \widetilde{\mathbf{Q}}_{\nu,k} \quad \text{(multi-resolution)}. \tag{2.8}$$

Here, $\mathbf{B}_\nu$ contains the spherical harmonic transfer function specifically for the beam that applies to input sky map ν. Further the pixelization may differ, and with $\mathbf{Y}_\nu$ we denote projection to the specific pixelization in use for sky map ν. This means that for WMAP data, $\mathbf{Y}_\nu$ is using the $N_{\text{side}} = 512$ HEALPix grid, for Planck LFI the $N_{\text{side}} = 1024$ grid and for Planck HFI the $N_{\text{side}} = 2048$ grid.

The mixing matrix $\widetilde{\mathbf{Q}}_{\nu,k}$ is now defined in spherical harmonic domain. However, mixing (that is, point-wise multiplication) is something that is naturally defined in pixel domain, in the same manner as in the previous section, so we define

$$\widetilde{\mathbf{Q}}_{\nu,k} = \mathbf{Y}_k^T \mathbf{W}_k \mathbf{Q}_{\nu,k} \mathbf{Y}_k, \tag{2.9}$$

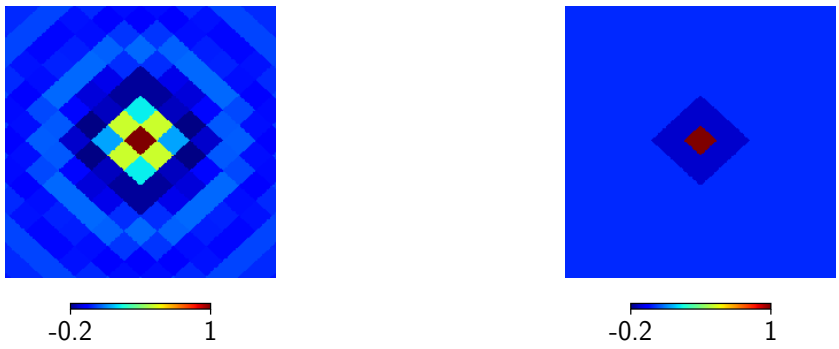
where $\mathbf{Y}_k$ denotes spherical harmonic synthesis to and $\mathbf{Y}_k^T \mathbf{W}_k$ denotes spherical harmonic analysis from an appropriate grid. The inner matrix $\mathbf{Q}_{\nu,k}$ is diagonal with each entry corresponding to the energy density of component k at frequency ν in a given position on the sky.

Further details on the multi-resolution component seperation model are given in Paper II.

2.5 Priors on component amplitudes

Paper II, Paper VII

CAR model example only published in this thesis



(a) $\mathbf{S}^{-1}$ (C_ℓ for CMB) (b) $\mathbf{S}^{-1}$ (CAR)

Figure 2.4: The inverse prior covariance $\mathbf{S}^{-1}$ visualized in pixel domain. Priors may vary in their specific shape, but what they will have in common is a negative ring around a positive peak, which penalizes solutions whose smoothness properties are not as specified by the prior. The square root matrix $\mathbf{S}^{-1/2}$, present in equation (2.10), also has this basic shape. (a) Prior for CMB component assuming a ΛCDM power spectrum. (b) Pixel-domain CAR prior as discussed in section 2.5

In our model each signal component vector, $\mathbf{s}_{\text{cmb}}$, $\mathbf{s}_{\text{dust}}$, and so on, has an associated prior covariance matrix $\mathbf{S}_k$. The role of this matrix is to couple the value of the component in each pixel to the values of the surrounding pixels, i.e., it represents a prior on the *smoothness* of the component. Loosely speaking, without a prior, more noise will be explained as being part of the signal. With a prior, we assume a given smoothness in the resulting components, and abberations that cause a departure from this smoothness will to a stronger degree be explained as noise. The tendency to take the prior into account depends on the noise characteristics – in regions with very low instrumental noise the component posterior will lie close to the data, in regions with more instrumental noise the component posterior will lie closer to the prior. If a mask is applied to the data, then the use of a prior is required, and the prior alone takes effect inside the masked-out region.

This is most easily seen if equation (3.1) is re-formulated as a *weighted least squares* problem. Solving equation (3.1) for $\mathbf{x}$ is equivalent to finding the $\mathbf{x}$ that minimizes

$$\left\| \begin{bmatrix} \mathbf{N}^{-1/2}\mathbf{P} \\ \mathbf{S}^{-1/2} \end{bmatrix} \mathbf{x} - \begin{bmatrix} \mathbf{N}^{-1/2}\mathbf{b} + \omega_1 \\ \omega_2 \end{bmatrix} \right\|. \tag{2.10}$$

Now, consider the shape of $\mathbf{S}^{-1/2}$ in pixel domain; it looks similar to the $\mathbf{S}^{-1}$ operators given in figure 2.4, with a negative ring around a positive peak. If $\omega_1 = \omega_2 = 0$, it is clear from the expression above that the prior impose a penalty to abrupt changes in $\mathbf{x}$ when finding the least squares solution. How severe this penalty is, and how strong it is on different harmonic scales, depends on the prior. If ω_1 and ω_2 are drawn randomly as described above, then the least squares solution will be penalized to the degree that the smoothness of the solution does not match $\mathbf{S}$.

As mentioned above, in the case of CMB Gibbs sampling, where we primarily seek the CMB power spectrum, we initialize the CMB "prior" C_ℓ to a

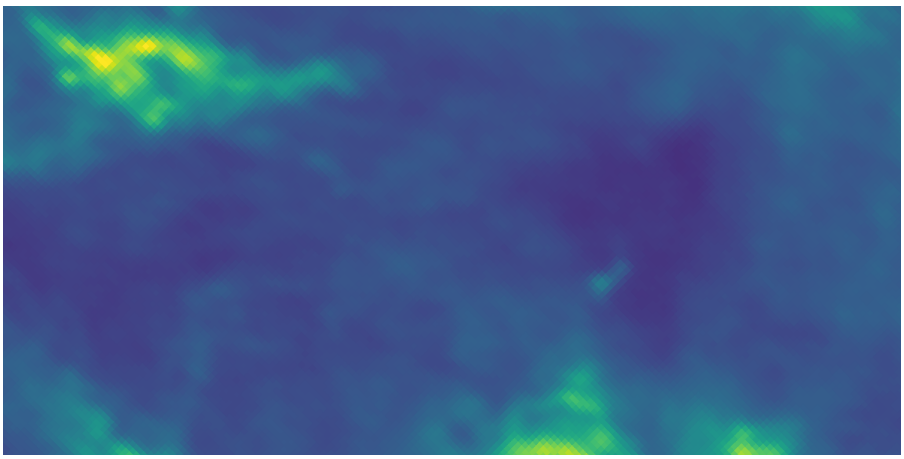
neutral starting point, then iterate between sampling the CMB signal and C_ℓ. That is, $\mathbf{S}$ is only the prior if considering one sub-step of the Gibbs chain in isolation.

For component seperation, where we primarily want to produce accurate maps of both the CMB and the foreground components, we ideally do not want to specify any priors. Especially for the CMB we typically leave it zero; except perhaps gradually introduce smoothness at very small scales that are noisy. For the foreground components there is a difference between regions on the sky. In regions with a high amount of foreground radiation, the signal amplitude so dominates the instrumental noise that the solution is essentially data-driven no matter what the prior is. In regions with less foreground radiation a prior can be used to ensure that one defaults to a smooth field; it can be argued that, in the lack of a high signal, it is better to estimate using a smooth field through the region than to fit what we know is essentially instrumental noise.

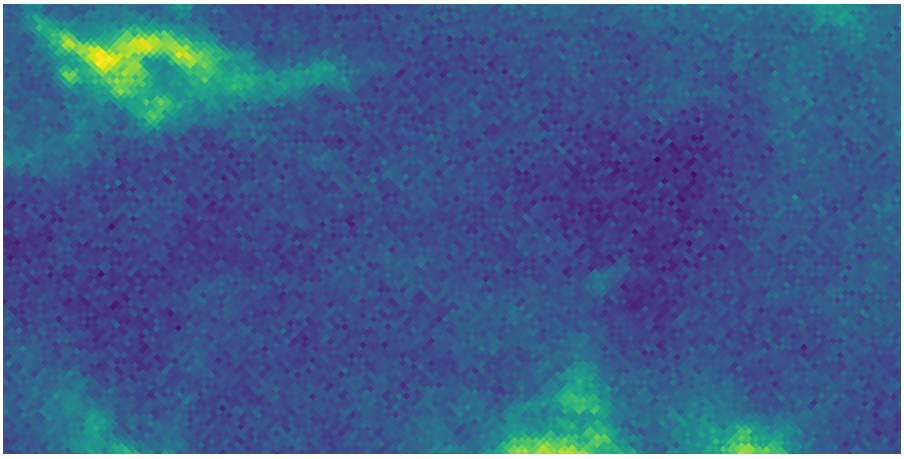
The currently implemented computational methods, including the state of the art algorithm in Paper II, assume that the priors are isotropic; that is, that the smoothness of the component is the same everywhere on the sky. In this case a typical foreground prior will be estimated from how the component acts for low ℓ (where the signal is strong) and extrapolated to high ℓ (small scales where the signal is weak). This procedure for instance gives a prior for thermal dust power spectrum of about $C_{\text{dust},\ell} \propto \ell^{-2.5}$, and the $\mathbf{S}_{\text{dust}}$ covariance matrix is then defined as a diagonal matrix with this power spectrum on its diagonal.

In the future, an attractive alternative is the Conditional Auto-Regressive (CAR) model, which allows the prior to vary between different regions of the sphere. Let s_i denote some pixel of the foreground component and $r(s_i)$ denote a weighted average of the pixels in a neighbourhood around s_i. Then, we define the prior in each pixel conditional on its neighbourhood, $p(s_i|r(s_i))$, as a Gaussian with mean $r(s_i)$ and some standard deviation τ_i. Each pixel has its own standard deviation τ_i, making this a much more flexible model than the isotropic power spectrum. See figure 2.5 for a demonstration, and, e.g., Cressie and Wikle (2011) for more information on the CAR model.

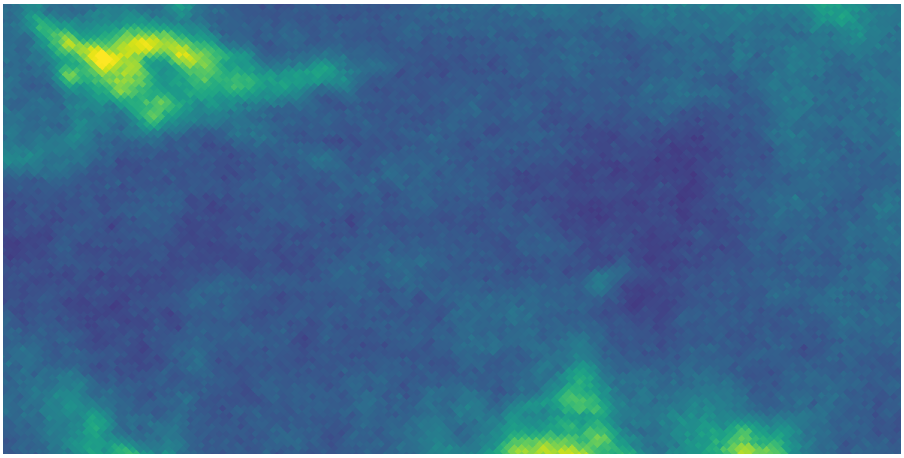
Computationally, CAR is a convenient model because it leads directly to the construction of a sparse inverse covariance matrix $\mathbf{S}^{-1}$. Computationally this is the biggest hurdle to solving the CR equation of (3.1). The other required piece is the computation of $\mathbf{S}^{-1/2}\omega_2$ for the right-hand side. Here one could either i) use MCMC methods to draw a random vector $\mathbf{S}^{1/2}\omega_2$ by iterating through pixels and draw each pixel from the prior conditional on its neighbours, and then compute $\mathbf{S}^{-1}\mathbf{S}^{1/2}\omega_2$, or ii) use a CAR-like specification to define $\mathbf{S}^{-1/2}$, and let $\mathbf{S}^{-1}$ be defined implicitly as the product $\mathbf{S}^{-1/2}\mathbf{S}^{-1/2}$.



(a) True thermal dust radiation



(b) Synthetic input data: True radiation plus instrumental noise



(c) Reconstruction using only input data and CAR prior

Figure 2.5: Example of the effect of a prior. In this case we have attempted reconstruction the thermal dust from a *single* frequency band. Without a prior there is nothing to go on in this case, and the best estimate would simply be the input data itself. By adding a prior stating that we assume the underlying emission to be a smooth field, some of the noise can be "removed". In this case we have used a CAR prior, stating that every pixel is expected to be the mean of its neighbouring pixel plus a random normal variate with with standard deviation of τ_i. We have constructed the prior such that τ_i is larger in regions with high emission (again using the input map, but smoothed with a Gaussian low-pass filter). Thus the prior adds more stabilization in regions where the noise is stronger relative to the emission, while still giving the component enough flexibility to climb up to the "hills".

Chapter 3

Preconditioning the CR system

The compute time of the Commander code is dominated by drawing constrained realizations; that is, by solving the linear system of equation (2.7), which we will refer to as the *CR system*. The main challenge in my PhD has been to develop better algorithms for this problem.

Paper II presents the result of this endeavour: An algorithm for solving the CR system that is intuitive, easy to implement, and very fast, providing a solution of the CR system in between 5 and 40 iterations of the algorithm. This chapter will provide some more intuition behind the CR system and how to build solvers for it, as well as describe the relationship between Paper I and Paper II. Finally another partial success is presented for the first time in section 3.4 (not submitted to a journal).

An implementation of some of the algorithms presented here can be found in the `CMBCR` code[1]. The code is experimental in nature and serves as a testbed for development of new algorithms. CMBCR runs on a single node only, only supports temperature analysis, and is written in a mixture of Python, Cython and Fortran. For production analysis within the Planck project we instead use the `Commander 2` code[2]. Commander 2 supports polarization, is distributed using MPI, is implemented in pure Fortran, and includes many other features beyond solving the CR system.

3.1 Iterative linear solvers

To recap, our system reads

$$(\mathbf{S}^{-1} + \mathbf{P}^T\mathbf{N}^{-1}\mathbf{P})\mathbf{x} = \mathbf{P}^T\mathbf{N}^{-1}\mathbf{d} + \mathbf{P}^T\mathbf{N}^{-1/2}\omega_1 + \mathbf{S}^{-1/2}\omega_2, \tag{3.1}$$

which we will write simply as

$$\mathbf{A}\mathbf{x} = \mathbf{b}.$$

[1] https://github.com/dagss/cmbcr/
[2] https://github.com/hke/commander/

Computationally we are not able to store all of the elements of this matrix, let alone employ routines for dense linear algebra. We are however able to efficiently apply the matrix to a vector, by applying the terms from the right to the left: First compute $\mathbf{Px}$, then $\mathbf{N}^{-1}(\mathbf{Px})$, and so on.

Luckily there is an entire field of research devoted to solving linear systems simply by multiplying with the system matrix. The fundamental idea is to guess a starting vector, say, $\mathbf{x}_1 = \mathbf{0}$. Then, for each iteration, a *residual* $\mathbf{r}_i = \mathbf{b} - \mathbf{A}\mathbf{x}_i$ is computed, which is used to produce an updated iterate $\mathbf{x}_{i+1}$ that lies closer to the true solution.

Note that the residual, which is readily available, is used as a proxy for the *error*, $\mathbf{e}_i = \mathbf{x}_{\text{true}} - \mathbf{x}_i$, which is unavailable as we do not know $\mathbf{x}_{\text{true}}$. The key is that since $\mathbf{A}\mathbf{x}_{\text{true}} = \mathbf{b}$,

$$\mathbf{r}_i = \mathbf{b} - \mathbf{A}\mathbf{x}_i = \mathbf{A}(\mathbf{x}_{\text{true}} - \mathbf{x}_i) = \mathbf{A}\mathbf{e}_i,$$

and since $\mathbf{A}$ is linear, reducing the magnitude of $\mathbf{r}_i$ will also lead to a reduction in the error $\mathbf{e}_i$. The aim of a good iterative solver is to reduce the magnitude of the residual as quickly as possible. If the residual becomes close enough to zero we have the solution (or one of the solutions) to the system.

In a production run the error $\mathbf{e}_i$ is naturally unavailable, but during development and debugging it is highly recommended to track it. This can be done by generating some $\mathbf{x}_{\text{true}}$, then generate $\mathbf{b} = \mathbf{A}\mathbf{x}_{\text{true}}$, and track the error $\mathbf{e}_i$ while running the solver on this input. The error has better information than the residual about where convergence is slow and where convergence is quick. If convergence lags behind in some specific region in the sky, or on some particular scales ℓ, then this lag is immediately obvious from inspecting the error, and less so by looking at the residual.

Since $\mathbf{A}$ in our case is symmetric and positive definite, the recommended iterative solver is the *Conjugate Gradients* (CG) method (see Shewchuk, 1994, for a tutorial). The number of iterations required depends on how clustered the eigenspectrum of the matrix is. There are a couple of rules of thumb to gauge convergence. The ratio of the highest eigenvalue divided by the lowest eigenvalue is known as the *condition number*, and the higher this number is the more iterations are required. Furthermore it helps to have eigenvalues clustered in regions with relatively flat behaviour; CG then uses a few iterations for each such cluster. In exponentially decaying parts of the spectrum the solver might almost break down, requiring one iteration for each coefficient in the solution vector.

Usually $\mathbf{A}$ has a condition number that makes the application of iterative solvers directly intractable. The trick is to come up with a *preconditioner* $\mathbf{M}$ that in some sense approximates the inverse matrix $\mathbf{A}^{-1}$, and then solve the transformed system $\mathbf{MAx} = \mathbf{Mb}$ instead. A good preconditioner is a symmetric, positive definite matrix $\mathbf{M}$ such that $\mathbf{Mx}$ can be quickly computed, and where the condition number of $\mathbf{MA}$ is low.

A large chunk of my research consisted of trying to come up with better preconditioners / linear solvers for the matrix $\mathbf{A}$ of equation (3.1). Preconditionering is the cornerstone of efficient solvers for partial differential equations (PDEs), and many relevant papers on preconditioners only considers them in

the context of such systems. However, most methods are fairly general and much of the PDE literature can be employed in statistical data analysis as well.

Our system can be re-written in other forms, which is a useful exercise since some of the literature on image processing and iterative solvers uses these variations. We already touched on formulating it as a weighted least squares problem in equation (2.10) in section 2.5. Another formulation is the *saddle-point problem*,

$$\begin{bmatrix} \mathbf{N} & \mathbf{P} \\ \mathbf{P}^T & -\mathbf{S}^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{y} \\ \mathbf{x} \end{bmatrix} = \begin{bmatrix} \mathbf{d} + \mathbf{N}^{1/2}\omega_1 \\ \mathbf{S}^{-1/2}\omega_2 \end{bmatrix}, \tag{3.2}$$

where $\mathbf{x}$ is the solution we seek and $\mathbf{y}$ is an auxiliary vector we solve for and then ignore. Note that it really does say $\mathbf{N}$ instead of $\mathbf{N}^{-1}$, and that $\mathbf{P}^T\mathbf{N}^{-1}$ is implicitly applied to the top block of the right-hand side. This is called a saddle-point problem because the eigenspectrum is split in two; one positive part and one negative part. The formulation does not really bring us any further, because a good preconditioner for a saddle-point system should embed an approximation to its *Schur complement*, and the Schur complement of the matrix above is again the matrix in equation (3.1). Much has been written about approximating Schur complements in this context. Benzi et al. (2005) contains an extensive review on the numerical solution of such systems, and much of what is said there applies to our system. In particular, it mentions the use of pseudo-inverses, which was the crucial clue that led to the state of the art solver presented in Paper II.

3.2 A closer look on A

Paper I, Paper II, Paper VII

We write the block representation of the basic multi-resolution component seperation system:

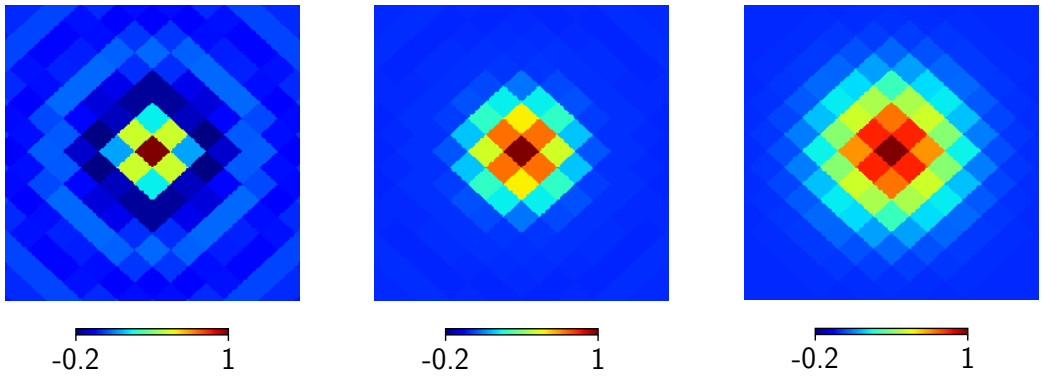
$$\mathbf{A}_{k,k'} = \mathbf{S}^{-1}_{k,k'} + \sum_{\nu} \mathbf{Q}^T_{\nu,k}\mathbf{B}^T_{\nu}\mathbf{Y}^T_{\nu}\mathbf{N}^{-1}_{\nu}\mathbf{Y}_{\nu}\mathbf{B}_{\nu}\mathbf{Q}_{\nu,k'}. \tag{3.3}$$

We describe the operators present in turn:

Prior term $\mathbf{S}^{-1}$

The prior term was discussed in more detail in section section 2.5. If and only if there is no mask defined in the system, so that the second term above in the system matrix above is non-singular, the prior term is optional. In Paper I and Paper II we assume an isotropic prior, in which case the $\mathbf{S}^{-1}$ operator represents full sky convolution; the convolution kernel in pixel domain is shown in figure 3.1 (a).

By its nature, the prior term takes effect in regions where the data fails to effectively constrain the solution. In regions with very low instrumental noise it does not have much effect on the solution, whereas in areas with high



(a) $\mathbf{S}^{-1}$ (C_ℓ) (b) $\mathbf{B}$ (c) $\mathbf{B}^T\mathbf{N}^{-1}\mathbf{B}$

Figure 3.1: Some of the operators that make up $\mathbf{A}$, visualized in pixel domain on an arbitrary scale. Each plot show the effect that the operator has when applied to a single-pixel unit vector, corresponding to the single column of the pixel-domain operator matrix. The prior shown in (a) take the form of a ring of negative coefficients around a positive peak. This is similar in nature to the Laplacian arising in elliptical PDEs, where multi-level solvers are very effective in solving the system in pixel-domain. In (c) we hold the inverse-noise map constant at unity so that $\mathbf{B}^T\mathbf{N}^{-1}\mathbf{B} = \mathbf{B}^T\mathbf{B}$, for demonstration purposes. The inverse-noise matrix $\mathbf{B}^T\mathbf{N}^{-1}\mathbf{B}$ never has negative coefficients, and inverting it by itself represents deconvolution of a low-pass filter, which is difficult to do efficiently in pixel domain. In principle all of these operators may vary based on location on the sphere; in practice modelling and approximations often make $\mathbf{B}$ and $\mathbf{S}^{-1}$ invariant to location, while $\mathbf{N}^{-1}$ varies.

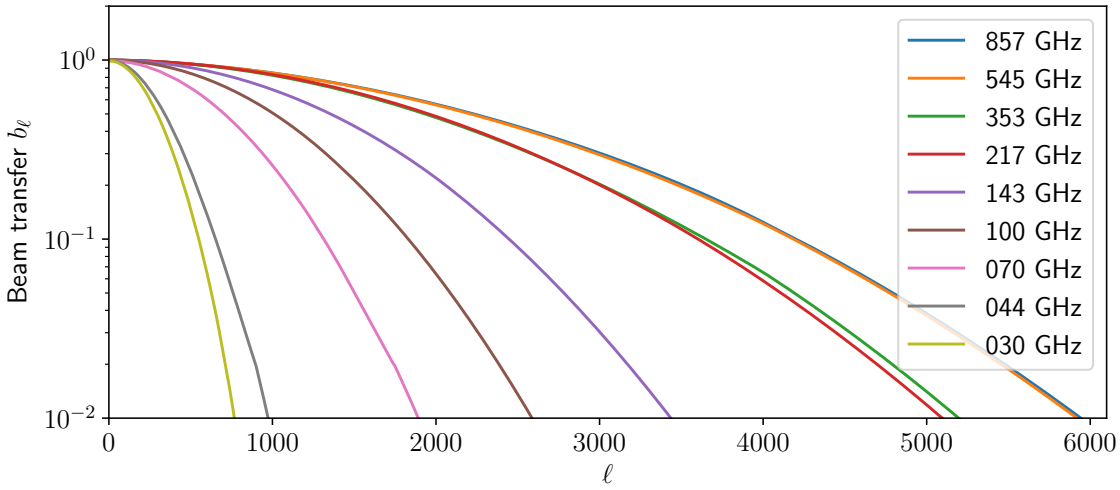


Figure 3.2: Spherical harmonic transfer functions b_ℓ for the Planck sky maps (diagonal of matrices $\mathbf{B}_\nu$). Note that the inverse-noise term $\mathbf{B}^T\mathbf{N}^{-1}\mathbf{B}$ behaves as b_ℓ^2 in harmonic domain.

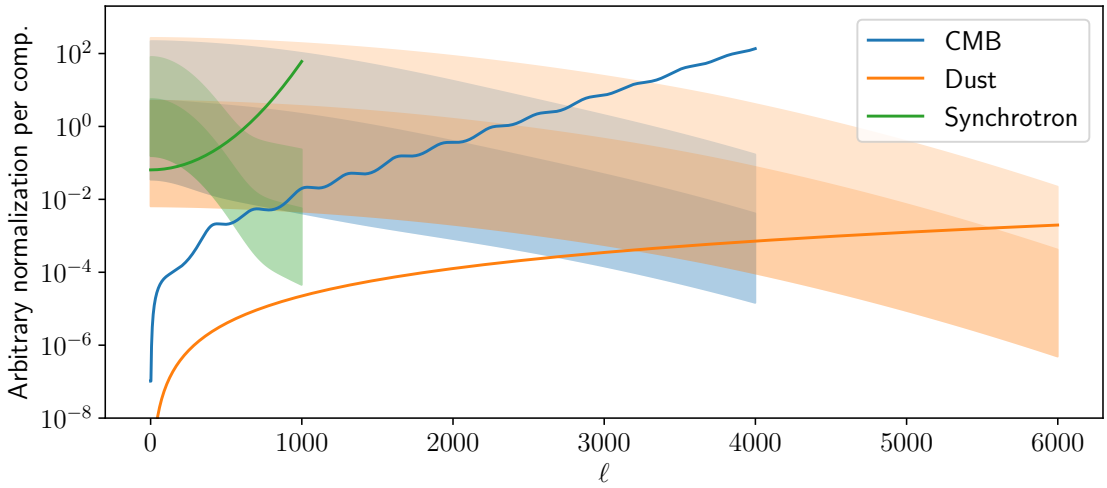


Figure 3.3: The significance of the prior term versus the inverse-noise term (full-sky) as a function of scale. Plotted are the coefficients of the prior matrix $\mathbf{S}^{-1}$ (solid lines) and the inverse-noise matrix $\sum_\nu \mathbf{Q}_{\nu,k}^T \mathbf{B}_\nu^T \mathbf{Y}_\nu^T \mathbf{N}_\nu^{-1} \mathbf{Y}_\nu \mathbf{B}_\nu \mathbf{Q}_{\nu,k}$ (lighter bands between minimum and maximum values). Adding small amounts of synthetic noise to the 1% least noisy pixels in the RMS map significantly reduces the coefficient spread while not really affecting the statistical analysis (darker bands between minimum and maximum values). The bands follow the trajectory of a weighted sum over $b_{\nu,\ell}^2$, with $b_{\nu,\ell}$ as displayed in figure 3.2, and the weights determined by the mixing maps.

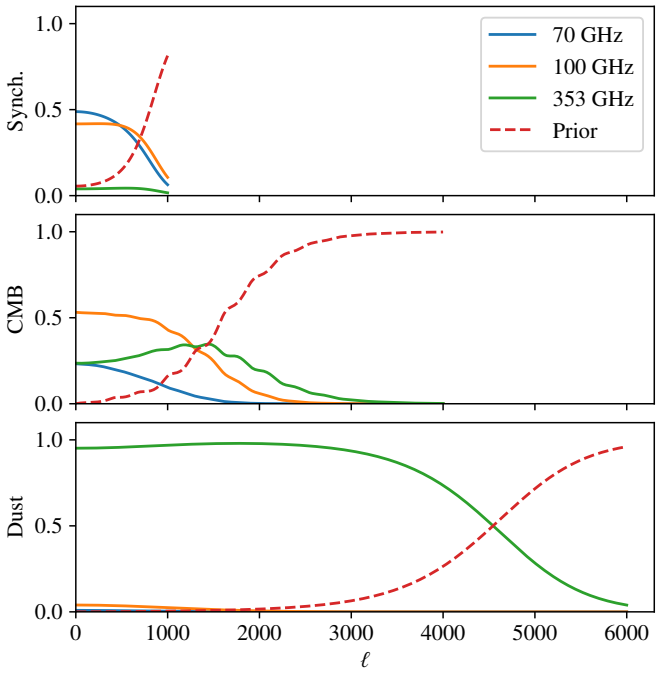


Figure 3.4: Visualization of the effect of the mixing matrices $\mathbf{Q}_{\nu,k}$ and how they interact with the beam matrices $\mathbf{B}_\nu$. For each component k we plot the magnitude (averaged over spatial position) of each of the terms that makes up the $\mathbf{A}$ matrix in equation (3.3); normalized so that the the sum is 1 for each ℓ. Note how the CMB has most support from the 100 GHz band for low ℓ, then gradually switches to the 353 GHz band and finally the prior as ℓ increases. In this demonstration only 3 frequencies were included for clarity; a full analysis of Planck data has 9 frequencies.

instrumental noise it can be the dominating term. If parts of the sky is masked out and one still seeks to predict a signal in this region, the prior term stands alone in this region of the sphere. Furthermore the prior plays a strong role on the smallest scales (high ℓ), and a weak role on the largest scales (low ℓ). Figure 3.3 shows typical diagonals of $\mathbf{S}^{-1}$ in spherical harmonic domain (solid lines), and how they compare in magnitude with the inverse-noise term for each ℓ.

Instrumental beam convolution B

The instrumental beams are also currently modelled as isotropic convolution operators, and so, like $\mathbf{S}^{-1}$, act as full sky convolutions. A pixel-domain example is given in figure 3.1 (b); more relevant for the solution of equation (3.3) is the shape of the squared beam in figure 3.1 (c).

While both $\mathbf{B}$ and $\mathbf{S}^{-1}$ are rotationally and locationally invariant full sky convolutions, the are very different operators. The prior term is a high-pass filter, whereas the beam is a low-pass filter whose pixel domain convolution kernel only takes positive values.

Inverse-noise matrices $\mathbf{N}_\nu^{-1}$

The role of this term is to propagate the effect of instrumental noise to the solution. It allows us to assign different weight to different pixels in the data vectors $\mathbf{d}_\nu$, so that pixels with more noise gets less weight. The inverse-noise matrix $\mathbf{N}_\nu^{-1}$ itself is assumed to be diagonal in pixel domain in our case, so that measurement noise is assumed to not be correlated between neighbouring pixels, this is simply an approximation for computational reasons.

The diagonal of $\mathbf{N}_\nu^{-1}$ is given by the inverse of the square of the RMS map. A typical RMS map in use for Planck is shown in figure 1.7. The RMS map primarily follows the scanning pattern of the telescope. The inverse-noise matrix is the primary driver behind spatial variability in the system. Consider again figure 3.3 — in the noisiest regions the CMB component is constrained mainly by the prior instead of the data already at $\ell \sim 900$, whereas in regions with less noise the data constrains the CMB solution all the way to $\ell \sim 2800$[3].

Mixing matrices $\mathbf{Q}_{\nu,k}$

The mixing matrices are present in a multi-component, multi-observation setup to make it possible to seperate the components. Often, such as for the CMB component, these just represents multiplication by a scalar. For other components such as thermal dust, they are derived from spectral index maps such as the one in figure 2.3.

Either way, they interact with the b_ℓ seen in figure 3.2 to produce a matrix representing the weight that should be given to each sky map. This weight varies as a function of ℓ, so that, e.g., the larger scales of the CMB component is

[3]This discusses constraining the CMB component signal in isolation on different parts of the sky. Constraining the CMB power spectrum C_ℓ is a different question.

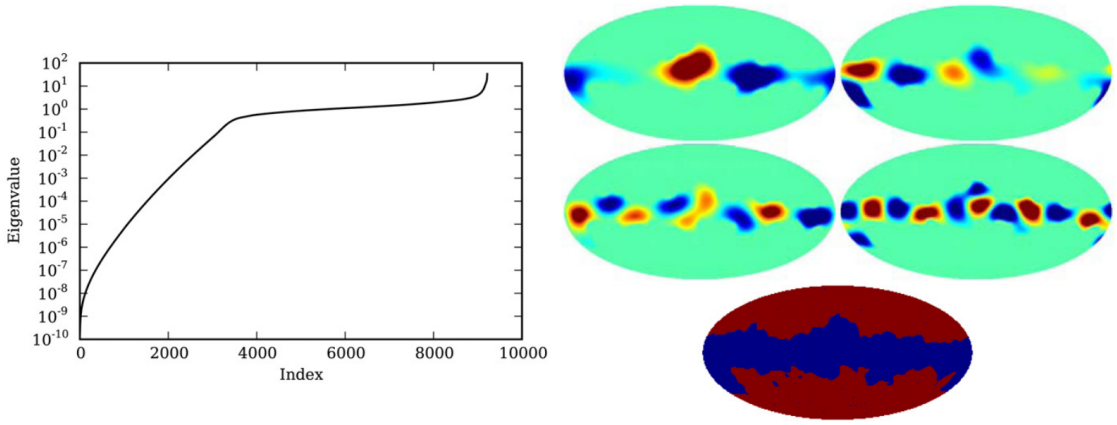


Figure 3.5: Eigendecomposition of the CR system using a diagonal preconditioner. The top pane displays the eigenvalues in sorted order, while in the middle we plot four eigenvectors corresponding to very low eigenvalues. They all build up the large scales within the mask. The CG algorithm would need to spend very many iterations to pin down these eigenvectors. See Paper I for further details.

first supported by low-frequency data, and the smaller scales by high-frequency data. Figure 3.4 visualizes this effect, ignoring spatial variability in the system.

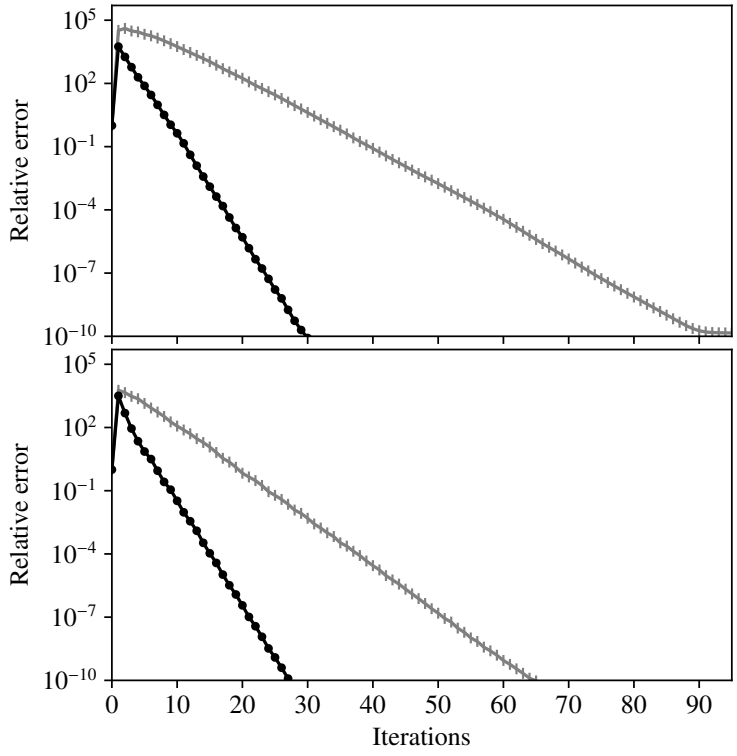
Mask

So far we have mainly discussed each operator as acting on the full sky. In many practical applications, we additionally want to mask out parts of the sky, either because of missing data, or because we do not trust our statistical model in a given region of the sky. Within such scenarios, it is useful to distinguish between two very different cases:

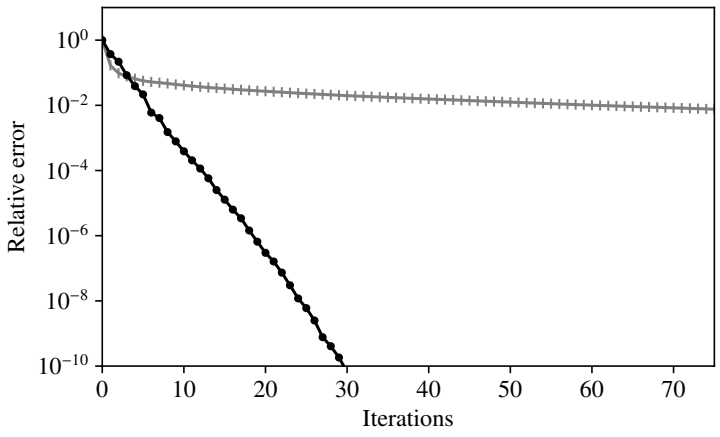
i) Partial sky coverage, where only a small patch of the sky has been observed, and we wish to perform component separation only within this patch. The typical use-case is for this setup is ground-based or sub-orbital CMB experiments. This case is in many ways very similar to the full sky case, but has been outside the scope of the work in this thesis.

ii) Natively full-sky coverage, but too high foregrounds in a given part of the sky to trust our model. In this case, one often masks out part of the sky, but still seeks a solution to the system under the mask, constrained by the observed sky at the edges of the mask and determined by the prior inside the mask. By ignoring data from this region we at least avoid that the CMB component is contaminated by foreground emission. Of course, the solution will not be the true CMB sky either, but it will have statistically correct properties for use inside of a Gibbs sampler (Jewell et al., 2004; Wandelt et al., 2004; Eriksen et al., 2004, 2008).

In Paper I the mask was introduced in the $\mathbf{N}_\nu^{-1}$ operator, while in Paper II it was moved to the $\mathbf{Q}_{\nu,k}$ operator. Either way, it causes high contrasts in $\mathbf{A}$ which must be taken into account when designing the preconditioner.



(a) Full-sky component seperation



(b) Constrained realization with a mask

Figure 3.6: Benchmark results from Paper II, which describes the best solver developed in this thesis. We plot the the error $\|\mathbf{x} - \mathbf{x}_{\rm true}\|$ as a function of the number of CG iterations, using the preconditioner developed in Paper II (solid circles) and a simple diagonal preconditioner (gray ticks). (a) Solving the CR system for the purposes of full sky component seperation, using 3 components and 9 sky maps at their individual resolutions. In the top plot the RMS map has higher contrasts than in in the bottom plot, corresponding to the wide and narrow bands for $\mathbf{N}^{-1}$ in figure 3.3, respectively. (b) Solving the CR system for the purposes of constrained realization in a Gibbs chain, applying a mask to the galactic center. This case is much harder for the diagonal preconditioner, which essentially fails to converge, while the method in Paper II is hardly affected. See Paper II for further details.

3.3 Preconditioning strategies for the CR system

Paper I, Paper II

The conceptually simplest preconditioner for $\mathbf{A}$ would be to use a subset of the coefficients of the matrix $\mathbf{A}$ in spherical harmonic domain; in particular the diagonal. Section 3.4 covers this technique and presents a moderate improvement on these preconditioners which has not been submitted as a seperate paper.

The problem with preconditioning in spherical harmonic domain is that one becomes blind to spatial variations. These are most severe in the presence of a mask. Figure 3.5 shows how the large modes within the mask results in a poorly conditioned system. The effect is that the spherical harmonic preconditioners fails to converge in high-resolution setups. Even in the full sky case, the inhomogenities in the RMS maps slows down these preconditioners significantly.

To get around this problem, Paper I introduced a pixel-domain solver. The focus is on the solution of the prior term $\mathbf{S}^{-1}$ in pixel domain. In isolation, the convolution kernel of $\mathbf{S}^{-1}$ is rather similar to the operator in Laplacian partial differential equations, and this similarity carries over to preconditioning. In short, for efficient pixel domain preconditioning one has to apply multi-grid (MG) methods, and when using such methods the resulting convergence is very good.

The method in Paper I breaks down if de-convolution of the instrumental beam must be performed. Consider figure 3.3 again. The pixel-domain MG method is only able to find a solution where the inverse-noise term has still not decayed significantly (below about 10% of the maximum value). Luckily, in the case of Planck, this is the regime where the prior term crosses over and starts to dominate the system. Paper I was therefore able to use the pixel-domain multigrid method for $\ell < 2200$, and a spherical harmonic preconditioner for $\ell \geq 2200$ — since this high-ℓ regime is dominated by the prior, it does not hurt that spatial variations are not taken into account. This combined method resulted in excellent convergence; however, a model with a single CMB component fitted against the Planck dataset is just barely within the regime where the solver will work. If there had been less instrumental noise at the same angular resolution, or less angular resolution (wider beam) at the same noise level, then the instrumental-noise band in figure 3.3 would have been able to decay to, say, about 0.01 before it was overtaken by the prior. That would have left a range of ℓs without an efficient preconditioner. For the same reason, solving for the thermal dust component using this method is not possible, at least without specifying a prior which smoothens out the smallest scales of the dust component. These restrictions of the method were only realized after Paper I was published. In addition, the solver of Paper I is rather complex and requires significant compute and memory resources for pre-computed data (although the spherical grid described in Paper V reduces the cost of the precomputations significantly).

This is where Paper II picks up. It introduces three new elements:

- First it introduces a so-called *pseudo-inverse* preconditioner which essentially works in spherical harmonic domain, but includes a very good

approximation to the spatially varying effects, as long as the contrasts in the RMS map and mixing maps are reasonable.

- A pixel-domain multi-grid method is used *only* in the domain under the mask, in contrast to Paper I which used the same method on the full sky. As a result it does not matter that the MG method is unable to deconvolve the instrumental beam; as there is no data to deconvolve under the mask.

- The mask was moved from the inverse-noise operator to the component mixing operator. This is a numerical measure which significantly reduces the ringing problems encountered in Paper I.

The resulting method is robust against different priors and instrumental models, and is also significantly easier to implement. The amount of pre-computation required is very modest.

Paper II goes into the concrete details of this solver. To complement this paper, the following section 3.4 gives an introduction to preconditioning in spherical harmonic domain, and develops a *banded preconditioner*. Informal tests indicate that this preconditioner converges up to twice as fast as the diagonal preconditioner for the Planck scanning pattern. The pseudo-inverse preconditioner of Paper II converges in as many or fewer iterations, is more versatile as it does not depend on the scanning pattern, and is much easier to implement. Still, the preconditioner developed in section 3.4 could potentially have an edge in some future experiment as it does not require expensive SHTs as part of the preconditioner, and so I publish it as part of this thesis. In theory, employing a seperate multi-grid solver within the mask can be combined with either the pseudo-inverse solver, or the banded spherical harmonic solver, although no numerical experiments have been carried out for the latter combination.

3.4 Preconditioning in spherical harmonic domain

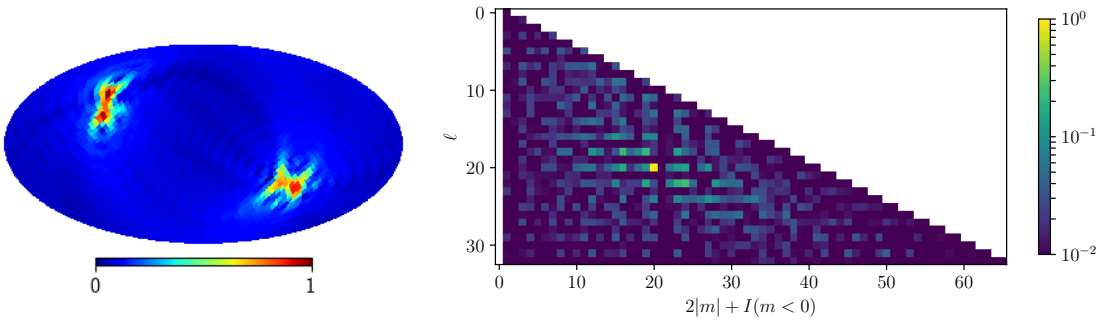
Published for the first time in this thesis

The simplest preconditioner is the diagonal preconditioner, $\mathbf{M} = \mathrm{diag}(\mathbf{A})^{-1}$, and we then need individual elements of the matrix. There are two ways to compute these:

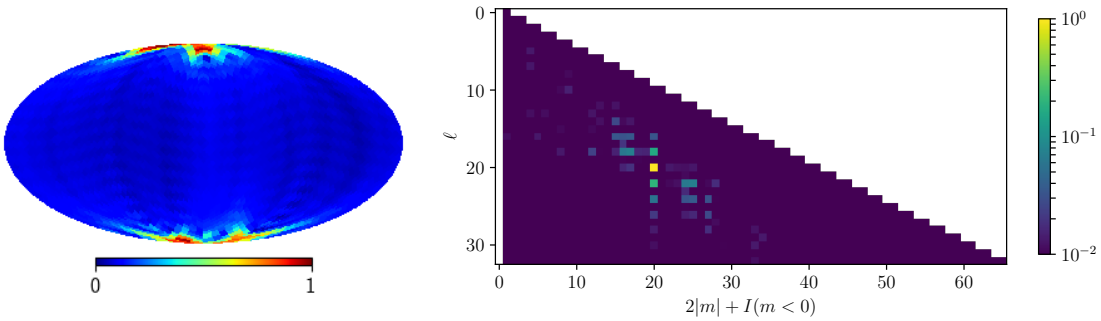
Sum over associated Legendre functions This method is given in equation (19) in Paper I, and is accurate to numerical precision. Routines that compute the necessary associated Legendre functions are included in the latest version of Libsharp[4].

Sum over Wigner 3j-symbols This method is given in Hivon et al. (2002), and represents an approximation, as the spherical grid is neglected and it is assumed that the operator is continuous. It is popular as the `drc3jj` routine for computing Wigner 3j symbols has been easily available since 1975.

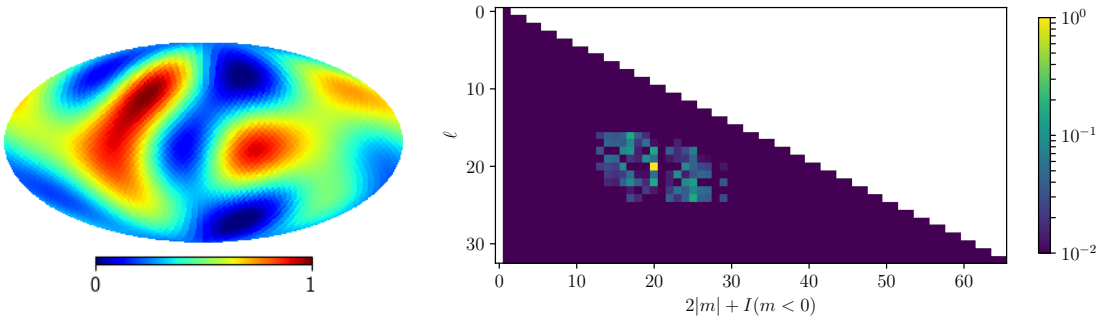
[4] Available from https://github.com/dagss/libsharp



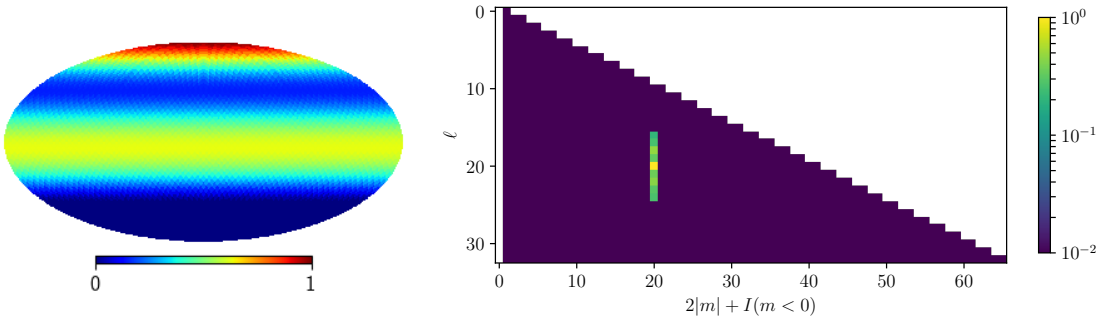
(a) $\mathbf{N}^{-1}$ for Planck 100 GHz, galactic coordinates



(b) $\mathbf{N}^{-1}$ for Planck 100 GHz, ecliptic coordinates



(c) Random $\mathbf{N}^{-1}$ constrained by $\tau(\hat{n}) \geq 0$ and $\tau_{\ell m} = 0$ for $\ell > 4$



(d) Random $\mathbf{N}^{-1}$ constrained by $\tau(\hat{n}) \geq 0$ and $\tau_{\ell m} = 0$ for $m \neq 0$, $\ell > 4$

Figure 3.7: Exploring $\widehat{\mathbf{N}}^{-1} \equiv \mathbf{Y}^T \mathbf{N}^{-1} \mathbf{Y}$. The left panels display the diagonal of $\mathbf{N}^{-1}$, scaled so that the maximum value is 1. The right panels display a single column (or row) from the corresponding matrix $\widehat{\mathbf{N}}^{-1}$; specifically, $|\widehat{\mathbf{N}}^{-1}_{\ell m, \ell' m'}| / \widehat{\mathbf{N}}^{-1}_{\ell m, \ell m}$ for $\ell' = 20$, $m' = 10$ (bright yellow). Note the logarithmic scale; we are primarily concerned with the relation between the diagonal element and other matrix elements in terms of orders of magnitude. The patterns displayed are largely the same independent of which column is chosen. That is, the main dependence of matrix element magnitude is on $|\ell - \ell'|$ and $|m - m'|$, while ℓ and m play a smaller role.

Both methods require $O(L)$ arithmetic operations to compute a single element of the matrix, and the difference is slight on high resolution grids. The latter method provide us with some analytical insights, and so we repeat it here. Let τ be the diagonal of $\mathbf{N}^{-1}$, viewed as a map, and further let $\tau_{\ell,m}$ be the expansion of this map to spherical harmonics. Then we have

$$(\mathbf{Y}^T\mathbf{N}^{-1}\mathbf{Y})_{\ell m,\ell' m'} \approx \frac{N_{\text{pix}}}{4\pi}(-1)^m \sum_{\ell''} \tau_{\ell'',|m-m'|} Y^{\ell,\ell',\ell''}_{m,m',|m-m'|}, \tag{3.4}$$

where $Y^{\ell,\ell',\ell''}_{m,m',m''}$ denotes the *Gaunt integral*, a normalized product of two Wigner 3j-symbols. The Gaunt integral has several useful symmetry and orthogonality properties (Hivon et al., 2002, and references therein). Now, the bandwidth of $\mathbf{A}$, that is, how far from the diagonal elements are non-zero, depends on the inverse-noise map τ. The relationship is illustrated in figure 3.7. Analytically, we have:

Along ℓ: The Gaunt integral vanishes unless $|\ell - \ell'| \leq \ell''$. If τ is smooth / band-limited so that $\tau_{\ell'',m''} = 0$ for $\ell'' < \ell^*$ then $|\mathbf{A}_{\ell m,\ell' m'}| = 0$ whenever $|\ell - \ell'| > \ell^*$, according to equation (3.4).

Along m: If τ only varies in the latitudinal direction, that is, it is azimuthally symmetric around the poles, we have that $\tau_{\ell,m} = 0$ for $m \neq 0$. From equation (3.4) it is then straightforward to see that $|\mathbf{A}_{\ell m,\ell' m'}| = 0$ if $m \neq m'$, so that $\mathbf{A}$ becomes block-diagonal matrix in ℓ-major ordering.

More generally, the first step of spherical harmonic analysis is to compute a set of Fourier transform along iso-latitude rings (see section 4.1). If the signal in τ along all such rings is smooth / band-limitedm so that the Fourier coefficients $q_m = 0$ for all $m > m^*$ and $|\mathbf{A}_{\ell m,\ell' m'}| = 0$ if $|m - m'| > m^*$, according to equation (3.4). Note that this results holds even in the presence of abrupt changes and small scale fluctuations in the latitudinal direction.

These results also hold in less absolute terms; if $\tau_{\ell,m}$ decays by an order of magnitude for large ℓ or m then $|\mathbf{A}_{\ell m,\ell' m'}|$ also decays by an order of magnitude as one moves away from the diagonal, outside of the band indicated. Also note that if τ is entirely flat one can show that $\mathbf{A} \approx (N_{\text{pix}}/4\pi)\tau\mathbf{I}$, where the quality of the approximation depends on the spherical grid.

Figure 3.7 was generated by computing $\mathbf{Y}^T\mathbf{N}^{-1}\mathbf{Y}\mathbf{u}$ for some unit vector $\mathbf{u}$ using SHTs, so the figure serves simultaneously as numerical verification and a demonstration of the results given above. Note the effect of rotating the τ-map in such a way it is mostly azimuthally symmetric in panel (b); elements where $|m - m'| \neq 0$ to have low magnitude relative to the diagonal. Also elements with odd $|\ell - \ell'|$ have low magnitude, due to the north/south symmetry in τ.

Now, a natural way to construct a preconditioner is to ignore elements that are small relative to the diagonal. The sparsity in panel (b) in figure 3.7 looks promising, and we construct a *banded preconditioner* based on this. First, we rotate the system into ecliptic coordinates, as described below. Then, we include in our preconditioner $\mathbf{M}^{-1} \approx \mathbf{A}$ all elements where $|m - m'| = 0$,

$|\ell - \ell'| \leq 8$, and $|\ell - \ell'|$ is even; ignoring other elements of $\mathbf{A}$. The resulting approximant $\mathbf{M}^{-1}$ is both banded and block-diagonal, which makes for a very efficient preconditioner, as it can be parallelized over m, and LAPACK contains routines for working directly on banded representations of matrices.

When applying the preconditioner to the multi-component equation (3.1), we interleave the coefficients from each component together, so that every element in a single-component $\mathbf{A}$ turns into an $N_{\rm comp} \times N_{\rm comp}$ block. Repeating equation (3.3), the system then also includes the mixing matrix for each component, with the block between component k and k' given by

$$\mathbf{A}_{k,k'} = \mathbf{S}^{-1}_{k,k'} + \sum_{\nu} \widetilde{\mathbf{Q}}_{\nu,k} \mathbf{B}_\nu \mathbf{Y}_\nu^T \mathbf{N}_\nu^{-1} \mathbf{Y}_\nu \mathbf{B}_\nu \widetilde{\mathbf{Q}}_{\nu,k'}.$$

The mixing maps are relatively flat, so we simply approximate $\widetilde{\mathbf{Q}}_{\nu,k'}$ with a scalar. Another idea would be to make an approximation where we exchange the order of $\widetilde{\mathbf{Q}}$ and $\mathbf{B}$ in the equation above, so that the mixing matrices can be multiplied directly with the inverse-noise maps, but this approximation performed worse in our tests.

The Planck data files are given in galactic coordinates, and we now turn to rotating the system into ecliptic coordinates. It is important at this point to stress that we do *not* want to rotate the inverse-noise map in HEALPix basis, as the HEALPix grid contains important information about how the TOD data has been binned together within each pixel. Rotation is most easily carried out in spherical harmonic domain, as the Wigner d-matrices can be used to transform the spherical harmonic coefficients directly to another coordinate system (see Paper IV and references therein)[5]. We denote the rotation operator from ecliptic to galactic as $\mathbf{R}$, with $\mathbf{R}^T = \mathbf{R}^{-1}$, so that the system in ecliptic coordinates is written

$$\mathbf{R}^T \mathbf{A} \mathbf{R} \mathbf{x} = \mathbf{R}^T \mathbf{b}.$$

The prior and beam matrices are rotationally invariant, so that

$$\mathbf{R}^T (\mathbf{S}^{-1} + \mathbf{B}\mathbf{Y}^T \mathbf{N}^{-1} \mathbf{Y} \mathbf{B}) \mathbf{R} = \mathbf{S}^{-1} + \mathbf{B}\mathbf{R}^T \mathbf{Y}^T \mathbf{N}^{-1} \mathbf{Y} \mathbf{R} \mathbf{B}$$

The HEALPix (HP) grid is however not rotationally invariant. The solution is to first resample the HEALPix inverse-noise map τ to a Gauss-Legendre grid following the procedure in Paper I and Paper V. This procedure takes care of accurately emulating the HEALPix pixelization effects on the Gauss-Legendre (GL) grid, and the resulting map can then be safely rotated:

$$\tau_{\text{GL-Rot}} = \mathbf{Y}_{\rm GL} \mathbf{W}_{\rm GL} \mathbf{R} \mathbf{Y}_{\rm HP}^T \tau_{\rm HP}.$$

We have verified that

$$\mathbf{R}^T \mathbf{Y}^T \mathbf{N}^{-1}_{\rm HP} \mathbf{Y} \mathbf{R} = \mathbf{Y}^T \mathbf{N}^{-1}_{\text{GL-Rot}} \mathbf{Y} \tag{3.5}$$

[5]HEALPix contains the `rotate_alm` routine for this purpose. Unfortunately, no MPI-distributed implementation has been written; writing such an implementation is feasible but non-trivial, beacuse the rotation treats coefficients of the same ℓ together, while the spherical harmonic coefficients are distributed between nodes by m (see section 4.3)

to relative error about 10^{-13} (typical of SHTs); thus we can construct the system

$$\mathbf{A}_{\text{rot}}\mathbf{x} = \mathbf{R}^T\mathbf{b}$$

without any rotation operators on the left hand side, and compute $\mathbf{M}_{\text{rot}}^{-1} \approx \mathbf{A}_{\text{rot}}$. We still have a choice about where to apply the rotation:

Rotating the preconditioner Solve $\mathbf{A}\mathbf{x} = \mathbf{b}$ preconditioned with $\mathbf{R}^T\mathbf{M}_{\text{rot}}\mathbf{R}$, calling `rotate_alm` as part of the preconditioner for every iteration in the CG search.

Rotating the system Solve $\mathbf{A}_{\text{rot}}\mathbf{x} = \mathbf{R}^T\mathbf{b}$ directly, using the inverse-noise map resampled to Gauss-Legendre grid

The two methods are numerically equivalent. Which approach performs better will depend on the concrete resolution parameters and relative performance of rotations vs. SHTs. For a low-resolution analysis the latter approach will always win.

The performance and potential use-cases for this preconditioner were discussed at the end of section 3.3.

Chapter 4

Spherical harmonic transforms

The solvers for the Constrained Realization system described in the previous chapter are, computationally speaking, recipes for kicking off a large number of spherical harmonic transforms (SHTs) in sequence. A good preconditioner can help us reduce the number of SHTs required, but in the end a Commander analysis is dominated by executing SHTs. A 2x speedup in the SHTs will be close to a 2x speedup in a Commander analysis.

Of course, SHTs have many other uses in many fields of science. Studying them and optimizing them is a worthy pursuit in its own right. Fourier transforms are very well developed, with several heavily optimized high-performance libraries available. SHTs are much less mature in this respect. The libraries in production use have a computational scaling of $O(L^3)$ for a spherical harmonic band-limit L; faster algorithms have however been devised, and is an active field of research.

4.1 Wavemoth: Fast SHT by matrix compression

Paper III

My work on SHTs started out with Paper III, which is summarized below. The spherical harmonic basis functions are given by

$$Y_{\ell m}(\theta, \phi) \equiv \tilde{P}_\ell^m(\cos\theta) e^{im\phi}, \tag{4.1}$$

where $\tilde{P}_\ell^m$ denotes the *associated Legendre functions*. Note that it is neatly seperated into one part that depends on the latitude θ and another that depends on the longitude ϕ.

The associated Legendre functions we can treat mostly like a black box. They can be written down in closed form, and are almost (but when m is odd, not quite) polynomials of degree up to ℓ, and since ℓ can be in the thousands for our usecase, computing these polynomials directly would be both expensive

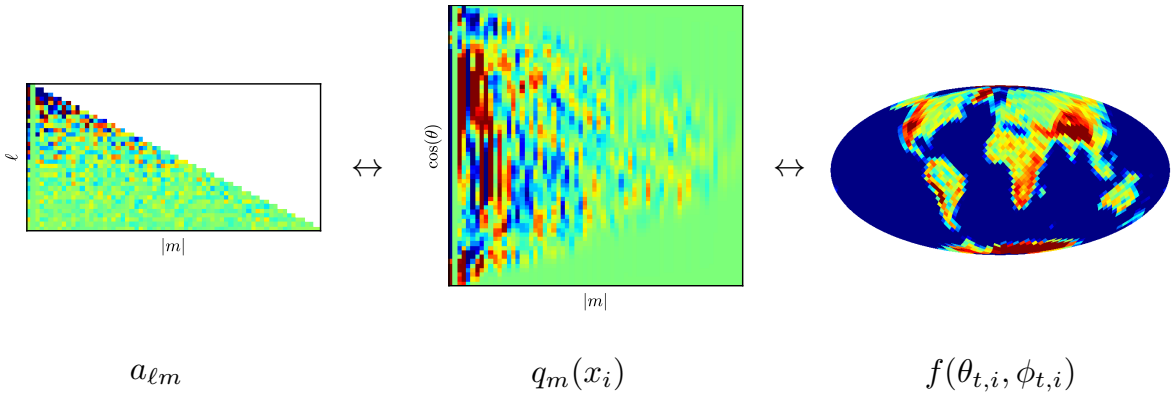


Figure 4.1: Separating the sums in an SHT. A Legendre transform is used on each column m to go between $a_{\ell m}$ (left) and the ring-phases $q_m(x_i)$ (center), while a Fourier transform is used on each ring to go between $q_m(x_i)$ and the spherical field $f(\theta_{k(i)}, \phi_i)$ (right).

and numerically tricky. Instead we apply recurrence relations that have been studied for their numerical stability; in essence relations on the form

$$\tilde{P}_\ell^m(z) = k_1(\ell, m)\tilde{P}_{\ell-1}^m(z) + k_2(\ell, m)\tilde{P}_{\ell-2}^m(z), \tag{4.2}$$

where k_1 and k_2 are short closed-form expressions. So, we just need to evaluate $\tilde{P}_0^m(z)$ and $\tilde{P}_1^m(z)$ as polynomials, and then we can compute the rest of the values using a recurrence relation (there are other relations that makes a step in m instead of ℓ, or two steps in ℓ, and so on).

Now, the spherical harmonic synthesis from equation (1.1) can be written

$$f(\theta_i, \phi_i) = \sum_{\ell=0}^{L} \sum_{m=-\ell}^{\ell} a_{\ell m}\tilde{P}_\ell^m(\cos\theta_i)e^{im\phi_i}, \tag{4.3}$$

where we truncate at some L. On the surface of it it takes $O(L^2)$ arithmetic operations to compute this sum. For a full transform, we compute the sum for N pixels, and usually $N = O(L^2)$, so we say that the brute-force SHT scales as $O(L^4)$. Now, the first trick to reduce this is to make sure that the $O(L^2)$ grid points are organized in $O(L)$ *rings* of constant latitude θ. Beyond this basic scaling relationship, each ring doesn't need to have the exact same number of grid points (some grids do, others do not). Now we can split the computation in two parts. First we do a set of *associated Legendre transforms*, where we compute

$$q_m(\cos\theta_k) \equiv \sum_{|m|}^{L} a_{\ell m}\tilde{P}_\ell^m(\cos\theta_k) \tag{4.4}$$

for each ring k. Doing this for $O(L)$ rings can be done in $O(L^3)$ by just summing up brute-force, applying equation (4.2) as we go. Then, having computed $q_m(\cos\theta_k)$ for each combination of (m, k), we can use Fourier transforms along the m-direction to produce the final function values $f(\theta_{k(i)}, \phi_i)$. The *Fast*

Fourier Transform (FFT) allows computing the Fourier transform in $O(L \log L)$ operations, and we do $O(L)$ such transforms. Thus the total SHT runs in $O(L^3)$ instead of $O(L^4)$ when organizing grid points in rings. This trick has been around for as long as SHTs have been computed.

This complexity of $O(L^3)$ represents the status quo in production quality SHT libraries. For a 2D Fourier transform we would simply apply the FFT along both axes of the grid to get lightning fast transforms; while for SHTs, equation (4.4) is computed by brute force. Can we do better? The answer is yes, but the field is not yet mature; there is not a single obvious algorithm with several production-quality libraries. Instead of have a set of creative algorithms, and whether to apply them depends on the circumstances.

Tygert (2010) presented a novel algorithm based on *matrix compression*. The idea behind matrix compression in general is to run a compression algorithm on the matrix in such a way that the result can be used in linear algebra *without decompression*, and actually speeds up the algorithm. When they work, they typically turn an $O(n^2)$ algorithm into a $O(n \log n)$ or even $O(n)$ algorithm. Such techniques bridges the gap between (much more widely used) sparse linear algebra and dense linear algebra; they may work in cases where no matrix elements are zero (unlike sparse matrices) but where (unlike dense matrices) all $O(n^2)$ elements are generated from $O(n)$ or fewer parameters. A family of such algorithms have been developed relatively recently, starting with $\mathcal{H}$-matrices and $\mathcal{H}^2$-matrices in 1999 (for a review see Börm et al., 2003). Hiererarchical Semi-Separable matrices is another well known example (Chandrasekaran et al., 2006).

When applying matrix compression to SHTs, the idea is to first precompute all values of $\tilde{P}_\ell^m(\cos\theta_i)$ once, compress them, and then only use the compressed pre-computed data to compute the SHTs. The algorithm is recursive in nature: After compressing the matrix once using the *Interpolative Decomposition*, the resulting blocks are rearranged, using permutations that directly mimics the so-called *butterfly permutations* used in the Fast Fourier Transforms. After such permutation the result can be compressed further. As the computational cost now depends on how well the compression algorithm works, Tygert (2010) provide no hard proof of the scaling, but conjecture that the scaling of the algorithm is $O(L^2 \log L)$.

Such scaling for SHTs looked promising indeed, putting it in same ballpark as the FFT. However, modern computers can do arithmetic operations on data that is already in the CPU *very* quickly, while moving data from system memory is much slower. Unlike hardware from the 80's, it depends on the circumstances whether it is better to load 8 bytes from memory, or better to carry out 50 arithmetic operations. The best way to figure out if using precomputed data worked well for SHTs was to try it out.

Wavemoth represents a prototype optimized implemention of the algorithm of Tygert (2010) in C, employing all the tricks in the book to make sure the algorithm gets a fair chance:

- Carefully consider how memory is accessed to make sure the CPU is not waiting too much for data to come from system memory, or to travel

between the cache hierarchies in the CPU. The fundamentals of this craft is proper *blocking* of the computation, which is described well in Goto and Geijn (2008); a must-read for anyone interested in writing efficient code on modern computers. The other aspect is to organize the precomputed data so that data is stored in the order it is needed, without much jumping around.

- Use of generated C code to be able to fine-tune the block sizes of the computations, without the overhead of dynamic loops. This is important because modern CPUs pipeline arithmetic operations: On each CPU cycle a new arithmetic operation can be kicked off, but each operation spends several cycles to complete; much like a factory assembly line. Thus doing several independent computations interleaved in the same loop is much faster, as one can fill up the "assembly line" rather than leave it with holes while waiting for results. While compilers try to optimize this, manual control of this aspect is still faster.

- Use of vector instruction sets which enables further parallelism, specific to Intel and AMD CPUs. Wavemoth is using SSE2, which was up to date at the time, although out of date as of this writing.

- The compression of the matrices was tuned to the memory bus speed vs. CPU speed of the benchmark hardware. On the Intel system best results were achieved by assuming that a memory load was as expensive as $\rho = 7.5$ arithmetic instructions, while on the AMD system $\rho = 18$ was a better value.

The most important result of the effort may have been to show that the scaling conjecture of Tygert (2010) was wrong. The algorithm seems to scale as $O(L^2(\log L)^2)$.

For resolutions of interest to CMB analysis today, Wavemoth still achieved a 3x speedup compared to the most performant SHT library at the time, `libpsht` by Reinecke (2011). The cost is the inconvenience of keeping 6 GB of precomputed data in memory. However, out of this 3x factor, a 2x speedup over `libpsht` was achieved simply by re-organizing how the brute-force computations were done, in order to better utilize the pipelining features of the CPU. Thus the matrix compression algorithm only gets the credit of a 1.5x speedup, while the 2x came from doing better in terms of raw CPU optimization. Later, `libsharp` was released which closes this gap.

The use of Wavemoth must however be carefully evaluated as the memory consumption also scales with the resolution. For CMB analysis it seems that the extra complexity and memory use is not worth the modest speed increase. On higher resolutions than the ones in use for CMB analysis, Wavemoth (in a more polished form) may well be the best choice. It could also be useful in settings where only an approximate SHT is needed, since one can compress the data harder in return for lower numerical accuracy in the resulting SHT. It is probably well-suited for combination with the pseudo-inverse preconditioner in Paper II, since preconditioners often only need to be accurate to the 10% or

1% level, although we have not attempted this combination. Note that due to the experimental nature, only synthesis is available in Wavemoth, although implementing analysis too should be straightforward.

4.2 libsharp: The standard SHT library

Paper IV

The CMB community has for many years used the HEALPix package for all SHT needs (Górski et al., 2005). It is in fact two libraries distributed as one, a Fortran library and a C++ library, each originally with its own SHT implementation, even using different conventions for storing $a_{\ell m}$ in memory. The HEALPix/Fortran SHTs also contained an MPI-distributed version written by Hans Kristian Eriksen. The distribution scheme was suboptimal, as it required all $a_{\ell m}$ to all be present on every node, as well as extra/redundant computation. The MPI problem was fixed by Stompor et al. (2006) in their S^{2}HAT code, which partitions the data evenly among nodes both in pixel domain and spherical harmonic domain, and require no redundant computation.

What happened next is that Reinecke (2011) presented `libpsht`, and it became arguably the best general purpose SHT library for non-distributed computation. Its focus was primarily on being extremely flexible with respect to how input and output data was presented in memory. Arrays are stored in different ways in different circumstances – if you transform many maps at the same time, are the coefficients interleaved in memory or stored one after another? How is the irregular $a_{\ell m}$ data stored? Libpsht does not insist on anything, but instead takes a description of how the data is stored in whatever array you already have in memory, and works with that, saving an extra copy of the data on input and output in most circumstances. This directly led it to being suitable as a common SHT backend for HEALPix/C++ and HEALPix/Fortran.

Martin Reinecke had already started on a rewrite of libpsht, named *libsharp*, when I dropped Wavemoth and joined forces with him. Quoting our Paper IV:

> Also, several new, highly efficient SHT implementations have been published in the meantime; most notably Wavemoth (Seljebotn 2012) and shtns (Schaeffer 2013). These codes demonstrate that libpsht's computational core did not make the best possible use of the available CPU resources.

Libsharp keeps the flexible programming interface of libpsht, while using the computationally more efficient approach of the brute-force implementation within Wavemoth. The exploratory faster algorithm from Wavemoth is not included. Libsharp was primarily implemented by Martin Reinecke prior to us making contact. My own contributions to the library has been in discussions, co-authoring the paper, some bug fixes, the new data ordering format discussed in the next section, as well as Python and Fortran wrapper interfaces.

Martin also added distributed MPI support to Libsharp using the optimal algorithm from S^{2}HAT (Stompor et al., 2006). Libsharp is the backend of newest versions of HEALPix/C++, HEALPix/Fortran, and Healpy, so that most CMB analysts end up using it, whether directly or indirectly.

4.3 Data ordering in SHTs

Paper IV

One of the features I contributed to libsharp was a new SHT convention. Commander has always been converting data to/from the HEALPix format, and this extra step is now not needed when using libsharp directly.

The SHT libraries we have discussed including libsharp only supports real fields on the sphere, as in $f(\hat{n}) \in \mathbb{R}$. Polarization is no exception; one then works with a T, Q- and U-component in each grid point, but these are still considered to be in $\mathbb{R}$. The spherical harmonic transforms are however complex in nature, working with $f(\hat{n}) \in \mathbb{C}$ – it is just that the fields that arise in practice during CMB analysis is constrained to $\mathbb{R}$, and the SHT libraries use this fact to optimize their implementation. Because spherical harmonics are complex, the convention has been to work with *complex* coefficients $a_{\ell m} \in \mathbb{C}$. Knowing that the coefficients corresponds to a real field $f(\hat{n}) \in \mathbb{R}$, we have the relation $a_{\ell,m} = (-1)^m a_{\ell,-m}$, and so we do not need to store $a_{\ell,m}$ for $m < 0$ in memory. Therefore the HEALPix convention is to work with arrays where $a_{\ell,m}$ is dropped for $m < 0$. Mathematically speaking the dropped $a_{\ell,m}$ are however very much around, it is just that we can compute them on the fly when needed.

As discussed in chapter 2, our use of the SHT is in linear systems with spherical harmonic synthesis appearing as the $\mathbf{Y}$ operator. If we set up

$$\mathbf{A}\mathbf{x} = \mathbf{b}.$$

where $\mathbf{x}$ and $\mathbf{b}$ are maps in spherical harmonic domain, using only $x_{\ell,m}$ and $b_{\ell,m}$ for $m \geq 0$, then the system is not constrained to producing solutions in $\mathbb{R}^n$ even if the right-hand side is real. What is missing is to add the constraint $a_{\ell,m} = (-1)^m a_{\ell,-m}$ to the system. This *can* be added by extending $\mathbf{A}$ and $\mathbf{b}$, but this is very wasteful in terms of memory use. A much simpler solution is to "repackage" the spherical harmonic coefficients so that $x_{\ell,m} \in \mathbb{R}, b_{\ell,m} \in \mathbb{R}$, and we get the correct degrees of freedom. A particularly convenient choice is to let

$$a^R_{\ell,0} = a^C_{\ell,0},$$

where the superscript denotes real vs. complex, and then for each $m \geq 0$ let

$$\begin{aligned} a^R_{\ell,m} &= \sqrt{2}\mathrm{Re}(a^C_{\ell,m}) \\ a^R_{\ell,-m} &= \sqrt{2}\mathrm{Im}(a^C_{\ell,m}). \end{aligned}$$

The rescaling ensures that $\mathbf{Y}$ is still orthonormal. In this convention a random Gaussian field can be constructed simply by drawing each coefficient independently from a standard normal distribution; this is more complicated for the complex spherical harmonic coefficients.

What is the computationally most efficient way to store these coefficients? For SHTs the answer is to store them in m-major ordering, so that $m = 0$ comes first, then $|m| = 1$, and so on. This is simply because SHTs process data in this order, see equation (4.4). In addition we interleave negative and positive m, i.e.,

$$a_{0,0},\ a_{1,0},\ a_{2,0},\ a_{1,1},\ a_{1,-1},\ a_{2,1},\ a_{2,-1},\ a_{2,2},\ a_{2,-2} \tag{4.5}$$

This is because a) there is a relation between $\tilde{P}_\ell^m$ and $\tilde{P}_\ell^{-m}$ which makes it efficient to treat them at the same time, and b) one can now convert the data directly in-place to complex SHTs simply by scaling all coefficients where $m \neq 0$, so that $\mathrm{Im}(a_{1,1})$ uses the same location as $a_{1,-1}$ in memory[1].

Having chosen an order, is it best to store them in a 2D array, so that maps can be accessed like `x(l, m)` and matrices like `A(l1, m1, l2, m2)` (in the case of Fortran)? The number of coefficients vary, with $2m+1$ coefficients for each m, so this requires padding with zeros in unused portions of the arrays. My recommendation is to use the "packed" format indicated in equation (4.5) directly, because all arrays can then be passed directly to standard linear algebra routines which understands nothing about spherical harmonics. The padding very much gets in the way if one wants to use LAPACK routines. The difference is slight between `x(l, m)` and `x(lm_to_idx(lmax, l, m))` anyway.

Libsharp directly supports packed, real spherical harmonics in the order of equation (4.5), without any extra data shuffling.

4.4 SymPix: A grid for efficient sampling of rotationally invariant linear operator

Paper V

In Paper I, we describe an algorithm for efficiently solving a particular linear system. One of the disadvantages of the algorithm is that it requires a lot of precomputed data; specifically it requires the evaluation of a computationally expensive function $A(\hat{n}_1, \hat{n}_2)$, where $\hat{n}_1$ and $\hat{n}_2$ are points on the HEALPix spherical grid, for every $\hat{n}_2$ in a disc in the neighborhood of $\hat{n}_1$. Computing this for the HEALPix grid takes a long time (12 CPU hours at $L = 3000$). Now, there are certain symmetries in the HEALPix grid we could have exploited to cut this with a factor of 24, however this would have require quite a lot of coding specific to the HEALPix grid.

Instead, we decided to invent a new grid dubbed *SymPix*, which we present in Paper V. The SHT algorithms leave us with a lot of freedom in what kind of spherical grid we want to use, and each spherical grid comes with a set of advantages and disadvantages. The unique advantage of SymPix is that it allows a great number of symmetries for the computation mentioned above, which cuts the precomputation time down to 5.4 minutes; a speedup of 130 times. The disadvantage of SymPix relative to HEALPix is that, if divided into pixels, each pixel will have different area. In the context where we use SymPix we do not use pixels as such, but rather consider the data as sample points of in a given grid pattern of an infinite-resolution continuous function, and so this restriction is not a real problem. However, big differences in sample points distances can lead to different scales being present in different regions on the sphere, which is a problem for the algorithm of Paper I. In this respect SymPix does slightly worse than HEALPix, but a lot better than other grids

[1] A disadvantage of this ordering is that the formula for accessing a specific coefficient becomes a bit complicated. With a certain ℓ-major ordering the index is as simple as $\ell^2+\ell+m$. This is however something that can be abstracted away in code.

such as the more popular Gauss-Legendre grid.

4.5 Legendre transforms on the GPU

Technical Report I

Technical Report I describes novel techniques used to implement the Legendre transforms on the GPU. To my knowledge it is the world's fastest code for Legendre transforms, at 40% of the peak FLOP rate of the GPU. The Legendre transforms is a very interesting thing to try to make work well on the GPU; as demonstrated by the final implementation performing 45 times better than the initial naive implementation.

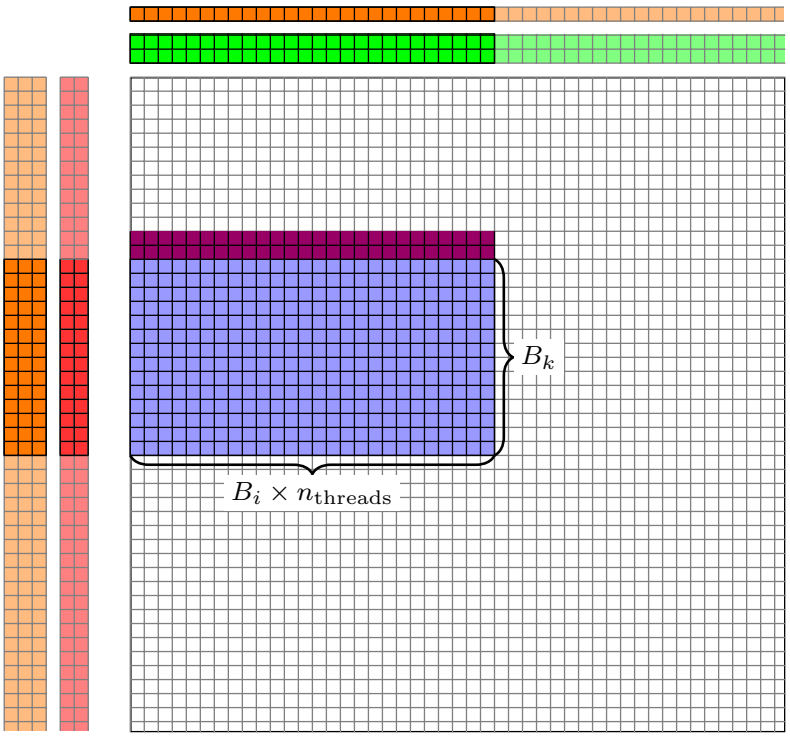
This code was written in order to pass a course on multi-architecture programming at the university. The result is published for the first time in this thesis.

Implementing the Legendre transforms on the GPU is a challenge because of the rather different programming model. The usual simple explanation of how to program for GPUs is that they perform well for problems that are embarrisingly parallel. In that case, the SHT would not be a good fit for the GPU, since every number in the input affect every number in the output, without much obvious parallelism to exploit on the high level. However, with detailed knowledge of how the GPU operates it is indeed possible to make different threads of computation cooperate on a fine-grained level.

The main features of GPU programming is (numbers are for Tesla M2050, which was a recent GPU when Technical Report I was written):

- Each GPUs contains 28 processors, so-called *Streaming Multi-processors* (SM)
- Each SM runs a large number of threads. The threads are grouped into so-called "warps" of 32 threads which run at the same time. The threads within a warp must execute the same instructions (in the case of taking different branches in an if-test, essentially both branches will be executed by all threads, but some threads will do nothing in each branch).
- Each SM runs several warps at the same time, so that when one warp is waiting for an operation to finish, another warp can run in the meantime. This is how GPUs deal with pipelining of instructions. How many warps to run on the same SM at the same time is configurable, but there is a single number of registers (thread-local variables) that is split evenly between all threads that run on an SM at the same time. Thus the more warps you schedule to run at the same time, the more efficient the pipelining will be, but the less local variables you have room for.
- Each SM has 48 KB local cache which can be accessed by all threads running on that SM at the same time, and used for inter-thread communication

Creating an efficient GPU program thus is mainly a matter of moving variables and data around in an efficient manner. It was crucial to use blocking



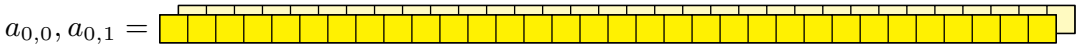
$$a_{k,j} = \sum_i \Lambda_{i,k} q_{i,j}, \quad \text{with} \quad \Lambda_{i,k+1} = (x_i^2 + \alpha_k)\beta_k \Lambda_{i,k} + \gamma_k \Lambda_{i,k-1}$$

Figure 4.2: Blocking of the Legendre transform. The goal is to compute the output $a_{k,j}$ (red) using some auxiliary values (orange) associated with both columns and row. The Legendre matrix values Λ (blue) must be generated on the fly as they are needed, from the two matrix elements above in the same column. When we are done computing this block, we persist the bottom two rows of Λ to memory and move to the block on the right. We can then keep the 5 scalars associated with each row in memory, but need to load the 3 scalars associated with the next set of columns (two inputs and one auxiliary coefficient). For the Tesla M2050 the optimal parameters were $B_k = 64$, $B_i = 4$, and $n_{\text{threads}} = 64$.

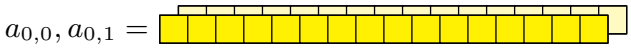
techniques, as explained in figure 4.2, so that loads/writes to global memory are properly amortized. Furthermore the Legendre transforms requires summing up variables across threads, and I implemented a non-trivial scheme for this (see figure 4.3). Finally, both since there are many tunable parameters, and since there was a complicated algorithm which could be unrolled into linear code instead of using loops and if-tests, the implementation is generated by meta-programming, so that the final CUDA/C code is generated by a Python program.

The code was never cleaned up and properly published independent of the experimental Wavemoth testbed, but the *very* interested reader may check out the 'cuda' branch of the `wavemoth/wavemoth` GitHub repository and have a look at the benchmark script in `examples/gpusht.py`[2].

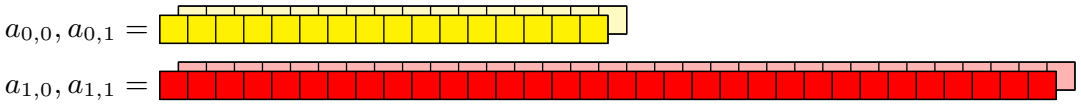
[2] https://github.com/wavemoth/wavemoth/blob/cuda/examples/gpusht.py



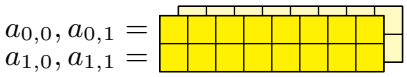
(a) Step 1: We want to compute the red output cells $a_{k,j}$ of figure 4.2. For each of the green input $q_{i,j}$ we multiply with $\Lambda_{i,k}$ to produce a contribution to the output sum. Each thread can trivially sum up the B_i columns it is responsible for; but summing across threads is much harder. In this figure there are two arrays with contributions to $a_{0,0}$ and $a_{0,1}$; each of 32 threads holds one scalar from each array.



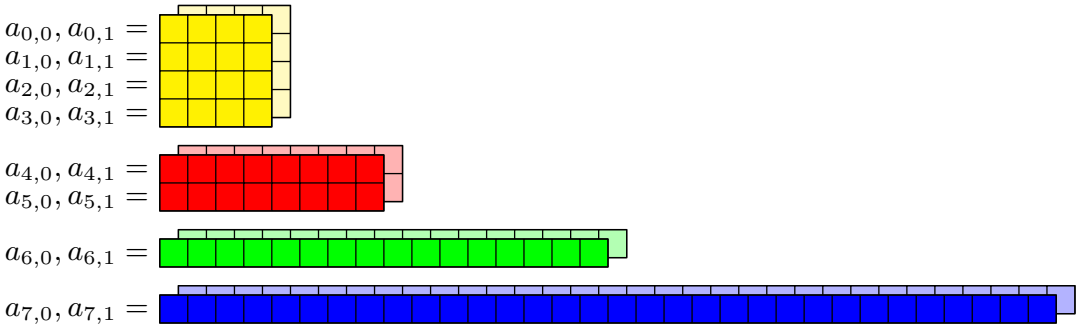
(b) Step 2: The threads group in pairs and sends each other values. After this, the 32 threads hold 1 scalar each; even-numbered threads holds the first vector and odd-numbered threads the other vector.



(c) Step 3: Continuing to add together values at this point would leave some threads stalling. To maintain high parallelism, we start to generate the next row of the output before we are done with summing up the first one.



(d) Step 4: We first repeat step 1 and 2 for a_1; leaving us with 4 arrays of length 16. Then we can sum them to 4 arrays of length 8 with 32 threads cooperating in pairs, with no thread idling.



(e) Step 13: Now the algorithm has progressed quite a bit. Once we had 8 arrays of length 8 we could sum them to 8 arrays of length 4, producing the yellow block, and so on. Continuing in this manner a few more times recursively produces the situation above. In the final step we have 32 output arrays, each of length 2, which we sum to the final 32 outputs.

Figure 4.3: Illustration of the algorithm used for efficient Legendre transforms on the GPU, describing how we deal with summing across multiple threads when computing $a_{k,j}$ from $q_{i,j}$. The color indicate register slot used in each thread; in panel (e) each thread is storing output data in 4 registers, one for each of the colors. Transforms in the other direction are trivial, and does not need this reduction algorithm.

Chapter 5

Application to Planck

Since the first paper from the Oslo/JPL "Commander" group on the CMB Gibbs sampler (Eriksen et al., 2004), the goal of the Oslo/JPL Commander effort has been the analysis of data from Planck. For a successful analysis a lot of questions must be answered. What priors $\mathbf{S}$ should be used? Which set of input maps? What foreground components should be treated as seperate components and which should be bundled together as an average component? Does the maps from the TOD processing look OK, or are there systematic effects which must be dealt with prior to an analysis with Commander? How to quantify and subtract the cosmic dipole? How to calibrate the instrumental gain? These questions are at least as important as the computational techniques, but have been out of scope for my work. Still I provide a high-level review here so as to put the algorithms presented in this thesis in context.

5.1 Planck 2013 results

Review

Recall from section 1.5 that in order to produce estimates of the cosmological parameters one runs an MCMC sampler (typically CosmoMC). This method requires as its input some *CMB power spectrum likelihood*, which quantifies how well the data fits a given proposed power spectrum C_ℓ, and as its output produces estimates such as the ones presented in figure 1.11.

The Gibbs sampling algorithm outlined in section 2.1 currently produce samples from the posterior of the observed power spectrum, $\sigma_\ell^{(i)} \sim p(\sigma_\ell|\mathbf{d})$. To turn this into a likelihood that CosmoMC can work with, one can use the *Blackwell-Rao estimator* (Chu et al., 2005). The idea is that the likelihood CosmoMC requires, $p(C_\ell|\mathbf{d})$, can be written as an integral over σ_ℓ, which can in turn be approximated using the samples from the Gibbs chain:

$$p(C_\ell|\mathbf{d}) = \int P(C_\ell|\sigma_\ell)P(\sigma_\ell|\mathbf{d})d\sigma_\ell \approx \frac{1}{N}\sum_{i=1}^{N} P(C_\ell|\sigma_\ell = \sigma_\ell^{(i)}). \tag{5.1}$$

Chu et al. (2005) further provides a simple analytical expression for evaluating

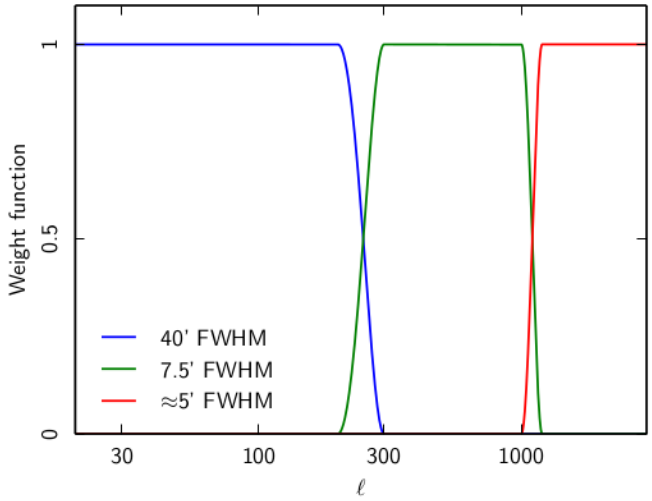


Figure 5.1: Combination of multiple Commander single-resolution estimates to produce a single high-resolution CMB estimate. First, three Commander estimates of the CMB are found, then they are summed together in spherical harmonic domain, weighed by the weight displayed in this figure. The low-resolution estimate at 40' FWHM includes all the Planck data and all components. The 7.5' FWHM estimate excludes frequencies lower than 143 GHz, and is therefore able to pick a finer common resolution. A consequence is also that low-frequency components is not included, only CMB, CO and thermal dust. Finally, the ∼5' FWHM-solution only includes CMB and thermal dust. See Planck Collaboration IX (2015) for further details.

$P(C_\ell|\sigma_\ell = \sigma_\ell^{(i)})$. Thus, the CMB samples from a standard Commander run can be turned into a likelihood for cosmological parameter estimation. The likelihood incorporates all the uncertainties that is present in the Commander model (e.g., instrumental noise in each pixel) and faithfully propagates them to the final parameter estimates. However, there is a problem: The higher ℓ becomes, the slower is the convergence of the approximation in equation (5.1) and a higher number of samples N is required. Therefore, for the Planck 2013 release on the CMB power spectrum they split the likelihood in two: A low-ℓ likelihood for $\ell < 50$, and a high-ℓ likelihood for $\ell \geq 50$. The low-ℓ likelihood is exact (up to sample convergence) and is provided by Commander, while for the high-ℓ likeihood they present two different approximations (Planck Collaboration XV, 2014). They also provide comparisons of some different ways in which the likelihoods can be joined together (Paper VIII details another such approach).

In addition to the highly important low-ℓ likelihood, Commander contributed component seperation results to the release, alongside similar results from `NILC`, `SEVEM`, and `SMICA` (Planck Collaboration XII, 2014). As only Commander 1 was available at the time, all the data had to be degraded by artificially low-pass filtering it with a Gaussian beam, to a common a HEALPix grid of resolution $N_{\text{side}} = 256$. To produce full-resolution maps, these low-resolution estimates were combined with the `Ruler` code to produce what is in the paper referred to as the `Commander-Ruler` map. First, the low-resolution run of Commander produces many samples of low-resolution mixing maps, $\mathbf{Q}_{\nu,k}$. Then, the Ruler code generates amplitude maps for each sample of the mixing map

samples. The starting point for the Ruler approach is the same as what we describe in section 2.3: A least squares system is set up that includes a mixing map $\mathbf{Q}_{\nu,k}$ for each frequency band and component. Where the Ruler approach differs is that it, as an approximation, neglects the instrumental beams and different resolutions of the maps entirely. The system then becomes block diagonal in pixel domain, with an N_{comp}-by-N_{comp} block for each pixel which is computationally trivial to invert. Together with these approximate amplitude maps they generate an "effective beam" representing the resolution of resulting maps. Neglecting the resolution differences in this way has its limits, and as a result it was not possible to include the 545 GHz and 857 GHz maps, only 353 GHz and below.

The Commander-Ruler model included a single low-frequency component (modelling the combined effect of synchrotron emission, free-free emissions and spinning dust emissions), a simple thermal dust model, and a single CO component, in addition to the CMB. The 2013 Planck release was temperature-only and no polarization results were present in the release.

5.2 Planck 2015 results

Review

The 2015 release contains the most sophisticated Commander model yet. The Planck data was complemented by the WMAP data as well as the 408 MHz "Haslam" map. Also, some of the Planck data was included with individual maps for each detector, instead of one average map for the frequency — this was done because each detector has *slightly* different frequency bandpass functions, which especially helps to constrain the CO components (with emission lines right in the middle of some of the Planck frequency bands). This wealth of data enables seperating the microwave emissions into a large number of individual components: Synchrotron, free-free, spinning dust, thermal dust, and three different CO components. For thermal dust, a physically modifified blackbody spectrum is used, with two parameters instead of one. The result of this Commander analysis becomes the official Planck foreground maps (Planck Collaboration X, 2015).

This joint analysis was performed on a common resolution of 1 degree, as this is the resolution of the Haslam map. In addition, 7.5 arc minute resolution results are provided by only including the high-frequency components and only frequencies from 143 GHz and above. For the final Commander CMB map, a hybrid map is produced by combining results from three different runs at different resolutions, as shown in figure 5.1. Note the similarities between this figure and and figure 4 in Paper II; the manual hybrid map approach has some similarities to pseudo-inverse preconditioning. As usual, this estimate was presented together with similar results from NILC, SEVEM and SMICA (Planck Collaboration IX, 2015).

Finally, Commander again provided results for the low-ℓ likelihood. In the 2015 release it was able to provide a Blackwell-Rao estimator all the way up to $\ell = 250$, although in the final analysis it was decided to use exact methods only up to $\ell < 30$, and the approximate likelihoods above.

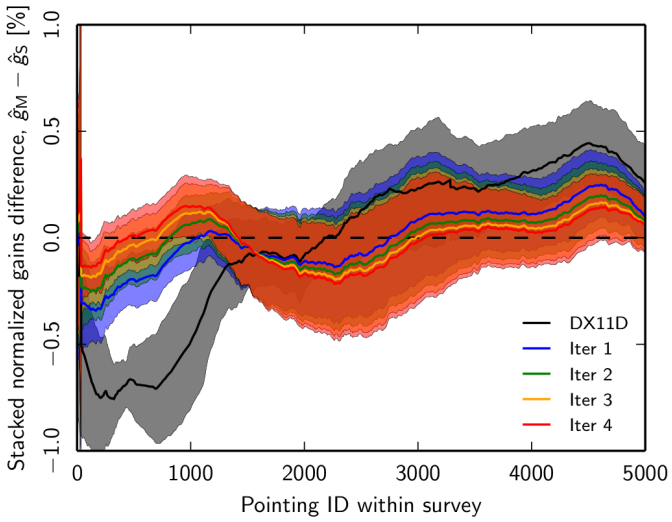


Figure 5.2: Use of Commander in LFI gain calibration. The solid lines (with 1σ confidence bands) represents an estimate of the gain error. For each iteration, new full-sky maps were produced from TOD data, component seperation done with Commander, and results sent back as input to gain calibration. Figure from Paper VI.

5.3 Planck 2017 results

Paper VII, Paper VI

The 2017 release is still being prepared as of this writing, and the relevant papers for this thesis are included in draft form. This will be the first release that uses Commander 2, with full-fledged multi-resolution component seperation as described in section 2.4 and Paper II. Thus a single run of Commander can include the lowest frequencies while still producing full-resolution thermal dust maps. In the 2017 release the Commander results for the CMB sky will again be presented alongside similar estimates from NILC, SEVEM and SMICA. New in this release is a new GNILC method which extends NILC to also provide foreground maps for thermal dust, free-free, and CO emission as well as the Cosmic Infrared Background. Paper VII describes the results in further detail.

Additionally, in this release Commander had a role in the core time-ordered data (TOD) processing of LFI. As part of the instrument calibration one needs to find a non-linear *gain function*, which translates electrical readings to emission intensity in the sky. This gain function is calibrated against a large dipole present in the microwave sky, which is induced by a Doppler effect caused by the movement of the Planck satellite. This dipole is removed prior to further processing, but is crucial for estimating the gain function. In the 2015 release, the uncertainty in the gain estimate was on the order of 0.1%. While an error on this level is acceptable for a temperature analysis, it represents a noticeable source of systematic error for an analysis of the polarization data. To make further improvements, it was necessary to take into account the true signal that was being scanned on the sky when estimating the dipole. One then has a circular dependency: To make such an estimate of the sky, one needs maps; to get maps of the sky, one needs a gain function. Circular dependencies are however not a problem (as discussed in section 2.1); one can simply iterate

back and forth until convergence. In this case, we did not invest the effort (or computational resources) of running a full Gibbs sampler; rather we simply iterated manually a few times, sending results back and forth between the Commander group and the LFI TOD group. See Paper VI for further details. This dependency of the gain calibration on the sky signal emphasises the need for Commander to directly work with TOD data in the future (see chapter 6).

Chapter 6

Summary and outlook

Bayesian analysis of CMB data

This thesis revolves around computational techniques for optimizing the *Commander* approach to CMB analysis: Design a Bayesian model motivated by the underlying physics, and then use Gibbs sampling and other MCMC methods to explore the posterior distribution. Most of the computation time during such an analysis is spent solving the *constrained realization* linear problem (CR system), and all of the research in this thesis is in some way related to solving this linear system.

There were many attempts at developing a good solver for the CR system that could handle the full resolution of the Planck data. Some failed, while other were partial successes like the solvers in section 3.4 and Paper I. In the end the solver presented in Paper II is a clear winner, being more efficient, more robust to changes in the model, and easier to implement than any of the alternatives we tested, and converging in less than 40 iterations for all the models tested. The underlying principles are also transparent and robust, and should be able to incorporate many future extensions.

What lies in the future for Commander and exact Bayesian analysis of the CMB? So far a number of simplifying assumptions have been made about the instrumental properties. The strongest such assumption is that it is possible to have a waterfall process, where one first turns the Time Ordered Data (TOD) into maps, and then carry out the Bayesian analysis on these maps. Given the experience from the latest Planck release, where one had to iterate between map-making and component seperation in order to successfully calibrate the gain, it is clear that this workflow is not feasible for much longer.

Thus it seems clear that "Commander 3" will need to work with Time Ordered Data (TOD) directly, instead of working with sky maps. This also makes it easier to work around several of the approximations made today in the model, such as assuming that the instrumental noise being uncorrelated between neighbouring pixels, and assuming a symmetric beam. Algebraically, the constrained realization/component seperation system would still look the same as equation (2.5), but each entry in the data vector of a band, $\mathbf{d}_\nu$, now repre-

sents a detector sample at a given frequency instead of a pixel on a spherical map.

Two points should be made about solving the CR system with TOD data. First, all the methods in this thesis may still find a use. Because working with TOD data is so expensive, one may prefer to solve the system using map input, symmetric beams, uncorrelated noise etc. as a preconditioner for the full TOD system, ideally just making a very few passes through TOD data. Second, as we point out in Paper II, it is possible that the pseudo-inverse preconditioning strategy is well suited also when working with TOD data directly.

Another area where the Commander framework should be developed in the future is in the signal component priors. We have so far assumed an isotropic prior for all components, which can be specified in the form of a power spectrum C_ℓ, with a sharp band-limit L. This model has a tendency to excite ringing in the resulting maps around sharp objects, unless much time is spent tuning the priors, or one adopts a very high band-limit L for all components. Working with pixel-domain vectors and, with the exception of the CMB, pixel-domain prior specifications, one could easily define the models that are more robust against this problem. Also, we know that the diffuse foregrounds has much variation where their signal is strong, but should be more heavily stabilized where their signal is weak. Such a non-isotropic prior is easier to model using a sparse matrix in pixel domain. Of particular interest are the so-called Conditional Auto-Regressive (CAR) models, as discussed in section 2.5.

Spherical harmonic transforms

The computational work-horse when solving the CR system is the SHT. Paper III develops the experimental SHT code *Wavemoth*, efficiently implementing an algorithm by Tygert (2010) based on *matrix compression*. At the time it was the fastest SHT code available for very high resolutions, although at the cost of impractically high memory use. It was also in some sense a negative result, as it demonstrated that the algorithm had a computational scaling of $O(L^2(\log L)^2)$, instead of $O(L^2 \log L)$ as claimed by Tygert (2010).

Research in fast SHT algorithms is still ongoing, as can also be witnessed by having a look at some of the papers citing Paper III, a recent example being Slevinsky (2017)[1], who proposes a fast transform turning the SHT problem into a 2D FFT problem. Another algorithm of particular interest to me is the one by Tygert (2008), which predates the one in Tygert (2010) and Paper III, but may well turn out to be better in the end. It has a proven scaling of $O(L^2 \log L)$, and while its pre-factor is rather high at full precision, the pre-factor contains a factor of $O(d^2)$, where d is the required precision (number of correct digits). Thus by reducing the requirements to the precision one can drastically reduce the pre-factor involved. For the requirements of our solver in Paper II, a precision of 10^{-5} should be sufficient for the linear solver, and the preconditioner has been tested to perform well with a precision of only 10^{-1}. At these levels the algorithm in Tygert (2008) should be very competitive, perhaps speeding up the SHTs by an order of magnitude. Note that the algorithm in Paper

[1] Pre-print, not peer-reviewed as of this writing

III also has tunable accuracy; unfortunately the "low" accuracy experiment is done at 10^{-8} in that paper. The largest disadvantage of the algorithm in Paper III is the memory requirements. These are much more modest in the algorithm of Tygert (2008), and I therefore feel the latter to have more potential in the future.

As discussed earlier, the lasting legacy of the Wavemoth code is the low-level optimization done on the brute-force core, in particular in exploiting the pipelining capabilities of the CPU. This approach was integrated by Martin Reinecke into a revised version of the *libpsht* code, resulting in the release of the subsequent *Libsharp* (see Paper IV), which is in use by thousands of researchers every day.

Technical Report I explores the implementation of the SHT on GPUs, by providing an implementation of the most challenging part: The Legendre transforms in the "inconvenient" direction. The implementation performs at 40% of the peak floating-point capacity of the GPU; this is quite good for an algorithm that requires interaction between threads of computation. To my knowledge there has not been any research on SHTs on the GPU since that report was written, and so the work should still be relevant. The first missing piece to turn the code into a production quality code is a Legendre transform in the opposite direction, but this is a trivial piece of code in comparison. The other missing piece is a set of 1D FFTs; these one could perform on the CPU using FFTW3[2] in a first iteration. Also the PyFFT[3] project provides an open source implementation of the FFT for the GPU which could perhaps be integrated.

[2] http://www.fftw.org/

[3] Despite the name, this is primarily a GPU code, using Python only to generate the code. https://pythonhosted.org/pyfft/

Appendix A

Bibliography

Michele Benzi, Gene H. Golub, and Jörg Liesen. Numerical solution of saddle point problems. *ACTA NUMERICA*, 14:1–137, 2005.

BICEP2/Keck Collaboration and Planck Collaboration. Joint Analysis of BICEP2/Keck Array and Planck Data. *Physical Review Letters*, 114(10):101301, 2015.

Steffen Börm, Lars Grasedyck, and Wolfgang Hackbusch. Introduction to hierarchical matrices with applications. *Engineering Analysis with Boundary Elements*, 27 (5):405 – 422, 2003. ISSN 0955-7997.

S. Chandrasekaran, M. Gu, and T. Pals. A fast *ulv* decomposition solver for hierarchically semiseparable representations. *SIAM Journal on Matrix Analysis and Applications*, 28(3):603–622, 2006.

M. Chu, H. K. Eriksen, L. Knox, K. M. Górski, J. B. Jewell, D. L. Larson, I. J. O'Dwyer, and B. D. Wandelt. Cosmological parameter constraints as derived from the Wilkinson Microwave Anisotropy Probe data via Gibbs sampling and the Blackwell-Rao estimator. *Physical Review D*, 71(10):103002, 2005.

Noel Cressie and Christopher Wikle. *Statistics for Spatio-Temporal Data*. Wiley, 2011.

Howard C. Elman. Preconditioning for the steady-state navier–stokes equations with low viscosity. *SIAM Journal on Scientific Computing*, 20(4):1299–1316, 1999.

F. Elsner and B. D. Wandelt. Efficient Wiener filtering without preconditioning. *Astronomy & Astrophysics*, 549:A111, January 2013.

H. K. Eriksen, I. J. O'Dwyer, J. B. Jewell, B. D. Wandelt, D. L. Larson, K. M. Górski, S. Levin, A. J. Banday, and P. B. Lilje. Power Spectrum Estimation from High-Resolution Maps by Gibbs Sampling. *Astrophysical Journal, Supplement*, 155: 227–241, 2004.

H. K. Eriksen, J. B. Jewell, C. Dickinson, A. J. Banday, K. M. Górski, and C. R. Lawrence. Joint Bayesian Component Separation and CMB Power Spectrum Estimation. *Astrophysical Journal*, 676:10-32, 2008.

K. M. Górski, E. Hivon, A. J. Banday, B. D. Wandelt, F. K. Hansen, M. Reinecke, and M. Bartelmann. HEALPix: A Framework for High-Resolution Discretization and Fast Analysis of Data Distributed on the Sphere. *Astrophysical Journal*, 622: 759–771, 2005. URL http://healpix.jpl.nasa.gov/.

Kazushige Goto and Robert A. van de Geijn. Anatomy of high-performance matrix multiplication. *ACM Trans. Math. Softw.*, 34(3), 2008.

E. Hivon, K. M. Górski, C. B. Netterfield, B. P. Crill, S. Prunet, and F. Hansen. MASTER of the Cosmic Microwave Background Anisotropy Power Spectrum: A Fast Method for Statistical Analysis of Large and Complex Cosmic Microwave Background Data Sets. *Astrophysical Journal*, 567:2–17, 2002.

J. Jewell, S. Levin, and C. H. Anderson. Application of Monte Carlo Algorithms to the Bayesian Analysis of the Cosmic Microwave Background. *Astrophysical Journal*, 609:1–14, 2004.

Antony Lewis and Sarah Bridle. Cosmological parameters from CMB and other data: a Monte- Carlo approach. *Phys. Rev.*, D66:103511, 2002.

Antony Lewis, Anthony Challinor, and Anthony Lasenby. Efficient computation of CMB anisotropies in closed FRW models. *Astrophys. J.*, 538:473–476, 2000.

S. Mitra, G. Rocha, K. M. Górski, K. M. Huffenberger, H. K. Eriksen, M. A. J. Ashdown, and C. R. Lawrence. Fast Pixel Space Convolution for Cosmic Microwave Background Surveys with Asymmetric Beams and Complex Scan Strategies: FEBeCoP. *Astrophysical Journal, Supplement*, 193:5, 2011.

S. Næss et al. The Atacama Cosmology Telescope: CMB polarization at $200 \leq \ell \leq 9000$. *Journal of Cosmology and Astroparticle Physics*, 10:007, 2014.

A. A. Penzias and R. W. Wilson. A Measurement of Excess Antenna Temperature at 4080 Mc/s. *Astrophysical Journal*, 142:419–421, 1965.

Planck Collaboration IX. Planck 2015 results. IX. Diffuse component separation: CMB maps. *Astronomy & Astrophysics*, 594:A9, 2015.

Planck Collaboration VI. Planck 2015 results. VI. LFI mapmaking. *Astronomy & Astrophysics*, 594:A6, 2015.

Planck Collaboration VII. Planck 2015 results. VII. High Frequency Instrument data processing: Time-ordered information and beams. *Astronomy & Astrophysics*, 594:A7, 2015.

Planck Collaboration VIII. Planck 2015 results. VIII. High Frequency Instrument data processing: Calibration and maps. *Astronomy & Astrophysics*, 594:A8, 2015.

Planck Collaboration X. Planck 2013 results. X. HFI energetic particle effects: characterization, removal, and simulation. *Astronomy & Astrophysics*, 571:A10, 2014.

Planck Collaboration X. Planck 2015 results. X. Diffuse component separation: Foreground maps. *Astronomy & Astrophysics*, 594:A10, 2015.

Planck Collaboration XII. Planck 2013 results. XII. Diffuse component separation. *Astronomy & Astrophysics*, 571:A12, 2014.

Planck Collaboration XIII. Planck 2015 results. XIII. Cosmological parameters. *Astronomy & Astrophysics*, 594:A13, 2015.

Planck Collaboration XV. Planck 2013 results. XV. CMB power spectra and likelihood. *Astronomy & Astrophysics*, 571:A15, 2014.

B. Racine, J. B. Jewell, H. K. Eriksen, and I. K. Wehus. Cosmological Parameters from CMB Maps without Likelihood Approximation. *Astrophysical Journal*, 820:31, 2016.

M. Reinecke. Libpsht - algorithms for efficient spherical harmonic transforms. *Astronomy & Astrophysics*, 526:A108, 2011.

Jonathan Richard Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. 1994. URL http://www.cs.cmu.edu/~jrs/jrspapers.html.

M. R. Slevinsky. Fast and backward stable transforms between spherical harmonic expansions and bivariate Fourier series. *ArXiv e-prints*, May 2017.

Radek Stompor, Mikolaj Szydlarski, and Laura Grigori. S^2HAT code, 2006. URL http://www.apc.univ-paris7.fr/APC_CS/Recherche/Adamis/MIDAS09/software/s2hat/s2hat.html.

J. M. Tang, R. Nabben, C. Vuik, and Y. A. Erlangga. Comparison of two-level preconditioners derived from deflation, domain decomposition and multigrid methods. *Journal of Scientific Computing*, 39(3):340–370, 2009. ISSN 1573-7691.

M. Tygert. Fast algorithms for spherical harmonic expansions, II. *Journal of Computational Physics*, 227:4260–4279, April 2008. doi: 10.1016/j.jcp.2007.12.019.

Mark Tygert. Fast algorithms for spherical harmonic expansions, III. *Journal of Computational Physics*, 229(18), 2010.

B. D. Wandelt, D. L. Larson, and A. Lakshminarayanan. Global, exact cosmic microwave background data analysis using Gibbs sampling. *Physical Review D*, 70 (8):083511, 2004.

Nils P. Wedi, Mats Hamrud, and George Mozdzynski. A fast spherical harmonics transform for global nwp and climate models. *Monthly Weather Review*, 141(10): 3450–3461, 2013.

Norman Yarvin and Vladimir Rokhlin. An improved fast multipole algorithm for potential fields on the line. *SIAM Journal on Numerical Analysis*, 36(2):629–666, 1999.

Part II

Papers

Paper I

A multi-level solver for Gaussian constrained CMB realizations

I tried more preconditioner variants for solving the CMB constrained realization problem at full resolution than I can remember. This paper presents the first published attempt and an important stepping stone towards our final answer to the problem in Paper II.

The contribution of this paper is in pointing out that pixel-domain multi-grid methods are needed to solve for the Laplacian-like system under the mask. However, the same methods are not able to carry out deconvolution of the instrumental beam. This was not needed for the test model in this paper, but, as we discovered when trying to generalize the solver to full-resolution component seperation, it is needed to recover the thermal dust signal. This problem is fixed in Paper II.

The Astrophysical Journal Supplement Series, 210:24 (13pp), 2014 February doi:10.1088/0067-0049/210/2/24

A MULTI-LEVEL SOLVER FOR GAUSSIAN CONSTRAINED COSMIC MICROWAVE BACKGROUND REALIZATIONS

D. S. Seljebotn[1], K.-A. Mardal[2,3], J. B. Jewell[4], H. K. Eriksen[1], and P. Bull[1]
[1] Institute of Theoretical Astrophysics, University of Oslo, P.O. Box 1029 Blindern, NO-0315 Oslo, Norway; d.s.seljebotn@astro.uio.no
[2] Department of Informatics, University of Oslo, P.O. Box 1080 Blindern, NO-0316 Oslo, Norway
[3] Centre for Biomedical Computing, Simula Research Laboratory, P.O. Box 134, NO-1325 Lysaker, Norway
[4] Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA 91109, USA
Received 2013 August 25; accepted 2013 November 29; published 2014 January 23

ABSTRACT

We present a multi-level solver for drawing constrained Gaussian realizations or finding the maximum likelihood estimate of the cosmic microwave background sky, given noisy sky maps with partial sky coverage. The method converges substantially faster than existing Conjugate Gradient (CG) methods for the same problem. For instance, for the 143 GHz Planck frequency channel, only three multi-level W-cycles result in an absolute error smaller than 1 μK in any pixel. Using 16 CPU cores, this translates to a computational expense of 6 minutes wall time per realization, plus 8 minutes wall time for a power-spectrum-dependent precomputation. Each additional W-cycle reduces the error by more than an order of magnitude, at an additional computational cost of 2 minutes. For comparison, we have never been able to achieve similar absolute convergence with conventional CG methods for this high signal-to-noise data set, even after thousands of CG iterations and employing expensive preconditioners. The solver is part of the Commander 2 code, which is available with an open source license at http://commander.bitbucket.org/.

Key words: cosmic background radiation – methods: numerical – methods: statistical

Online-only material: color figures

1. INTRODUCTION

Apart from a substantial kinematical dipole, the cosmic microwave background (CMB) radiation is observed to be isotropic to around one part in 10^4. Below this level, there are random fluctuations over a wide range of angular scales. The prevailing "concordance" cosmological model explains these anisotropies as the imprints of Gaussian-distributed, statistically isotropic perturbations of spacetime that were generated during an inflationary epoch in the early universe. Correlations between the fluctuations provide a wealth of information about inflation and the subsequent growth of structure, and so being able to accurately measure and characterize them is of paramount importance to modern cosmology.

As detector technology has improved, it has become possible to probe smaller and smaller angular scales with ever-increasing noise sensitivities. The resulting improvement in resolution and signal-to-noise ratio presents a formidable computational challenge, as one must now reliably reconstruct the CMB sky to high accuracy over tens of millions of pixels, while simultaneously taking into account complexities of the data such as inhomogeneous noise, foreground contamination, and regions of missing/masked data.

Consider an observed map of the CMB, for instance similar to those provided by the *Wilkinson Microwave Anisotropy Probe* (*WMAP*; Bennett et al. 2013) and Planck (Planck Collaboration 2013a) experiments. The ideal CMB map would consist of an error-free value at every single position on the sky. In reality this is of course not possible, because of instrumental imperfections (such as noise and beam-smoothing) and strong foreground contamination from astrophysical sources; there will always be uncertainties in a real CMB map. Therefore, rather than aiming to extract "a single true CMB sky map," a more realistic solution is to compute an ensemble of many possible CMB skies, each of which is both noise-free, full-sky, and *statistically consistent* with the observed data. This idea has already been implemented for CMB analysis purposes in terms of a Gibbs sampling framework, as described by Jewell et al. (2004), Wandelt et al. (2004), and Eriksen et al. (2004, 2008b).

An underlying assumption in this line of work is that both the CMB sky and instrumental noise are random Gaussian fields with covariance matrices **S** and **N**, respectively. In most applications—following the basic inflationary prediction—one additionally assumes that the CMB field is isotropic, so that the CMB covariance matrix can be specified in terms of a simple angular power spectrum, C_ℓ. Of course, this power spectrum is not known a priori, but must instead be estimated from the data, and indeed, this is usually the main goal for most CMB experiments.

The Gibbs sampling framework provides a well-structured mathematical solution to this power spectrum estimation problem, by establishing the full joint Bayesian posterior distribution of the CMB sky and CMB power spectrum. This is found by iteratively sampling from the (more tractable) conditional distributions according to a simple algorithm: (1) make an arbitrary initial "guess" for the CMB power spectrum, (2) draw a CMB sky map compatible with the data and the assumed power spectrum, (3) draw a power spectrum compatible with the sky sample that was just drawn, and (4) iterate. The resulting set of sky and power spectrum samples will (after some burn-in period) converge to the true joint posterior distribution.

Although simple to write down, this algorithm is also computationally rather expensive due to step (2), which essentially amounts to solving a large linear system with one or more random terms on the right-hand side, corresponding to different realizations. We will refer to this system as the *constrained realization* (CR) system. The same linear system can also be solved for the maximum likelihood CMB sky map estimate, which is sometimes referred to as the *Wiener-filtered map*. Since the degrees of freedom of the CR system scale with the number of pixels, brute force solutions are out of bounds except for very low-resolution data sets. However, it is computationally feasible to multiply an arbitrary vector with the system matrix by repeatedly changing basis functions (i.e., spherical harmonic

transforms, SHTs), so that the system can be solved using iterative linear equation solvers. The main problem is to optimize the convergence rate of these solvers to produce a solution in a timely manner.

Commander (Eriksen et al. 2004), the CMB Gibbs sampler mentioned above, solves the CR system through the Conjugate Gradient (CG) method, using a combination of a block preconditioner on large angular scales and a diagonal preconditioner on small angular scales. While this approach was successful for analyzing *WMAP* observations (O'Dwyer et al. 2004; Eriksen et al. 2007a, 2007b, 2008a), the higher signal-to-noise level of data from more recent experiments like Planck effectively halts convergence of the solver. Indeed, as we will see in Section 2.4, the number of CG iterations intrinsically scales with the signal-to-noise ratio of a given data set, limiting the utility of CG for data sets such as these. To produce the low-ℓ power spectrum likelihood for the Planck mission, for example, the data had to be downgraded to low angular resolution and a substantial amount of regularization noise added (Planck Collaboration 2013b). Even then, several thousands of CG iterations were required for convergence. To go to full angular resolution with this scheme is simply not computationally feasible.

A somewhat better approach was described by Smith et al. (2007), who applied the CG method recursively, such that a CG solution on a coarse grid was used as the preconditioner for CG on a finer grid. We are not aware of any head-to-head comparisons of this method versus the one described by Eriksen et al. (2004), but our understanding is that, although it is faster, it still scales with the signal-to-noise ratio of the data set, and therefore does not inherently fix the fundamental convergence problems for high-sensitivity, high-resolution analysis.

More recently, Elsner & Wandelt (2012, 2013) introduced a stationary iterative method for solving the CR equation. They did not quote the usual statistics for convergence, such as total reduction in residual and error, however. Not knowing the accuracy of their solution, we are unable to compare the efficiency of their method directly to ours. While they do quote the change in the χ^2 statistic of the posterior probability density *between successive iterations*, iterative methods (and stationary methods in particular) are vulnerable to breaking down in terms of convergence rate well before reaching true convergence. Also, the χ^2 explicitly ignores large scales under the mask. While there certainly are applications where this is acceptable, CMB Gibbs sampling is not one of them, since it explicitly iterates between considering the CMB signal a sample from the posterior, which mostly ignores the masked area, and a sample from the prior, which gives equal weight to the masked area.

In this paper we present a new solver for the CR system that is radically different from the CG approach, and instead builds on the multi-level (or multi-grid) framework. These algorithms are best known in the astrophysics community as solvers for elliptical partial differential equations (PDEs), although they are in fact more generally applicable to solving many types of linear systems (Brandt 2001). We apply multi-level theory to the CR equation (although the algorithm is not entirely traditional), and show that the resulting algorithm converges to the exact solution with only a handful of iterations even for the most sensitive Planck channel. Most importantly, and contrary to the CG solver, the convergence rate is nearly independent of the signal-to-noise ratio of the data set.

Multi-level methods have been explored before in the CMB community for the purposes of map-making. Doré et al. (2001) described a standard multi-grid method for map-making, although it was eventually unable to compete with standard CG and approximate map-makers. Grigori et al. (2012) also presented a promising two-level CG preconditioner for map-making based on the domain-decomposition method in Havé et al. (2013). The map-making equation is different from CR equation, however, in that one does not solve for the CMB signal under a mask. As we will see in Section 2.4, it is this feature in particular that makes convergence difficult to achieve on the CR system.

2. EXPLORING THE CR LINEAR SYSTEM

2.1. Matrix Notation for Spherical Harmonic Transforms

The details of changing between pixel domain and spherical harmonic (SH) domain are usually glossed over in the literature. Since we will be solving a large linear system that couples signals on all scales—from individual pixels to the full-sky—it is of the utmost importance to be precise about how these conversions are performed. If implemented incorrectly, even small pixel-scale errors can lead to overall divergence of the entire method.

There is no perfect grid on the sphere, and in choosing a particular one, a number of trade-offs must be considered. In our current implementation we adopt both the HEALPix[5] pixelization (Górski et al. 2005) and the Gauss–Legendre spherical grid (Reinecke 2011, and references therein). The HEALPix software package contains routines that are useful for our pixel-domain computations, while the latter is required for accurate evaluation of Equation (2) below.

Given such a grid on the sphere (by which we mean a set of positions $\hat{n}_i$ on the sky), we can use *SH synthesis* to transform a field expressed in spherical harmonic basis, with coefficients $s_{\ell m}$, to a field sampled on the sphere,

$$\hat{s}(\hat{n}_i) = \sum_{\ell=0}^{\ell_{\max}} \sum_{m=-\ell}^{\ell} s_{\ell m} Y_{\ell m}(\hat{n}_i). \quad (1)$$

We will write this operation in matrix form as $\widehat{\mathbf{s}} = \mathbf{Y}\mathbf{s}$, where $\mathbf{Y}$ encodes the value of the SHs evaluated at each $\hat{n}_i$ of the chosen grid. Note that $\mathbf{Y}$ is not a square matrix, as spherical grids need to over-sample the signal to faithfully represent it up to some bandlimit $\ell_{\max}$. In typical applications there are between 30% and 100% more pixels along the rows of $\mathbf{Y}$ than there are SH coefficients along the columns. For the purposes of our method, it will turn out that we need to under-pixelize the signal instead, so there will be more columns than rows in $\mathbf{Y}$.

The opposite action of converting from pixel basis to harmonic basis is *SH analysis*, which generally takes the quadrature form

$$s_{\ell m} = \int_{4\pi} Y^*_{\ell m}(\hat{n}) \hat{s}(\hat{n}) d\Omega \approx \sum_{i=1}^{N_{\rm pix}} Y^*_{\ell m}(\hat{n}_i) w_i \hat{s}(\hat{n}_i), \quad (2)$$

where w_i combines quadrature weights and pixel area. Similar to the synthesis case, this operation can be written in matrix form as $\mathbf{s} = \mathbf{Y}^T \mathbf{W} \widehat{\mathbf{s}}$, where $W_{ij} = w_i \delta_{ij}$. A crucial feature of our method is the ability to (for the most part) avoid SH analysis, however. Instead, we will rely on *adjoint SH synthesis*, $\mathbf{Y}^T$, which simply appears algebraically as the transpose of $\mathbf{Y}$.

[5] http://healpix.sourceforge.net

Note that, unlike in the case of the more famous discrete Fourier transform, $\mathbf{Y}$ is not a square orthogonal matrix, and synthesis and analysis differ by more than transposition and a scale factor. One may in some situations have that $\mathbf{Y}^T\mathbf{W}\mathbf{Y} = \mathbf{I}$, but this depends on both $\ell_{\rm max}$, $N_{\rm pix}$ and the spherical grid.

The action of applying $\mathbf{Y}$, $\mathbf{Y}^T$, $\mathbf{Y}^T\mathbf{W}$ or $\mathbf{W}\mathbf{Y}$ to a vector is in general referred to as a SHT. Carefully optimized libraries are available that perform SHTs in $O(\ell_{\rm max} N_{\rm pix})$ time; we use the `libsharp` library (Reinecke & Seljebotn 2013).

2.2. Data Model

We now define our data model, and assume from the beginning that the CMB is Gaussian and isotropic (e.g., Planck Collaboration 2013c). Following the notation of Eriksen et al. (2004), it is convenient to define the CMB signal to be a vector $\mathbf{s}$ of SH coefficients, in which case the associated covariance matrix $\mathbf{S}$ is given by

$$S_{\ell m,\ell' m'} = \delta_{\ell\ell'}\delta_{mm'}C_\ell,$$

where C_ℓ is the CMB power spectrum.

Using the notation of the previous section, the model for the observed sky map pixel vector, $\mathbf{d}$, is

$$\mathbf{d} = \mathbf{Y}_{\rm obs}\mathbf{B}\mathbf{s} + \mathbf{n}, \tag{3}$$

where $\mathbf{B}$ denotes beam-smoothing and the pixel window function, $\mathbf{n}$ is Gaussian instrumental noise, and the subscript of $\mathbf{Y}_{\rm obs}$ indicates projection to the pixelization of the map $\mathbf{d}$.

We assume a symmetric instrumental beam, so that the beam matrix $\mathbf{B}$ is a diagonal matrix given by $B_{\ell m,\ell' m'} = b_\ell p_\ell \delta_{\ell\ell'}\delta_{mm'}$, where b_ℓ is the instrumental beam and p_ℓ the pixel window function of the observed grid. We also assume white instrumental noise, such that the noise covariance matrix, $\mathbf{N}$, is diagonal. We discuss the likely impact of asymmetric beams and correlated noise in Section 5.

Discretization of the model is done simply by picking some $\ell_{\rm max}$ for the $\mathbf{s}$ vector. The noise vector $\mathbf{n}$ is related to the map-making process, averaging the noise of time-ordered data that fall within the same pixel, and so is inherently discrete rather than being a discretization of any underlying field. As already mentioned above, no SH analysis of $\mathbf{d}$ (and therefore $\mathbf{n}$) is required when solving the CR system; rather, one solves for the projected $\mathbf{s}$, and so the noise treatment is always perfectly consistent with the assumed model.

2.3. The CR Linear System

Given the data model above, we are interested in exploring the Bayesian posterior distribution $p(\mathbf{s}|\mathbf{d}, C_\ell)$, the CMB signal given the data and CMB power spectrum. Let us first define

$$\mathbf{A} \equiv \mathbf{S}^{-1} + \mathbf{B}\mathbf{Y}_{\rm obs}^T\mathbf{N}^{-1}\mathbf{Y}_{\rm obs}\mathbf{B}, \tag{4}$$

where in what follows we will refer to the first term as the *prior term*, and the second as the *inverse-noise term*. It can be shown that if we now solve the *CR system*

$$\mathbf{A}\mathbf{x} = \mathbf{B}\mathbf{Y}_{\rm obs}^T\mathbf{N}^{-1}\mathbf{d}, \tag{5}$$

the solution $\mathbf{x}$ will be the maximum likelihood estimate of $\mathbf{s}$. Alternatively, if particular random fluctuation terms are added to the right-hand side of Equation (5), the solution $\mathbf{x}$ will instead be samples from the posterior (Jewell et al. 2004; Wandelt et al. 2004). Since $b_\ell \to 0$ as ℓ increases, the diagonal prior term will at some point dominate the dense inverse-noise term, so that truncation at sufficiently high $\ell_{\rm max}$ does not affect the solution of the system.

As stressed in Section 2.1, $\mathbf{Y}_{\rm obs}^T$ denotes SH *adjoint synthesis*, and not SH analysis. Pixels that are masked out, typically due to strong foreground contamination, are simply missing from the data vector $\mathbf{d}$, and so the corresponding rows are not present in $\mathbf{Y}_{\rm obs}$. This means $\mathbf{Y}_{\rm obs}$ is not an orthogonal matrix, but that is not a concern since we never perform SH analysis of pixels on the observation grid. The solution $\mathbf{x}$ is still well-defined everywhere on the sky due to the prior term $\mathbf{S}^{-1}$. This is typically implemented by introducing zeroes in $\mathbf{N}^{-1}$ rather than removing rows of $\mathbf{Y}_{\rm obs}$, which has the statistical interpretation of giving those pixels infinite variance. The two interpretations are algebraically equivalent.

2.4. Eigenspectrum and CG Performance

The CR system in Equation (4) is symmetric and positive definite, which suggests the use of the CG algorithm. For the behavior of CG and other Krylov methods, we are primarily interested in the eigenspectrum after preconditioning (Shewchuk 1994, and references therein), i.e., the eigenspectrum of $\mathbf{M}\mathbf{A}$, where $\mathbf{M} \approx \mathbf{A}^{-1}$. To illustrate the fundamental problem with the CG algorithm for the application considered here, we show in Figure 1 the eigenspectrum of a low-resolution setup, using a diagonal preconditioner. This case corresponds to a simulation of the 143 GHz Planck frequency map (Planck Collaboration 2013a), downgraded to an angular resolution of $5\overset{\circ}{.}4$, bandwidth-limited at $\ell_{\rm max} = 95$, and with a mask applied that removes 40% of the sky. The overall shape of the spectrum appears to be mostly independent of the resolution, with a significant fraction of degrees of freedom found in the tails. This behavior is representative of that found in real-world cases.

The problematic feature is the exponential drop in the eigenvalues seen to the left of the figure. Theoretical results indicate that the CG search needs at least one iteration per eigenvalue located in exponentially increasing parts of the eigenspectrum (Axelsson & Lindskog 1986a, 1986b). This leads to extreme degradation of CG performance, which is indeed what has been observed with Commander on high-resolution, high-sensitivity data.

The exponential spectral feature is due to large-scale modes under the mask. For all but the smallest angular scales, the $\mathbf{N}^{-1}$ term dominates by many orders of magnitude, so that the $\mathbf{S}^{-1}$ term is hardly seen at all. However, vectors that only build-up signal under the mask after beam-smoothing will only see the $\mathbf{S}^{-1}$ term of the matrix, as the $\mathbf{N}^{-1}$ term vanishes in that case. The eigenvectors corresponding to the smallest eigenvalues are therefore characterized by having large scales localized within the mask. Moreover, the solution under the mask is constrained by the values at the mask edge, meaning the $\mathbf{N}^{-1}$ term takes effect, and this constraint is harder closer to the edges. The result is an exponentially falling eigenspectrum, rather than separated clusters of eigenvalues that CG could more easily deal with.

Phrased differently, for data having a high signal-to-noise ratio, the pixels near the edge of the mask carry a large predictive power on the signal inside the mask—a signal that must be reconstructed by the CG algorithm by navigating through a nearly degenerate system. In total, the CG convergence rate is determined by a combination of the overall signal-to-noise ratio and the size and shape of the mask. We have been unable to achieve proper convergence with this method for the

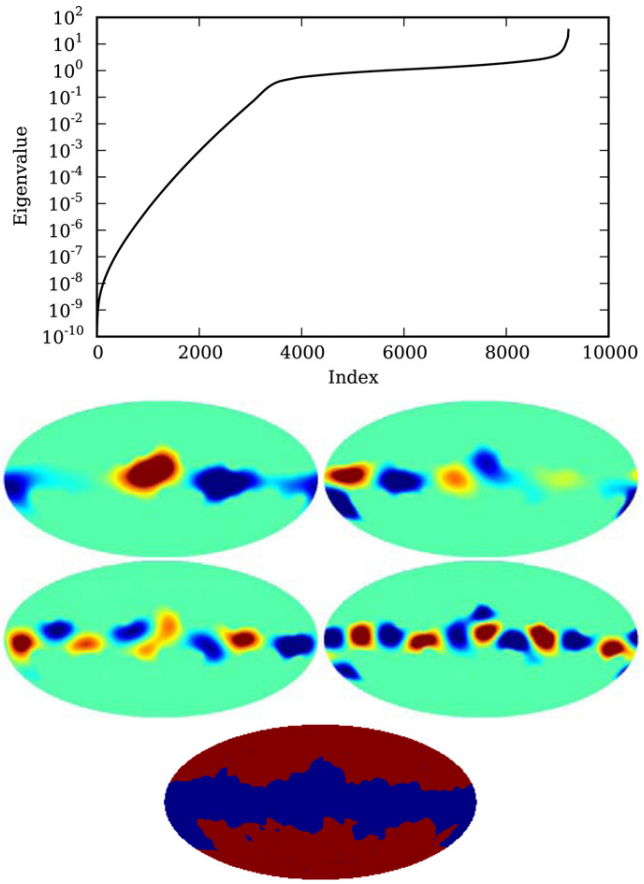


Figure 1. Eigendecomposition of the CR system using a diagonal preconditioner. Top panel: the eigenvalues of $\mathrm{diag}(\mathbf{A})^{-1}\mathbf{A}$ for the 143 GHz Planck channel with a mask covering 40% of the sky, smoothed with a 5.°6 FWHM beam and truncated at $\ell_{\max} = 95$. Bottom panel: a selection of eigenvectors corresponding to very low eigenvalues. The structure of the mask (bottom) is clearly visible in the eigenvectors.

(A color version of this figure is available in the online journal.)

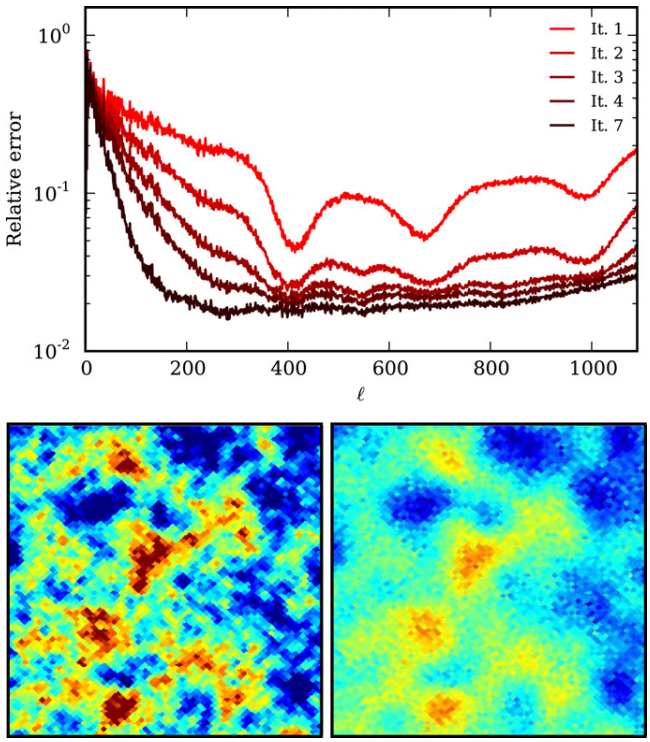


Figure 2. Effect of the error smoother/approximate inverse $\widehat{\mathbf{M}}$. Top: relative error $\|\mathbf{x}_\ell - \mathbf{x}_{\mathrm{true},\ell}\|/\|\mathbf{x}_{\mathrm{true},\ell}\|$. For each iteration, the error smoother developed in Section 3.5 is applied on a HEALPix $N_{\mathrm{side}} = 512$ grid. The error smoother is only able to get closer to the solution for some part of the frequency spectrum, and quickly stagnates since no improvement is made to the larger or smaller scales. Bottom: the left patch shows the initial error when starting at $\mathbf{x} = \mathbf{0}$, while the right patch shows the error after the first iteration. The remaining large-scale errors can be represented on a coarser grid. This observation leads to the multi-level algorithm.

(A color version of this figure is available in the online journal.)

signal-to-noise ratio of a Planck-like experiment, for example, independent of preconditioners or number of iterations; downgrading and adding regularization noise is required to produce robust results.

3. THE MULTI-LEVEL SOLVER

3.1. Motivation for a Multi-level Method

The matrix $\mathbf{A}$ of Equation (4) is defined in SH domain, and describes the coupling strength between pairs of (ℓ, m) and (ℓ', m'). Except in unrealistic scenarios with very simple instrumental noise and mask, we have found no pattern in the magnitudes of the matrix coefficients $A_{\ell m,\ell' m'}$ that is consistent enough to be exploited in a solver.

By moving to pixel domain, however, we can *create* such an exploitable pattern in the magnitudes of the matrix coefficients. In Section 3.3 we will construct a corresponding pixel-domain matrix $\widehat{\mathbf{A}}$ that is *localized*, in the sense that $\widehat{A}_{ij}$ has small magnitude (less than 1% of $\widehat{A}_{ii}$) unless pixels i and j are very close together on the sphere.

It is no surprise that the $\mathbf{N}^{-1}$ term of Equation (4) enjoys this property, since we have assumed that instrumental noise is uncorrelated between pixels. When it comes to the $\mathbf{S}^{-1}$ term, we note that $1/C_\ell$ is roughly proportional to $\ell(\ell+1)$, at least for $\ell \lesssim 1000$. These are the eigenvalues of the Laplacian on the sphere, with $\mathbf{Y}$ being the corresponding eigenbasis. Therefore we can hope that a projection of $\mathbf{S}^{-1}$ to pixel domain should be close to a Laplacian. The Laplacian is often approximated with a matrix where $\widehat{A}_{ij} = 0$ unless pixel i and j are neighbors or $i = j$. While our case will be less perfect, it still suggests that multi-level methods can be very efficient, since those are highly successful for PDEs involving the Laplacian.

In Section 3.5, we exploit the localization properties in pixel domain to develop an approximate inverse $\widehat{\mathbf{M}} \approx \widehat{\mathbf{A}}^{-1}$. Figure 2 demonstrates the use of this approximate solver as part of a simple stationary method

$$\mathbf{x} \leftarrow \mathbf{x} + \widehat{\mathbf{M}}(\mathbf{b} - \widehat{\mathbf{A}}\mathbf{x}), \tag{6}$$

where we initialize $\mathbf{x} \leftarrow \mathbf{0}$ and then iteratively update the solution. Note that if we replace $\widehat{\mathbf{M}}$ with $\mathrm{diag}(\mathbf{A})^{-1}$, Equation (6) represents what are known as Jacobi iterations.

The problem that is evident from Figure 2 is that $\widehat{\mathbf{M}}$ will only make improvements to one part of the frequency spectrum—namely, the highest frequencies that can be represented on the grid used. This is the typical case when multi-level methods are applied; iterations of the form of Equation (6) are usually only efficient at resolving the relations between pixels/elements that are strongly coupled, which, when $\widehat{\mathbf{A}}$ is localized, translates to resolving the solution at highest frequencies. Little or no improvement is made between pixels that are weakly or indirectly

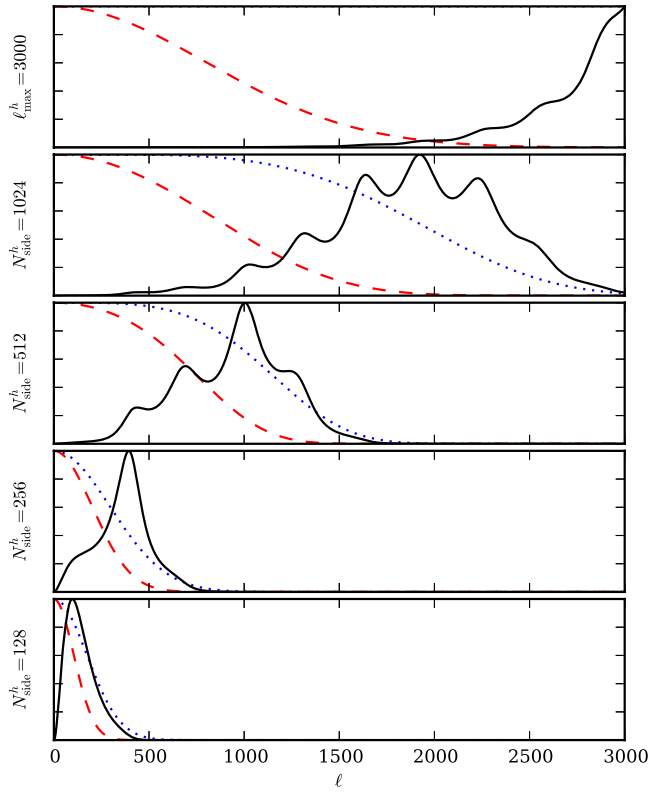


Figure 3. Effect of filters in harmonic domain for the top five levels. For each level H, starting from the original system of Equation (4) at the top, we plot the transfer filter $f^H_{h,\ell}$ (dotted blue), the filtered prior $(\tilde{f}^H_\ell)^2/C_\ell$ (solid black), and an approximation to the diagonal of the inverse-noise term (dashed red). Functions are normalized to an arbitrary scale (see Figure 4 for the absolute scale). Note how the prior term on the pixel levels looks superficially similar to wavelets/needlets in harmonic domain (Scodeller et al. 2011, and references therein). The real-space transform is also similar to wavelets/needlets (not plotted).

(A color version of this figure is available in the online journal.)

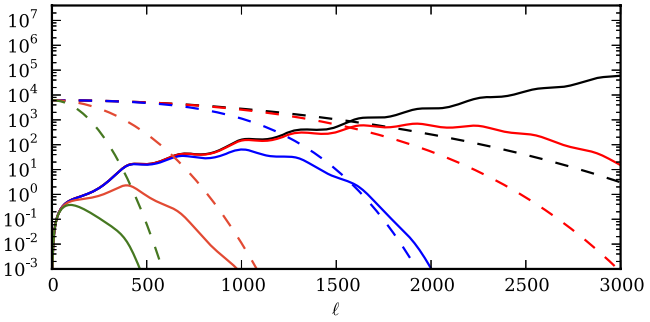


Figure 4. Same as Figure 3, but all levels plotted together with a logarithmic scale and with absolute normalization. We plot the filtered prior $(\tilde{f}^H_\ell)^2/C_\ell$ (solid), and the diagonal of the inverse-noise term for 26 μK constant rms and no mask (dashed). This noise level corresponds to the average of the rms map of the 143 GHz Planck band. The levels are: the original system (black), $N^h_{\rm side}=1024$ (red), $N^h_{\rm side}=512$ (blue), $N^h_{\rm side}=256$ (orange), and $N^h_{\rm side}=128$ (green). Note the effect of the filters on the signal-to-noise ratio; harmonic scales go from being data-dominated to noise-dominated at the point where the solid and dashed lines intersect.

(A color version of this figure is available in the online journal.)

coupled in $\widehat{\mathbf{A}}$, so that no improvement is made to the coarser scales. Put another way, the error, $\mathbf{e}\equiv\mathbf{x}-\mathbf{x}_{\rm true}$, has its high-frequency components reduced, while the low frequencies are left relatively unaffected. The approximate inverse $\widehat{\mathbf{M}}$ is therefore dubbed a *smoother* in multi-level terminology. We will use the term *error smoother* to distinguish it from the act of applying a low-pass filter (which is instead called *restriction* in this context).

The key is now to project the matrix $\mathbf{A}$ to pixel grids at different resolutions, producing a set of matrices $\widehat{\mathbf{A}}_h$, where h is a level indicator. For each $\widehat{\mathbf{A}}_h$ we construct a corresponding error smoother $\widehat{\mathbf{M}}_h\approx\widehat{\mathbf{A}}_h^{-1}$ that resolves the errors in one region of the frequency spectrum only. Using these levels together, we arrive at a method that converges very well over the entire frequency spectrum.

3.2. The Multi-level Algorithm

In this section we give a brief overview of multi-level theory, together with the specification of our algorithm. For a more detailed introduction to multi-grid methods, consult one of the number of standard texts (e.g., Hackbush 1985). Ingredients of multi-level algorithms are:

1. A set of bases to project the linear system into in order to work on different parts of the solution. Usually these form a hierarchy of levels from finest to coarsest, so that each level solves for different frequencies of the solution. It is customary to label levels relatively, using h for the current level and H for the coarser level.
2. A way to transfer vectors between the different levels. The *restriction* operator, $\mathbf{I}^H_h$, takes a vector from a finer level to a coarser level, while the *interpolation* operator $\mathbf{I}^h_H$ works in the opposite direction. For symmetric systems, one often takes $\mathbf{I}^h_H=(\mathbf{I}^H_h)^T$.
3. One linear operator (left-hand side matrix) for each level. For the case where interpolation is chosen to be transposed restriction, these are often defined recursively as
$$\mathbf{A}_H=\mathbf{I}^h_H\mathbf{A}_h\left(\mathbf{I}^h_H\right)^T \tag{7}$$
for the projection of a fine matrix $\mathbf{A}_h$ to a coarser matrix $\mathbf{A}_H$.
4. An *error smoother* $\mathbf{M}_h$ for each $\mathbf{A}_h$ that removes the higher frequencies of the error on level h, as discussed in the previous section.

Multi-level algorithms are often implemented on a grid or a tessellation in real space, with a sparse linear operator, and using averages of neighboring points as the restriction operator $\mathbf{I}^H_h$. In our case, $\widehat{\mathbf{A}}_h$ on each level is not sparse, and, at least without approximations, multiplying $\widehat{\mathbf{A}}_H=\mathbf{I}^h_H\widehat{\mathbf{A}}_h\mathbf{I}^H_h$ with a vector would be computationally very expensive on the coarser levels as it would require interpolating back to the highest-resolution grid.

To avoid this cost, we instead define our levels in SH domain. Let $\tilde{f}^h_\ell$ be a SH low-pass filter that emphasizes one part of the frequency spectrum, and define $\mathbf{F}_h$ to be a diagonal matrix with elements $\tilde{f}^h_\ell$. We then define

$$\mathbf{A}_h\equiv\mathbf{F}_h\mathbf{A}\mathbf{F}^T_h\equiv\mathbf{D}_h+\mathbf{B}_h\mathbf{Y}^T_{\rm obs}\mathbf{N}^{-1}\mathbf{Y}_{\rm obs}\mathbf{B}_h, \tag{8}$$

where the prior term $\mathbf{D}_h$ is diagonal with entries given by $(\tilde{f}^h_\ell)^2/C_\ell$ and the modified beam matrix $\mathbf{B}_h$ is diagonal with elements given by $\tilde{f}^h_\ell b_\ell p_\ell$. In this case, the system is bandlimited by some $\ell^h_{\rm max}\leqslant\ell_{\rm max}$, above which $\tilde{f}^h_\ell=0$. Figures 3 and 4 show the filters used in our setup; we discuss the choice of filters further in Section 3.3.

With this choice, we can clearly satisfy the multi-level hierarchy of Equation (7) by choosing the restriction operator $\mathbf{I}_h^H$ as an $(\ell_{\rm max}^H+1)^2$-by-$(\ell_{\rm max}^h+1)^2$ block matrix, where the block for $\ell \leqslant \ell_{\rm max}^H$ is diagonal with entries

$$f_{h,\ell}^H \equiv \frac{\tilde{f}_\ell^H}{\tilde{f}_\ell^h}, \tag{9}$$

and the block for $\ell_{\rm max}^H < \ell \leqslant \ell_{\rm max}^h$ is zero.

As already mentioned in Section 3.1, the error smoother that we have available, $\widehat{\mathbf{M}}_h$, is defined in pixel domain. For every SH level we therefore tag on a corresponding sibling pixel level with matching HEALPix resolution $N_{\rm side}^h$. The result is the following level structure:

$$\begin{array}{ccc} \text{SH at } \ell_{\rm max}^h = 3000 & \longleftrightarrow & \text{Pixels at } N_{\rm side}^h = 1024 \\ \updownarrow & & \\ \text{SH at } \ell_{\rm max}^h = 2048 & \longleftrightarrow & \text{Pixels at } N_{\rm side}^h = 512 \\ \updownarrow & & \\ \text{SH at } \ell_{\rm max}^h = 1280 & \longleftrightarrow & \text{Pixels at } N_{\rm side}^h = 256 \\ \updownarrow & & \\ \vdots & & . \end{array}$$

The arrows indicate that transfers between different scales happen only through the SH levels. As emphasized in Section 2.1, no SH analysis operations are performed in each conversion (only synthesis and adjoint synthesis operations), and the implied under-pixelization in the above scheme is therefore numerically unproblematic.

The full details of how to properly move between the levels to obtain a solution is given in pseudo-code in Figure 5. We highlight some aspects in what follows.

Assume that we are currently on some SH level h (where the original equation is simply the top level), with corresponding system

$$\mathbf{A}_h \mathbf{x}_{{\rm true},h} = \mathbf{b}_h. \tag{10}$$

We start with some search vector $\mathbf{x}_h$ (initialized to zero), and want to improve it to get closer to the true value $\mathbf{x}_{{\rm true},h}$. In order to make use of $\widehat{\mathbf{M}}$, we must now move to the corresponding pixel level. Our chosen restriction operator, denoted $\mathbf{Y}_h$, is SH synthesis to a HEALPix grid[6] of resolution $N_{\rm side}^h$. The key to efficient multi-level solvers is to transfer the residual vector $\mathbf{r}_h$, and *not* the search vector $\mathbf{x}_h$;

$$\mathbf{r}_h \leftarrow \mathbf{b}_h - \mathbf{A}_h \mathbf{x}_h \tag{11}$$

$$\widehat{\mathbf{r}}_h \leftarrow \mathbf{Y}\mathbf{r}_h, \tag{12}$$

where $\widehat{\mathbf{r}}_h$ is the pixel-domain projection of $\mathbf{r}_h$. Then, we approximately solve the projected system for a *correction vector* $\widehat{\mathbf{c}}_h$,

$$\widehat{\mathbf{c}}_h \leftarrow \widehat{\mathbf{M}}\widehat{\mathbf{r}}_h \approx \left(\mathbf{Y}_h \mathbf{A}_h \mathbf{Y}_h^T\right)^{-1} \widehat{\mathbf{r}}_h, \tag{13}$$

where the computation of $\widehat{\mathbf{M}}\widehat{\mathbf{r}}_h$ is further described in Section 3.5. The approximation is better for small scales than for large scales. Finally, we let the interpolation operator be the transpose of restriction, $\mathbf{Y}^T$, so that the correction is brought over to the SH search vector by adjoint SH synthesis,

$$\mathbf{x}_h \leftarrow \mathbf{x}_h + \mathbf{Y}^T \widehat{\mathbf{c}}_h. \tag{14}$$

Together, these steps act as the error smoothing of a SH level, labeled pre- and post-smoothing in Figure 5. Here we have motivated the procedure as arising from moving between levels, but the idea of solving for a correction in a projected system arises in many settings (Tang et al. 2009), and other variations on this theme may prove fruitful in the future.

The vertical movement between coarser and finer levels follows the same pattern, but uses the restriction operator $\mathbf{I}_h^H$ defined in Equation (9) instead of pixel projection $\mathbf{Y}$. First, a fine residual is computed and restricted (i.e., low-pass filtered) to the coarser level,

$$\mathbf{r}_h \leftarrow \mathbf{b}_h - \mathbf{A}_h \mathbf{x}_h, \tag{15}$$

$$\mathbf{r}_H \leftarrow \mathbf{I}_h^H \mathbf{r}_h. \tag{16}$$

Then, a coarse correction $\mathbf{c}_H$ is sought that approximates the solution of the coarse system

$$\mathbf{I}_h^H \mathbf{A}_h \left(\mathbf{I}_h^H\right)^T \mathbf{c}_H = \mathbf{A}_H \mathbf{c}_H = \mathbf{r}_H. \tag{17}$$

CR-Cycle$(h, \mathbf{x}, \mathbf{b})$:
Inputs:
h – The current level
$\mathbf{x}$ – Starting vector
$\mathbf{b}$ – Right-hand side

H denotes the coarser level relative to h.
Output:
Improved solution vector $\mathbf{x}$

if h is bottom level:
  $\mathbf{x} \leftarrow \mathbf{A}_h^{-1}\mathbf{b}$ *By dense Cholesky*
else:
  $\mathbf{x} \leftarrow \mathbf{x} + \mathbf{Y}_h^T \widehat{\mathbf{M}}_h \mathbf{Y}_h (\mathbf{b} - \mathbf{A}_h \mathbf{x})$ *Pre-smoothing*
  $\mathbf{r}_H \leftarrow \mathbf{I}_h^H (\mathbf{b} - \mathbf{A}_h \mathbf{x})$ *Restricted residual*
  $\mathbf{c}_H \leftarrow \mathbf{0}$ *Coarse correction*
  repeat $n_{\rm rec}^h$ times:
    $\mathbf{c}_H \leftarrow$ CR-Cycle$(H, \mathbf{c}_H, \mathbf{r}_H)$ *Recurse*
  $\mathbf{x} \leftarrow \mathbf{x} + (\mathbf{I}_h^H)^T \mathbf{c}_H$ *Apply correction*
  $\mathbf{x} \leftarrow \mathbf{x} + \mathbf{Y}_h^T \widehat{\mathbf{M}}_h \mathbf{Y}_h (\mathbf{b} - \mathbf{A}_h \mathbf{x}_)$ *Post-smoothing*
return $\mathbf{x}$

CR-Solve$(\mathbf{b}, \epsilon)$:
Inputs:
$\mathbf{b}$ – Right-hand side
ϵ – Requested improvement in residual
Output:
Approximate solution $\mathbf{x}$

$\mathbf{x} \leftarrow \mathbf{0}$
repeat:
  $\mathbf{x} \leftarrow$ CR-Cycle$(1{\rm st}, \mathbf{x}, \mathbf{b})$
  $\mathbf{r} \leftarrow \mathbf{b} - \mathbf{A}\mathbf{x}$ *Reused in next CR-Cycle*
  if $\mathbf{r}^T \mathbf{S}^{-1} \mathbf{r} < \epsilon \mathbf{b}^T \mathbf{S}^{-1} \mathbf{b}$: *Improvement relative to* C_ℓ
    return $\mathbf{x}$

Figure 5. Multi-level CR solver. The matrices involved are defined in the main text. In place of the simple iteration scheme of CR-Solve, one can use CR-Cycle as a preconditioner within another solver, such as CG. By varying the $n_{\rm rec}^h$ parameter, a variety of solver cycles can be constructed, such as a V-cycle ($n_{\rm rec}^h = 1$) or W-cycle ($n_{\rm rec}^h = 2$). Note that, for simplicity, the top-level diagonal error correction is omitted; see the main text.

[6] The pixel level is actually coarser than the SH level, because $\ell_{\rm max}^h$ must be chosen so high that the grid cannot resolve all the scales of the projected field. See Section 3.4.

Except for at the bottom level, this happens by initializing a search vector $\mathbf{c}_H$ to zero and recursively applying the algorithm. Finally, the correction is interpolated and applied to our current search vector,

$$\mathbf{x}_h \leftarrow \mathbf{x}_h + \mathbf{I}_H^h \mathbf{c}_H. \tag{18}$$

Using this idea of transferring residuals and corrections between levels with different bases, one can form a variety of multi-level *cycles*, moving between the levels in different patterns. Our choice in the end is a W-cycle on the coarser levels and a V-cycle on the finer levels, as described in Section 4.1 and the pseudo-code.

In addition to the pixel levels described above, the top and bottom levels are special. The smallest scales ($\ell \gtrsim 2200$ in our experimental setup) are strongly noise-dominated, making the SH domain matrix $\mathbf{A}$ nearly diagonal. As a result, we do not project to a pixel grid, but simply use $\mathrm{diag}(\mathbf{A})^{-1}$ as the error smoother. Note, however, that this process would destroy the solution on scales that are not entirely noise-dominated, and so we first apply a high-pass filter to the correction vector before applying it to the solution search vector. For the largest scales ($\ell \leqslant 40$), we do not project to pixel domain either, but simply solve $\mathbf{Ax} = \mathbf{b}$ restricted to $\ell \leqslant 40$ by explicitly computing the matrix entries and using a simple Cholesky solver.

For the top solver level, we need to compute the diagonal of $\mathbf{Y}_{\mathrm{obs}}^T \mathbf{N}^{-1} \mathbf{Y}_{\mathrm{obs}}$ in SH domain, and for the bottom solver level we similarly need all entries of $\mathbf{Y}_{\mathrm{obs}}^T \mathbf{N}^{-1} \mathbf{Y}_{\mathrm{obs}}$ for ℓ up to some ℓ_{dense}. While such entries can be computed using Wigner 3j-symbols (Hivon et al. 2002; Eriksen et al. 2004), the following procedure has some significant advantages. First, while the computational scaling is the same, it is much faster in practice, in particular due to the optimized code for associated Legendre polynomials $P_{\ell m}$ available in `libpsht` (Reinecke 2011). Second, it is accurate to almost machine precision for any grid, whereas the method relying on Wigner 3j-symbols relies on approximation by evaluation of an integral, and is therefore inaccurate for low-resolution HEALPix grids.

Let ξ_{kj} be the jth of J_k pixels on ring k in the masked inverse-noise map. One can then evaluate

$$\begin{aligned}\left(\mathbf{Y}_{\mathrm{obs}}^T \mathbf{N}^{-1} \mathbf{Y}_{\mathrm{obs}}\right)_{\ell_1 m_1, \ell_2 m_2} &= \sum_k \sum_{j=1}^{J_k} \xi_{kj} Y_{\ell_1 m_1}(\theta_j, \phi_{kj}) Y^*_{\ell_2 m_2}(\theta_j, \phi_{kj}) \\ &= \sum_k \widetilde{P}_{\ell_1 m_1}(\cos\theta_k) \widetilde{P}_{\ell_2 m_2}(\cos\theta_k) \sum_{j=1}^{J_k} \xi_{kj} e^{i(m_1 - m_2)\phi_{kj}},\end{aligned}$$

where the normalized associated Legendre function is

$$\widetilde{P}_{\ell m}(\cos\theta) = \sqrt{\frac{2\ell+1}{4\pi} \frac{(\ell-m)!}{(\ell+m)!}} P_{\ell m}(\cos\theta). \tag{19}$$

The inner sum can be precomputed for each ring k and every $(m_1 - m_2)$ by discrete Fourier transforms, allowing the evaluation of matrix elements in $O(N_{\mathrm{ring}}) = O(\ell_{\mathrm{max}})$ time. In the case that we want a dense low-ℓ block, the procedure to downgrade the inverse-noise operator from Section 3.4 should be applied first, to reduce the computational cost from $O(\ell_{\mathrm{dense}}^2 \ell_{\mathrm{max}})$ to $O(\ell_{\mathrm{dense}}^3)$.

3.3. Filter Selection and Pixel-domain Localization

So far we have not specified the exact form of the low-pass filters $\tilde{f}_\ell^h$ required for every level. It turns out that careful selection of these filters is essential to ensure that the pixel projection of $\mathbf{A}_h$ is localized, and hence that the construction of an efficient error smoother is possible.

As indicated in Equation (13), the SH system $\mathbf{A}_h$ on each level h is projected to pixel domain with

$$\widehat{\mathbf{A}}_h \equiv \mathbf{Y}_h \mathbf{A}_h \mathbf{Y}_h^T = \widehat{\mathbf{D}}_h + \widehat{\mathbf{B}}_h^T \mathbf{N}^{-1} \widehat{\mathbf{B}}_h, \tag{20}$$

where the prior and pixelized beam terms are this time given by (respectively)

$$\widehat{\mathbf{D}}_h = \mathbf{Y}_h \mathbf{F}_h \mathbf{S}^{-1} \mathbf{F}_h \mathbf{Y}_h^T \tag{21}$$

$$\widehat{\mathbf{B}}_h = \mathbf{Y}_{\mathrm{obs}} \mathbf{B}\, \mathbf{F}_h \mathbf{Y}_h^T. \tag{22}$$

Note that the pixelization along the rows of $\widehat{\mathbf{B}}_h$ is the observational grid, while the pixelization down the columns is that of the current level.

The matrices $\widehat{\mathbf{D}}_h$ and $\widehat{\mathbf{B}}_h$ are both rotationally invariant. By the addition theorem of SHs, the coupling strength between two points on the sphere separated by angular distance θ is given by

$$g(\theta) = \sum_\ell \frac{2\ell+1}{4\pi} g_\ell P_\ell(\cos\theta), \tag{23}$$

where we insert $g_\ell = (\tilde{f}_\ell^h)^2 / C_\ell$ for $\widehat{\mathbf{D}}_h$ and $g_\ell = \tilde{f}_\ell^h b_\ell p_\ell$ for $\widehat{\mathbf{B}}_h$. The pixel-domain localization of such matrices depends entirely on g_ℓ. In our experience, the g_ℓ that lead to localized matrices in pixel domain tend to be flat or polynomially increasing before an exponential drop. Since b_ℓ already describes a localized beam, and $1/C_\ell$ increases non-exponentially, crafting a localized system $\widehat{\mathbf{A}}_h$ at each level is indeed possible.

Selecting the filters $f_{h,\ell}^H$, whose products form $\tilde{f}_\ell^h$ and $\mathbf{F}_h$ for each level, is a non-trivial matter. The main characteristic the filters must have is that each $\tilde{f}_\ell^h$ falls off quickly enough in real space to avoid strong couplings between the edge of the mask and the interior. Figure 6 shows what happens if this is not controlled correctly—the long-range couplings make the construction of an error smoother $\widehat{\mathbf{M}}$ impossible. In contrast, Figure 7 shows the behavior of the operators in the well-tuned case.

A filter that we found to work very well is given by squaring the exponent of a Gaussian,

$$q_\ell = \exp(-\ell^2(\ell+1)^2\lambda). \tag{24}$$

The scale parameter λ is simply chosen from the scale behavior that we want. In our test runs, we chose the constraints $q_{2570} = 0.1$ at the $N_{\mathrm{side}}^h = 1024$ level and $q_{1536} = 0.1$ at the $N_{\mathrm{side}}^h = 512$ level.

This filter has the following advantages over a simple Gaussian:

1. It decays much more quickly in ℓ, while in real space it decays almost as quickly in the tails as the Gaussian. This allows us to avoid increasing the bandlimit of the original system beyond $\ell_{\mathrm{max}} = 3000$.
2. The rapid decay with ℓ is also beneficial to counter the behavior of $1/C_\ell$. In the range $2000 < \ell < 3000$, $1/C_\ell$ follows a rather steep trajectory (between $\sim\ell^7$ and ℓ^8) which, when only countered by a Gaussian, causes some ringing and less locality.

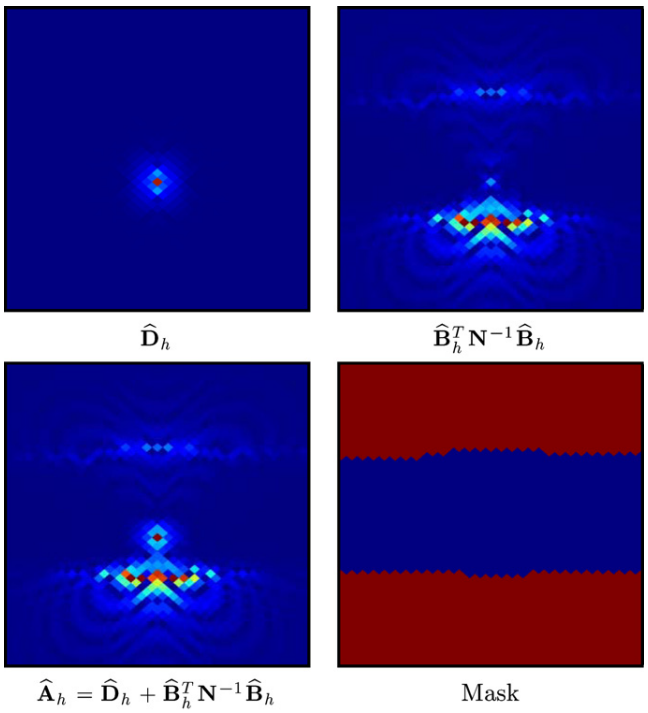


Figure 6. Effect of a poor choice of filter $\tilde{f}_\ell^h$. Each panel shows the couplings between a single pixel and its neighboring region, corresponding to a row/column of $\widehat{\mathbf{A}}_h$. In this case we used a low-pass filter based on modifying a standard needlet (Scodeller et al. 2011, and references therein). While the harmonic properties of this filter were very attractive, the tails do not decay quickly enough in real space. The resulting strong, long-range couplings are fatal to our algorithm.

(A color version of this figure is available in the online journal.)

3. Using Gaussian filters shapes the $\mathbf{N}^{-1}$ term so that couplings around a given pixel are similar to a Gaussian with FWHM of 4 pixels. That is, the couplings between neighboring pixels are rather strong. The filter defined above produces much weaker couplings between neighbors. This is not currently an advantage, because we let every pixel "see" a radius of $k = 8$ pixels around itself anyway in the error smoother. However, it could become an advantage in the future if k is chosen adaptively for each pixel.

Despite these features, the simple Gaussian filter behaved better at the coarser levels with very high signal-to-noise, as can be seen by comparing the second panel of Figure 7 with the first panel of Figure 8. In our tests we chose a Gaussian filter $f_{h,\ell}^H$ for levels $N_{\rm side}^h \leqslant 256$, tuned so that the cumulative filter $\tilde{f}_\ell^h$ on each level roughly corresponds to a Gaussian with FWHM of 2 pixels.

3.4. Band-limitation and Coarsening $\boldsymbol{Y}^T \boldsymbol{N}^{-1} \boldsymbol{Y}$

Figure 9 shows the effect of choosing the bandlimit $\ell_{\rm max}^h$ too low. On the coarser levels, ringing from the inverse-noise term causes strong non-local couplings unless the bandlimit is set as high as $6N_{\rm side}^h$. This limit depends on the signal-to-noise ratio, and $\ell_{\rm max} = 4N_{\rm side}^h$ is sufficient on the $N_{\rm side} = 512$ level.

The HEALPix grid can only represent a field accurately up to $\ell_{\rm max}^h \sim 2N_{\rm side}^h$, and will in fact see different scales on different parts of the sphere, due to the necessary irregularities in the pixelization. This is the primary reason for the non-traditional level traversal structure chosen in Section 3.2. The pixel projection operator $\mathbf{Y}_h$ removes some parts of the projected field that the grid cannot represent, but this is after all how a multi-level restriction normally works, and so poses no problems.

The filter $\tilde{f}_\ell^h$ allows us to set $\ell_{\rm max}^h$ much lower than the full $\ell_{\rm max}$. The two SHTs involved in $\mathbf{Y}_{\rm obs}^T \mathbf{N}^{-1} \mathbf{Y}_{\rm obs}$ still involve an $N_{\rm side} = 2048$ grid, however, so the coarsest levels are still almost as computationally expensive as the finest levels.

To work around this, the key is to note that the operator $\mathbf{Y}_{\rm obs}^T \mathbf{N}^{-1} \mathbf{Y}_{\rm obs}$ does not "see" scales in the inverse-noise map beyond $2\ell_{\rm max}^h$. This follows from an expansion into Wigner 3j-symbols (Hivon et al. 2002; Eriksen et al. 2004). Simply degrading the inverse-noise map to a coarser resolution HEALPix grid was found to be far too inaccurate, so more care is needed.

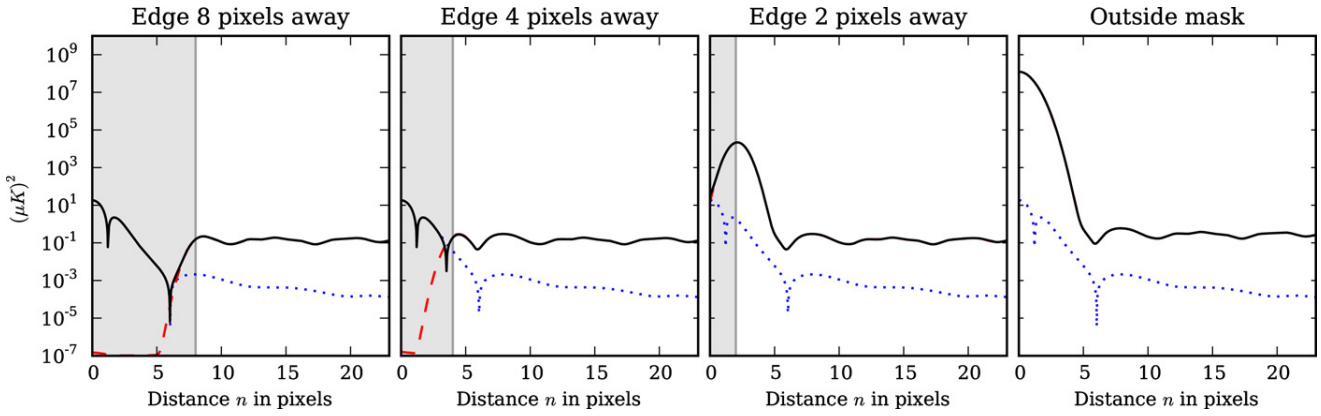


Figure 7. Effect of the mask on $\widehat{\mathbf{A}}_h$. Each panel shows the coupling strength in absolute value in the $\widehat{\mathbf{A}}_h$ operator, between a sample point at (θ, ϕ) (plotted at the origin), and another sample point n pixels away at $(\theta, \phi + n\Delta)$, where Δ is the angular size of one pixel. The couplings of $\widehat{\mathbf{A}}_h$ (black) are a sum of the prior term $\widehat{\mathbf{D}}_h$ (dotted blue) and the inverse-noise term $\widehat{\mathbf{B}}_h^T \mathbf{N}^{-1} \widehat{\mathbf{B}}_h$ (dashed red). For each panel, we vary the position of (θ, ϕ) relative to the mask (gray band), so that the origin is in each case a value on the diagonal of $\widehat{\mathbf{A}}_h$. Displayed here is our $N_{\rm side}^h = 32$ level in the case of 1.9 μK constant rms noise (the minimum rms level of the Planck 143 GHz band). The filter $\tilde{f}_\ell$ is a product of all the inter-level filters $f_{H,\ell}^h$ (as described in the text), but corresponds roughly to a Gaussian with FWHM of 2 pixels divided by the pixel window p_ℓ. The "floor" at 10^{-1} is caused by the non-Gaussian features of the instrumental beam, b_ℓ. For comparison, a perfect Gaussian instrumental beam is used in Figure 9.

(A color version of this figure is available in the online journal.)

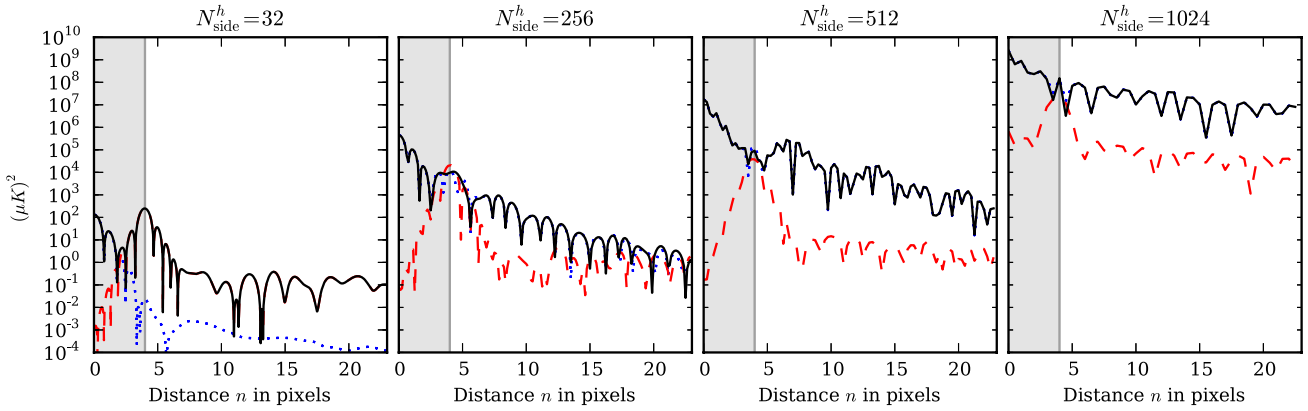


Figure 8. Effect of resolution on $\widehat{\mathbf{A}}_h$. See Figure 7 for legend and experimental setup. In this figure, we also show the effect of the filter q_ℓ of Equation (24), with λ appropriately tuned for the resolution in each case. As the resolution is increased, the signal-to-noise ratio decreases, making the influence of the edge of the mask less important.

(A color version of this figure is available in the online journal.)

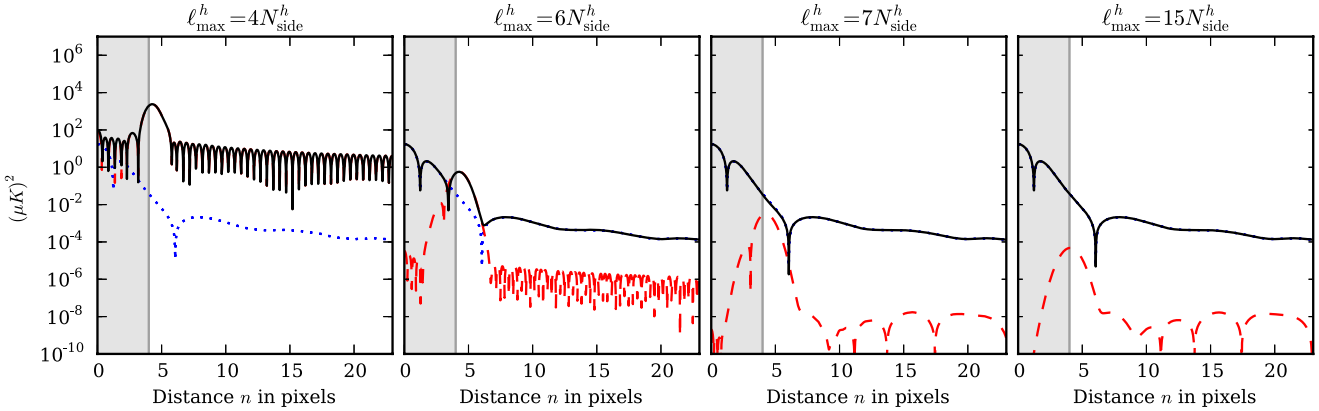


Figure 9. Effect of the band-limit $\ell^h_{\rm max}$ on $\widehat{\mathbf{A}}_h$. See Figure 7 for legend and experimental setup. The settings for each panel are the same except for varying $\ell^h_{\rm max}$. Here, the product $\tilde{f}^h_\ell b_\ell p_\ell$ is a pure Gaussian with FWHM of 2 pixels. Since the instrumental beam is in this case taken to be a perfect Gaussian, there is also no "floor" at 10^{-1} (compare with Figure 7 for the effect of a non-Gaussian beam).

(A color version of this figure is available in the online journal.)

First, we rewrite the operator as

$$\mathbf{Y}^T_{\rm obs}\mathbf{N}^{-1}\mathbf{Y}_{\rm obs} = \mathbf{Y}^T_{\rm obs}\mathbf{W}_{\rm obs}\left(\mathbf{W}^{-1}_{\rm obs}\mathbf{N}^{-1}\right)\mathbf{Y}_{\rm obs}, \qquad (25)$$

where $\mathbf{W}_{\rm obs}$ denotes the quadrature weights of the HEALPix $N_{\rm side} = 2048$ grid, so that $\mathbf{Y}^T_{\rm obs}\mathbf{W}_{\rm obs}$ corresponds to SH analysis, as described in Section 2.1. Then, we write ξ_i for the pixels on the diagonal of $\mathbf{W}^{-1}\mathbf{N}^{-1}$, and $\xi_{\ell m}$ for the same map expanded into SHs. Since the operator of Equation (25) does not see coefficients beyond $2\ell^h_{\rm max}$, we can truncate $\xi_{\ell m}$ and project it onto a Gauss–Legendre grid of the same order, which (unlike HEALPix grids) allows SH analysis that is accurate to almost machine precision. Using this re-weighted and downgraded inverse-noise map as the diagonal of a new inverse-noise matrix $\tilde{\mathbf{N}}^{-1}_h$, we have that

$$\mathbf{Y}^T_{\rm obs}\mathbf{N}^{-1}\mathbf{Y}_{\rm obs} = \tilde{\mathbf{Y}}^T_h\tilde{\mathbf{W}}_h\tilde{\mathbf{N}}^{-1}_h\tilde{\mathbf{Y}}_h, \qquad (26)$$

where $\tilde{\mathbf{Y}}_h$ and $\tilde{\mathbf{Y}}^T_h\tilde{\mathbf{W}}_h$ indicate SH synthesis and analysis on the Gauss–Legendre grid.

3.5. Error Smoother Construction for the CR System

A simple diagonal error smoother does not converge in our setup, primarily because pixels on the edge of the mask can have a very strong influence on the solution in the interior of the mask, as seen in Figure 7. Also, when applying Gaussian filters, the couplings between neighboring pixels are rather strong, preventing the use of a diagonal error smoother even far from the mask.

The basic strategy for our error smoother is to make sure that every pixel "sees" neighboring pixels in some radius k around it. In our case we let $k = 8$ on all levels, although improvements on this may be possible, especially in cases with lower signal-to-noise than ours.

We start by dividing the sphere into tiles of size k-by-k. Then, we include the couplings between pixels in the same and neighboring tiles while ignoring any couplings between pixels further apart, so that couplings are included in a radius of at least k pixels around every pixel. The result is a block sparse matrix, as shown in Figure 10.

Next, we explicitly compute the parts of $\widehat{\mathbf{D}}_h$ (Equation (21)) and $\widehat{\mathbf{B}}_h$ (Equation (22)) that fall within the sparsity pattern by evaluating the sum over Legendre polynomials from Equation (23). After preparing the block sparse matrix approximations, we use matrix multiplication without fill-in to compute $\widehat{\mathbf{B}}^T\mathbf{N}^{-1}\widehat{\mathbf{B}}$—that is, we neglect resulting blocks outside of the same sparsity pattern. The approximant for $\widehat{\mathbf{D}}_h$ can then be added directly. Finally, we perform a zero-fill-in incomplete Cholesky

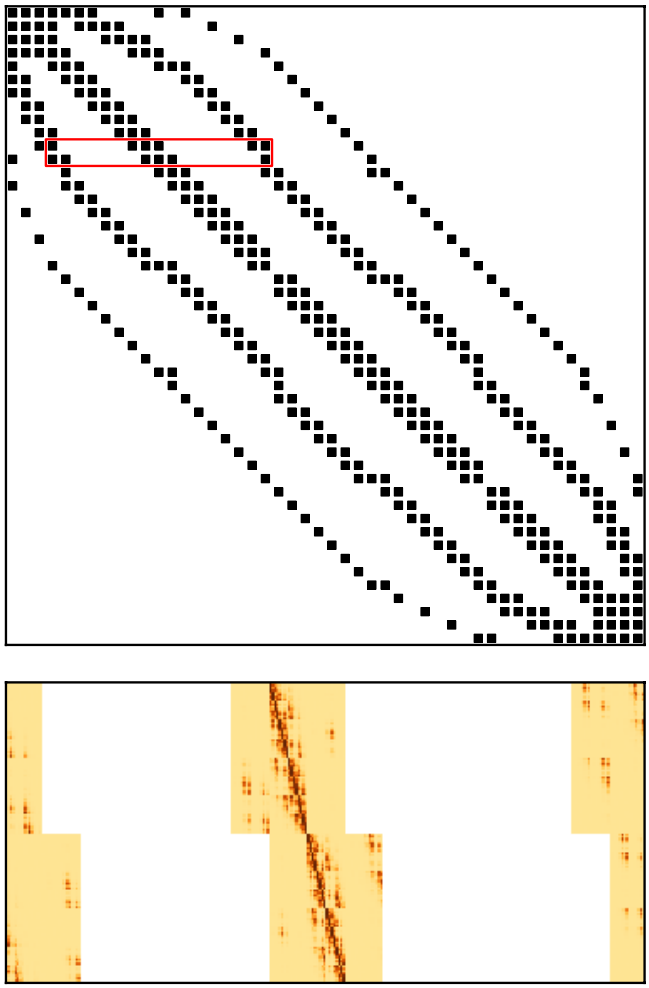


Figure 10. Structure of the block sparse matrices used in the error smoothers. Top panel: the sparsity pattern when every tile is coupled to its eight neighboring tiles. In this case, the pattern of tiles is an $N_{\rm side} = 2$ HEALPix grid in ring-ordering. Bottom panel: the blocks of $\widehat{\mathbf{B}}_h = \mathbf{Y}_{\rm obs}\mathbf{B}\mathbf{Y}_h^T$ corresponding to the red rectangle in the top panel. The blocks on the diagonal contain within-tile couplings, while off-diagonal blocks are couplings between pixels in neighboring tiles. Each block is rectangular because $\mathbf{Y}_{\rm obs}$ samples on a grid with $4\times$ more pixels than the grid sampled by $\mathbf{Y}_h$.

(A color version of this figure is available in the online journal.)

factorization (ICC), i.e., we perform in-place Cholesky factorization of the block sparse approximant as usual, but ignore any element updates outside of the sparsity pattern during the factorization process.

Without modification, the factorization process usually fails, either due to the sparse approximant of the full dense matrix ending up non-positive-definite, or because of elements dropped during the ICC. When this happens, we do a binary search for the lowest ridge adjustment α that, when added to the diagonal, makes the factorization procedure succeed, and scale this α by a factor of 1.5 for the final factorization. Typical ridge values α are in the range 10^{-2} to 10^{-4} times the maximum element of $\mathbf{A}_h$.

After factorization, applying the smoother is simply a matter of doing the usual triangular solve. This is an inherently sequential process, and the smoother therefore currently runs on a single CPU core. Since an error smoother only needs to work locally, we expect to be able to apply domain decomposition techniques, partitioning the sphere into large domains that overlap by k or $2k$ pixels, and applying one error smoother on each domain. Proper parallelization of the error smoother is left for future work, however. Also note that the process described above is the very simplest incomplete factorization algorithm, and more sophisticated incomplete factorization algorithms are standard in the literature.

In Section 4.1, we quote numbers for the execution time and memory usage of the smoother. One possibility for reducing memory consumption in the future is to let k be adaptive, as it can be made smaller away from the edges of the mask. All error smoother computations are done in single precision. In the current implementation, computing $\widehat{\mathbf{B}}$ is very expensive, as we sample it directly on the $N_{\rm side} = 2048$ grid. This is not a fundamental scaling problem, but rather an issue of implementation, as the degraded inverse-noise map on the Gauss–Legendre grid described in Section 3.4 could also be used in this setting.

4. IMPLEMENTATION AND RESULTS

4.1. Numerical Results and Performance

The basic assumptions for our experimental setup have already been laid out in Section 2.2. We choose for our example the rms map and symmetric beam approximation of the 143 GHz channel of Planck, as provided in the Planck 2013 data release (Planck Collaboration 2013a).

We tried running both with the 40%-sky, 80%-sky, and 97%-sky masks used in the Planck analysis, in all cases together with the 143 GHz point source mask. The mask has some impact on speed of convergence, but not enough to warrant attention, and we therefore only present the results from the 80%-sky mask, which was the *slowest* to converge.

For the power spectrum, C_ℓ, we use the standard best-fit Planck+WP+high-ℓ six-parameter ΛCDM spectrum (Planck Collaboration 2013d), but set C_0 and C_1 to the value of C_2 as a wide prior for any residual monopole or dipole component. Statistically, the prior for the monopole and dipole is of little relevance, since the data so strongly constrain these components. Note that the present algorithm will not let us condition on a given monopole and dipole (i.e., set $C_0 = C_1 = 0$), at least without modifications.

To produce the right-hand side, $\mathbf{b}$, corresponding to a random test realization, we draw a simulated $\mathbf{x}_{\rm true}$ from the prior $p(\mathbf{s}|C_\ell)$, and multiply it with $\mathbf{A}$ of Equation (4). This synthetic setup allows us to track the true error, $\mathbf{e} = \mathbf{x}_{\rm true} - \mathbf{x}$. In a real setting the right-hand side is of course generated from observed data, and in this case one can only track the residual, $\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x}$.

The error smoothers are least efficient on the largest scales. At the same time, these are much cheaper to process than the small-scale smoothers due to the $O(\ell_{\rm max}^3)$ scaling of the SHTs. We therefore choose a partial W-cycle, where the levels for $N_{\rm side}^h \leqslant 1024$ participate in a W-cycle ($n_{\rm rec}^h = 2$ in Figure 5), but the very expensive error smoother of the $N_{\rm side}^h = 1024$ level, as well as SHTs at $\ell_{\rm max} = 3000$, are only run once on the way down and once on the way up (a V-cycle).

In Figure 11 we plot the resulting convergence, in terms of absolute error as a function of W-cycle iteration count. Here we see that the error falls exponentially with cycle count, at the rate of roughly one order of magnitude per iteration. The largest error anywhere on the sky is smaller than 1 μK after only three W-cycles, and approaches the numerical precision limit after eight cycles.

Table 1
Structure and Computational Cost of a W-cycle

Level	$\ell_{\max}$	No. of Visits	Time in $\mathbf{Y}_h$ (Wall s)	Time in $\mathbf{Y}_{\rm obs}$ (Wall s)	Time in $\widehat{\mathbf{M}}_h$ or $\mathbf{A}_h^{-1}$ (Wall s)	Total Time (Wall s)
$\ell_{\max} = 3000$	3000	1	···	6×2.8	···	16.8
$N_{\rm side} = 1024$	3000	1	4×1.2	4×2.8	2×11	38.0
$N_{\rm side} = 512$	2048	2	8×0.3	10×1.3	4×2.3	24.6
$N_{\rm side} = 256$	1280	4	16×0.07	20×0.40	8×0.57	13.7
$N_{\rm side} = 128$	768	8	32×0.016	40×0.10	16×0.14	6.75
$N_{\rm side} = 64$	384	16	64×0.004	80×0.03	32×0.035	3.78
$N_{\rm side} = 32$	224	32	128×0.002	160×0.008	64×0.009	2.11
$\ell_{\max} = 40$	40	32	···	···	32×0.028	0.90
Other work						8
Full W-cycle						114

Notes. All times are given in wall time seconds using 16 CPU cores. The total number of operations of each kind for the W-cycle is indicated in each case; this number is not a multiple of the number of visits because the input vector $\mathbf{x}$ is zero on the first visit (except on the first level). Ignoring this aspect, each pixel level requires: (1) two-level-transfer spherical harmonic transforms ($\mathbf{Y}_h$), (2) three multiplications with $\mathbf{A}_h$, each with two inverse-noise spherical harmonic transforms ($\mathbf{Y}_{\rm obs}$), and (3) two applications of the error smoother $\widehat{\mathbf{M}}$. The top spherical harmonic level also requires two applications of $\mathbf{A}_h$, while the smoother application time is negligible. The bottom spherical harmonic level consists only of dense triangular solves.

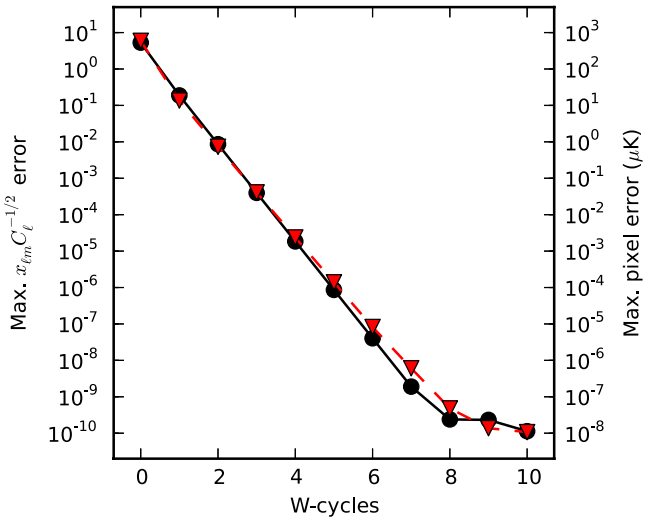


Figure 11. Absolute errors as a function of W-cycle count. For every iteration we plot the maximum error over all $C_\ell^{-1/2} x_{\ell m}$ (black circles, left axis), as well as the largest error across all pixels (red triangles, right axis).

(A color version of this figure is available in the online journal.)

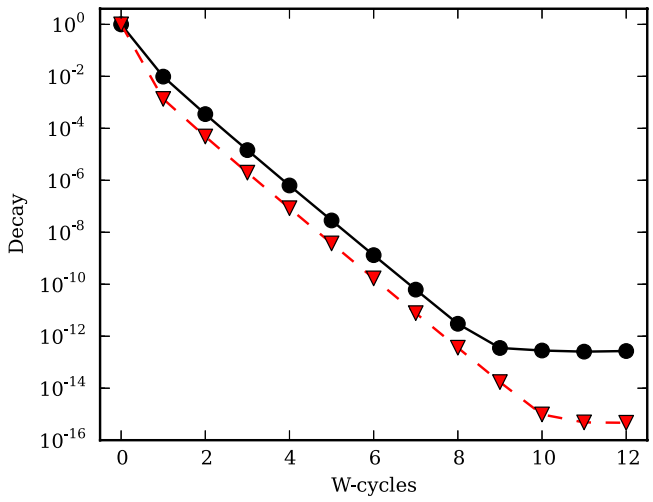


Figure 12. Comparison of absolute errors relative to C_ℓ, $(\mathbf{x}-\mathbf{x}_{\rm true})^T \mathbf{S}^{-1} (\mathbf{x}-\mathbf{x}_{\rm true})$ (black circles), and similarly scaled residuals, $(\mathbf{b} - \mathbf{A}\mathbf{x})^T \mathbf{S}^{-1} (\mathbf{b} - \mathbf{A}\mathbf{x})$ (red triangles). Both are normalized with respect to the initial error/residual. The two quantities behave very similarly, implying that the residual is an excellent proxy for the true error.

(A color version of this figure is available in the online journal.)

As mentioned above, since we know what the true solution is for the simulated data, we are also able to trace the absolute error, $\mathbf{e} = \mathbf{x}_{\rm true} - \mathbf{x}$, although only the residual, $\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x}$, is available in real-world applications. Figure 12 shows that these have qualitatively very similar behavior as a function of W-cycle count, which implies that the residual can be used as a robust proxy for the actual error for the multi-level algorithm. The same is not true for the CG method, for which the error can flatten earlier than the residual due to the presence of the nearly singular modes in $\mathbf{A}$.

Finally, in Figure 13 we show the relative error as a function of multipole moment and W-cycle count. This plot highlights the problematic angular scales, and is therefore particularly useful during the debugging and tuning phase of the analysis; for example, the use of a V-cycle rather than a W-cycle would make the large scales noticeably lag behind in convergence on this plot. Another example is that, if the filters $\tilde{f}_h$ are poorly tuned (potentially causing the method to diverge), the responsible level can often be picked out on this plot.

The total run-time for this setup was 114 s wall time per W-cycle on 16 CPU cores (AMD 6282 running at 2.6 GHz). Table 1 breaks this cost down further to the individual levels and actions. The bulk of the memory use is by the error smoothers, which consume about 20 GiB of memory (see Table 2). The total process footprint was around 30 GiB, although unnecessary temporary arrays abound in the current implementation.

Table 2 presents the cost of the necessary precomputations. For every new combination of instrumental beam, noise map and mask, or for a new choice of multi-level filters $f_{h,\ell}^H$, one needs to precompute an approximation to $\widehat{\mathbf{B}}_h^T \mathbf{N}^{-1} \widehat{\mathbf{B}}_h$ for every solver level. These precomputations required a total of 44 CPU hours in our tests, but are trivially parallel. We also expect that one will usually load the results from disk. The approximation

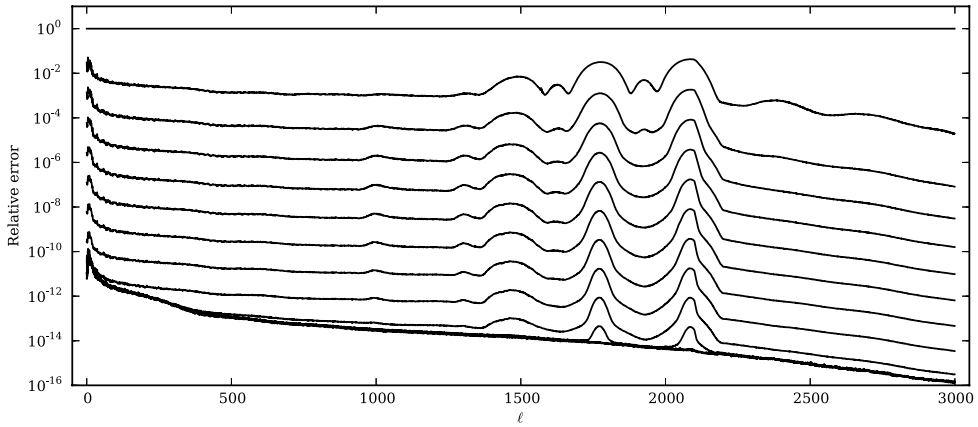


Figure 13. Relative error as a function of angular scale. Starting from the top, each line shows the error for a given multi-level W-cycle. Specifically, we plot $\|\mathbf{x}_{\mathrm{true},\ell} - \mathbf{x}_\ell\|/\|\mathbf{x}_{\mathrm{true},\ell}\|$, where $\mathbf{x}_\ell$ denotes a vector with the coefficients for a given ℓ only. This plot is especially useful during development and tuning of the code, as one can immediately see which error smoothers do not perform well.

Table 2
Error Smoother Precomputation Cost/Memory Use per Solver Level

N_{side}	Time Obs. (CPU minutes)	Time C_ℓ (CPU minutes)	Time ICC (CPU minutes)	Mem. Use (GiB)
1024	727	85	1.15	15
512	509	15	0.35	3.7
256	340	2.4	0.10	0.93
128	230	0.36	0.02	0.23
64	452	0.05	0.007	0.058
32	363	0.01	0.002	0.014
Total	2621	103	1.6	20

Notes. All times are given in CPU minutes (wall time times the number of CPU cores used). Precomputations can be divided into the part that must be performed whenever the observational setup (beam/mask/noise map) changes and the part that must be performed whenever the prior (C_ℓ) changes. If any part changes, the non-parallel incomplete Cholesky factorization (ICC) must also be performed again.

for $\widehat{\mathbf{D}}_h$ must be recomputed every time C_ℓ changes, which in the case of Gibbs sampling means every time one wants to run the solver. Fortunately, this computation is much cheaper and only requires around 100 CPU minutes of trivially parallel work, plus 2 minutes of non-parallel work. We argue in Section 5 that it should be possible to greatly decrease precomputation time in future.

The main weakness in the current implementation is the lack of parallelization in the error smoothers. Not only does the code need to be run on a single node, but the 40 s spent on error smoothing runs on a single CPU core, with the 15 other cores idling. Parallelization of the smoother would bring the wall time much closer to 80 s, as well as allowing the distribution of the 20 GiB of smoother data among several cluster nodes.

4.2. Notes on Implementation and Dependencies

The CR solver is part of Commander 2, which is made available as open source software under the BSD license (core code) and the GNU General Public License (GPL) (full software when including dependencies). For more information, see http://commander.bitbucket.org/.

Commander 2 is implemented in a mixture of Python (using NumPy and SciPy), Cython (Behnel et al. 2011), Fortran 90, and C. For SHTs we use `libsharp` (Reinecke & Seljebotn 2013). For our benchmarks we have used OpenBLAS (Goto & van de Geijn 2008; Xianyi et al. 2012) for linear algebra.

The main computation time is spent in `libsharp` or OpenBLAS, and as such is already highly optimized. The computation of Equation (23) benefited greatly from being structured as described in the appendix of Seljebotn (2012). In addition to what is mentioned there, we made use of the AVX and FMA4 instruction sets. Also, note that all the computations for the error smoother could be performed in single precision.

5. DISCUSSION

We have presented a new algorithm for solving the Gaussian CR system for high-resolution CMB data. This method is based on ideas from multi-grid (or multi-level) theory, and is fundamentally different from the CG methods traditionally used for this problem. Being only weakly dependent on the signal-to-noise ratio of the data set under consideration, our new method converges exponentially to numerical precision when properly tuned, and is capable of producing CRs for the full resolution of a Planck-like data set within minutes. For comparison, we have yet to achieve robust full-sky convergence with CG methods for the same data set. Indeed, this particular issue was the single most important obstacle preventing a full-resolution analysis of the Planck 2013 data release with the Commander code.

The ultimate goal of this line of work is to perform an exact global Bayesian analysis of the high-resolution, high-sensitivity observations now being produced by CMB experiments, including component separation as described by Eriksen et al. (2008b). For this to be successful, multi-frequency and multi-component analysis must be added to the algorithm. Other complications, such as the possible asymmetry of the CMB on large scales (e.g., Planck Collaboration 2013c), will also need to be taken into account. As such, the present paper represents only the first step toward a complete solution. We also emphasize that the algorithm as presented here is only the first implementation of a more general framework, and we expect that many improvements with respect to computational speed, application to more general cases, overall robustness and stability, and even user interfaces, will be introduced in the near future. Before concluding this paper, we will mention a few relevant ideas, but leave all details for future publications.

Firstly, as is evident from Figure 7, our method is quite sensitive to the behavior of the tails of the instrumental beams

extending as far out as the 10^{-5} level, as these formally constrain the solution inside the mask. These tails are not realistically known to such high accuracy, and so this issue is therefore a modeling problem as well as a numerical problem. In practice, it seems that in the absence of other options, one should just choose a form for the tails that falls quickly enough to not have an effect on the solution, and that allows a small computational bandlimit, $\ell_{\rm max}$. In short, optimally tuning the tails of the beam profile may render a more stable solution at a lower computational cost.

For an exact analysis of data from current and forthcoming CMB experiments, one would ideally like to account for the effect of asymmetric beams. With the above in mind, we envision two solutions for this. One option is to modify the algorithm so that the beams are defined in pixel space, as is done in FEBECop (Mitra et al. 2011) for instance, and then carry the FEBECop beams through to the computation of the smoother. The main challenge in this scenario is how to avoid very expensive matrix–vector multiplications at the coarse levels. Alternatively, and perhaps more simply, one could use the multi-level solver for perfect symmetric beams described here as a preconditioner for a CG search, which then accounts for the beam asymmetries in its own internal matrix multiplications.

Correlated noise is another significant complication for current CMB observations. While these correlations have a complicated morphology in pixel space, being convolved with the scanning strategy of the experiment, they are simple to describe in the time-domain. With the vastly improved convergence rate of the multi-level method presented here—requiring only a handful of iterations to reach sub-μK errors—it may for the first time be realistic to define the CR system in time-domain, rather than map-domain. As for asymmetric beams, this can either be done by defining the multi-level scheme directly in time-domain, or, if that does not succeed, by using the multi-level solver for uncorrelated noise as a preconditioner for a time-domain CG search. Going to time-domain also provides a direct route to handling beam asymmetries and optical sidelobes by full-sky convolution (Wandelt & Górski 2001).

The current computational bottleneck in our implementation is the time needed to precompute the error smoothers. The time is spent almost exclusively on sampling rotationally invariant operators at every position on the sphere by brute force evaluation of Equation (23). While the code for this computation is already highly optimized, as mentioned above, we do not exploit any symmetries from pixel to pixel. The grid used within the multi-level process is arbitrary, and not necessarily related to the grid of the inverse-noise map, **N**, or data vector, **d**. A future implementation of the algorithm will therefore employ a different grid with greater symmetry than the HEALPix grid, which will only require evaluation of the smoother blocks 3–7 times per pixel ring, thus reducing the computational scaling from $O(k^2 \ell_{\rm max} N_{\rm pix})$ to $O(k^2 \ell_{\rm max} \sqrt{N_{\rm pix}})$.

Finally, the error smoother evaluation is currently not parallelized, and only executes on a single CPU core. As the error smoothers only need to work well for the local couplings, we expect to be able to partition the sphere into multiple partially overlapping domains, and apply an error smoother on each domain in parallel, at the cost of some extra computation on the domain borders. Assuming that this approach is successful, the SHTs will once again become the bottleneck of the overall algorithm.

We thank Mikolaj Szydlarski and Martin Reinecke for useful discussions. D.S.S., H.K.E., and P.B. are supported by European Research Council grant StG2010-257080. K.A.M. is supported by the Research Council of Norway through a Centre of Excellence grant to the Centre for Biomedical Computing at Simula Research Laboratory.

REFERENCES

Axelsson, O., & Lindskog, G. 1986a, NuMat, 48, 479
Axelsson, O., & Lindskog, G. 1986b, NuMat, 48, 499
Behnel, S., Bradshaw, R., Citro, C., et al. 2011, CSE, 13, 2
Bennett, C. L., Larson, D., Weiland, J. L., et al. 2013, ApJS, 208, 20
Brandt, A. 2011, in Multiscale and Multiresolution Methods, ed. T. J. Barth et al. (Berlin: Springer), 1
Doré, O., Teyssier, R., Bouchet, F. R., Vibert, D., & Prunet, S. 2001, A&A, 374, 358
Elsner, F., & Wandelt, B. D. 2012, in Proc. Big Bang, Big Data, Big Computers (arXiv:1211.0585)
Elsner, F., & Wandelt, B. D. 2013, A&A, 549, A111
Eriksen, H. K., Dickinson, C., Jewell, J. B., et al. 2008a, ApJL, 672, L87
Eriksen, H. K., Huey, G., Banday, A. J., et al. 2007a, ApJL, 665, L1
Eriksen, H. K., Huey, G., Saha, R., et al. 2007b, ApJ, 656, 641
Eriksen, H. K., Jewell, J. B., Dickinson, C., et al. 2008b, ApJ, 676, 10
Eriksen, H. K., O'Dwyer, I. J., Jewell, J. B., et al. 2004, ApJS, 155, 227
Górski, K. M., Hivon, E., Banday, A. J., et al. 2005, ApJ, 622, 759
Goto, K., & van de Geijn, R. 2008, ACM Trans. Math. Softw., 34, 3
Grigori, L., Stompor, R., & Szydlarski, M. 2012, in Proc. of the Int. Conf. on High Performance Computing, Networking, Storage and Analysis, ed. J. K. Hollingsworth (Los Alamitos, CA: IEEE Computer Society Press), 91
Hackbush, W. 1985, Multi-grid Methods and Applications (Berlin: Springer)
Havé, P., Masson, R., Nataf, F., et al. 2013, SIAM J. Sci. Comput., 35, 3
Hivon, E., Górski, K. M., Netterfield, C. B., et al. 2002, ApJ, 567, 2
Jewell, J., Levin, S., & Anderson, C. H. 2004, ApJ, 609, 1
Mitra, S., Rocha, G., Górski, K. M., et al. 2011, ApJS, 193, 5
O'Dwyer, I. J., Eriksen, H. K., Wandelt, B. D., et al. 2004, ApJL, 617, L99
Planck Collaboration , Ade, P. A. R., & Aghanim, N. 2013a, A&A, submitted (arXiv:1303.5062)
Planck Collaboration , Ade, P. A. R., & Aghanim, N. 2013b, A&A, submitted (arXiv:1303.5075)
Planck Collaboration , Ade, P. A. R., & Aghanim, N. 2013c, A&A, submitted (arXiv:1303.5083)
Planck Collaboration , Ade, P. A. R., & Aghanim, N. 2013d, A&A, submitted (arXiv:1303.5076)
Reinecke, M. 2011, A&A, 526, A108
Reinecke, M., & Seljebotn, D. S. 2013, A&A, 554, A112
Scodeller, S., Rudjord, Ø., Hansen, F. K., et al. 2011, ApJ, 733, 121
Seljebotn, D. S. 2012, ApJS, 199, 5
Shewchuk, J. R. 1994, http://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.ps
Smith, K. M., Zahn, O., & Doré, O. 2007, PhRvD, 76, 043510
Tang, J. M., Nabben, R., Vuik, C., & Erlangga, Y. A. 2009, JSCom, 39, 3
Wandelt, B. D., & Górski, K. M. 2001, PhRvD, 63, 123002
Wandelt, B. D., Larson, D. L., & Lakshminarayanan, A. 2004, PhRvD, 70, 083511
Xianyi, Z., Qian, W., & Yunquan, Z. 2012, in IEEE 18th International Conference on Parallel and Distributed Systems (ICPADS), Model-driven Level 3 BLAS Performance Optimization on Loongson 3A Processor, ed. W. Cai, R. Siow Mong Goh, & M. Snir (Los Alamitos, CA: IEEE Computer Society Press), 684

Paper II (submitted)

Multi-resolution Bayesian CMB component separation through Wiener-filtering with a pseudo-inverse pre-conditioner

The algorithms developed in this paper allows for full-resolution, multi-resolution Bayesian analysis of CMB data. It not only solves all of the problems we had with generalizing the algorithm in Paper I to component seperation; it is also more elegant and simpler to implement.

Note: This paper has been submitted to A&A, but, as of this writing, not been peer-reviewed.

Paper III

Wavemoth – Faster spherical harmonic transforms by butterfly matrix compression

This was the first paper in my PhD, and a lot of fun. Mark Tygert had presented a novel algorithm for SHTs which he conjectured had a scaling of $O(L^2 \log L)$. He had only implemented it in MATLAB, and since it makes use of precomputed data, and moving precomputed data around can be slow, a production quality implementation was required to check the constant prefactor involved. I set out to make this implementation.

It seems strange that one can run a compression algorithm on a matrix to produce new smaller matrices – then run the compression algorithm again – and again – and compress the data more every time. It is even stranger that one can then proceed to use the compressed data directly, without any decompression. Yet, this is exactly what is going on here.

Tygert's conjecture was wrong – the scaling was $O(L^2(\log L)^2)$, with small gains in practice at the resolutions in use in CMB analysis. The lasting contribution of this work is in fact how fast the code was for the *brute-force* transforms, as it exploited the pipelining capabilities of modern CPUs. This heritage lives on in `libsharp` (Paper IV), which is in use by thousands of researchers every day.

The Astrophysical Journal Supplement Series, 199:5 (12pp), 2012 March doi:10.1088/0067-0049/199/1/5

WAVEMOTH–FAST SPHERICAL HARMONIC TRANSFORMS BY BUTTERFLY MATRIX COMPRESSION

D. S. Seljebotn
Institute of Theoretical Astrophysics, University of Oslo, P.O. Box 1029 Blindern, N-0315 Oslo, Norway; d.s.seljebotn@astro.uio.no
Received 2011 October 24; accepted 2011 December 16; published 2012 February 15

ABSTRACT

We present *Wavemoth*, an experimental open source code for computing scalar spherical harmonic transforms (SHTs). Such transforms are ubiquitous in astronomical data analysis. Our code performs substantially better than existing publicly available codes owing to improvements on two fronts. First, the computational core is made more efficient by using small amounts of pre-computed data, as well as paying attention to CPU instruction pipelining and cache usage. Second, Wavemoth makes use of a fast and numerically stable algorithm based on compressing a set of linear operators in a pre-computation step. The resulting SHT scales as $O(L^2 \log^2 L)$ for the resolution range of practical interest, where L denotes the spherical harmonic truncation degree. For low- and medium-range resolutions, Wavemoth tends to be twice as fast as *libpsht*, which is the current state-of-the-art implementation for the HEALPix grid. At the resolution of the Planck experiment, $L \sim 4000$, Wavemoth is between three and six times faster than libpsht, depending on the computer architecture and the required precision. Because of the experimental nature of the project, only spherical harmonic synthesis is currently supported, although adding support for spherical harmonic analysis should be trivial.

Key word: methods: numerical

Online-only material: color figures

1. BACKGROUND

The spherical harmonic transform (SHT) is the spherical analog of the Fourier transform and is an essential tool for data analysis and simulation on the sphere. A scalar field $f(\theta, \phi)$ on the unit sphere can be expressed as a weighted sum of the spherical harmonic basis functions $Y_{\ell m}(\theta, \phi)$,

$$f(\theta, \phi) = \sum_{\ell=0}^{\infty} \sum_{m=-\ell}^{\ell} a_{\ell m} Y_{\ell m}(\theta, \phi). \quad (1)$$

The coefficients $a_{\ell m}$ contain the spectral information of the field, with higher ℓ corresponding to higher frequencies. In calculations the spherical harmonic expansion is truncated for $\ell > L$, and the spherical field represented by $O(L^2)$ grid samples. Computing the sum above is known as the *backward SHT* or *synthesis*, while the inverse problem of finding the spherical harmonic coefficients $a_{\ell m}$ given the field f is known as the *forward SHT* or *analysis*.

In order to compute an SHT, the first step is nearly always to employ a separation of sums, which we review in Section 2.3, to decrease the cost from $O(L^4)$ to $O(L^3)$. We will refer to codes that take no measures beyond this to reduce complexity as brute-force codes. Of these, HEALPix (Górski et al. 2005) is one very widely used package, in particular, among cosmic microwave background (CMB) researchers. Recently, the libpsht package (Reinecke 2011) halved the computation time with respect to the original HEALPix implementation, simply through code optimizations. As of version 2.20, HEALPix uses libpsht as the back end for SHTs. Other packages using the brute-force algorithm include S^{2}HAT (Hupca et al. 2010; Szydlarski et al. 2011), focusing on cluster parallelization and implementations on the GPU, as well as GLESP (Doroshkevich et al. 2005) and ssht (McEwen & Wiaux 2011), focusing on spherical grids with more accurate spherical harmonic analysis than what can be achieved on the HEALPix grid.

The discovery of fast Fourier transforms (FFTs) has been all important for signal analysis over the past half century, and there is no lack of high-quality commercial and open source libraries to perform FFTs with stunning speed. Unfortunately, the straightforward divide-and-conquer FFT algorithms do not generalize to SHTs, and research in fast SHT algorithms has yet to reach maturity in the sense of widely adopted algorithms and libraries.

The libftsh library (Mohlenkamp 1999) uses local trigonometric expansions to compress the spherical harmonic linear operator, resulting in a computational scaling of $O(L^{5/2} \log L)$ in finite precision arithmetic. SpharmonicKit (Healy et al. 2003) implements a divide-and-conquer scheme that scales as $O(L^2 \log^2 L)$. We comment further on these in Section 4.4. Other algorithms have also been presented but either suffer from problems with numerical stability, are impractical for current resolutions, or simply lack publicly available implementations (e.g., Suda & Takami 2002; Kunis & Potts 2003; Rokhlin & Tygert 2006; Tygert 2008, 2010).

We present Wavemoth,[1] an experimental open source implementation of the algorithm of Tygert (2010). This algorithm has several appealing features. First, it is simple to implement and optimize. Second, it is inherently numerically stable. Third, its constant pre-factor is reasonable, yielding substantial gains already at $L \sim 2000$. The accuracy of the algorithm is finite but can be arbitrarily chosen. For any given accuracy, the computational scaling is $O(L^2 \log^2 L)$, but lowering the requested accuracy makes the constant pre-factor smaller.

We stress that our work consists solely in providing an optimized implementation. While we review the basics of the algorithm in Section 3, Tygert (2010) should be consulted for details and proofs. We have focused in particular on the HEALPix grid and use libpsht as our baseline for comparisons.

[1] http://github.com/wavemoth; commit 59ec31b8 was used to produce the results of this paper.

However, all methods work equally well for any other grid with isolatitude rings.

Section 2 reviews SHTs in more detail, as well as the computational methods that are widely known and used across all popular codes. Section 3 reviews the algorithm of Tygert (2010) and how we have adapted it to our purposes. Section 4 focuses on the high-level aspects of software development and provides benchmarks, while the Appendix provides the low-level implementation details.

2. BASELINE ALGORITHMS

2.1. The Spherical Harmonic Basis Functions

We use the convention that points on the sphere are parameterized by a colatitude $\theta \in [0, \pi]$, where 0 corresponds to the "north pole," and a longitude $\phi \in [0, 2\pi)$. The spherical harmonic basis functions $Y_{\ell m}(\theta, \phi)$ can then be expressed in terms of the *associated Legendre functions* $P_\ell^m(z)$. Assuming $m \geqslant 0$, we have

$$\begin{aligned} Y_{\ell m}(\theta, \phi) &= \sqrt{\frac{2\ell+1}{4\pi}\frac{(\ell-m)!}{(\ell+m)!}} P_\ell^m(\cos\theta) e^{im\phi} \\ &\equiv \widetilde{P}_\ell^m(\cos\theta) e^{im\phi}, \end{aligned} \tag{2}$$

where we define the *normalized associated Legendre function* $\widetilde{P}_\ell^m$. Our definition follows that of Press et al. (2007); the normalization differs by a factor of $\sqrt{1/2}$ from the one in Tygert (2010).

Note that while the spherical harmonics $Y_{\ell m}$ and the coefficients $a_{\ell m}$ are complex, $\widetilde{P}_\ell^m$ is real for the argument range of interest. For negative m, the symmetry $Y_{\ell,-m} = (-1)^m Y_{\ell m}^*$ can be used, although this is only needed for complex fields. Wavemoth only supports real fields, which have spherical harmonic expansions obeying $a_{\ell m} = (-1)^m a_{\ell -m}^*$.

2.2. Discretization and the Forward Transform

For computational work one has to assume that one is working with a band-limited signal, so that $a_{\ell m} = 0$ when $\ell > L$. The SHT synthesis is then given simply by evaluating Equation (1) in a set of points on the sphere.

The opposite problem of computing $a_{\ell m}$ given $f(\theta_j, \phi_j)$, namely, spherical harmonic analysis, is less straightforward. In the limit of infinite resolution, we have

$$a_{\ell m} = \int f(\theta, \phi) Y_{\ell m}^*(\theta, \phi) d\Omega, \tag{3}$$

where $d\Omega$ indicates integration over the sphere. This follows easily from the orthogonality property,

$$\int Y_{\ell m} Y_{\ell' m'}^* d\Omega = \delta_{\ell\ell'}\delta_{mm'}. \tag{4}$$

There is no canonical way of choosing sample points on the sphere. The simplest grid conceptually is the equiangular grid. Doroshkevich et al. (2005) and McEwen & Wiaux (2011) describe grids that carry the orthogonality property of the continuous spherical harmonics over to the discretized operator. In contrast, the HEALPix grid (Górski et al. 2005) trades orthogonality for the property that each pixel has the same area, which is convenient for many operations in the pixel basis.

Independent of what grid is chosen, a natural approach to spherical harmonic analysis is to use a quadrature rule with some weights w_j, so that

$$a_{\ell m} = \sum_{j=1}^{N_{\mathrm{pix}}} w_j f(\theta_j, \phi_j) Y_{\ell m}^*(\theta_j, \phi_j). \tag{5}$$

On the HEALPix grid the numerical accuracy of this approach is limited, but it is still the most common procedure.

Some real-world signal analysis problems do not need the forward transform at all. In the presence of measurement noise in the pixel basis, one can argue that the best approach is not to pull the noise part of the signal into spherical harmonic basis at all. For instance, consider the archetypical CMB data model,

$$\mathbf{d} = \mathbf{Y}\mathbf{s} + \mathbf{n}, \tag{6}$$

where $\mathbf{d}$ represents a vector of pixels on the sky with observed data (not necessarily the full sky), $\mathbf{s}$ represents our signal of interest in spherical harmonic basis, and $\mathbf{n}$ represents instrumental noise in each pixel. Spherical harmonic synthesis is denoted $\mathbf{Y}$; note that Equation (1) describes a linear operator and can be written as $\mathbf{f} = \mathbf{Y}\mathbf{a}$.

If we now assume that $\mathbf{s}$ and $\mathbf{n}$ are Gaussian random vectors with vanishing mean and known covariance matrices $\mathbf{S}$ and $\mathbf{N}$, respectively, then the maximum likelihood estimate of the signal is given by

$$\hat{\mathbf{s}} = (\mathbf{S}^{-1} + \mathbf{Y}^\dagger \mathbf{N}^{-1} \mathbf{Y})^{-1} \mathbf{Y}^\dagger \mathbf{N}^{-1} \mathbf{d}, \tag{7}$$

with $\hat{\mathbf{s}}$ in spherical harmonic basis. This system can be solved with reasonable efficiency by iterative methods. Note that we are here only concerned with the effect of $\mathbf{Y}$ as a non-invertible projection, and that no spherical harmonic analysis is ever performed, only the adjoint synthesis. Thus, neither the non-orthogonality caused by the HEALPix grid nor masking out large parts of the sky is a concern. See Eriksen et al. (2008) and references therein for more details on this technique in the context of CMB analysis.

2.3. Applying the Fast Fourier Transform

The first step in speeding up the SHT beyond the $O(L^4)$ brute-force sum is a simple separation of sums. For this to work well, pixels must be arranged on a set of isolatitude rings, with equidistant pixels within each ring. All grids in use for high-resolution data have this property.

We show the case for the SHT synthesis; analysis can be treated in the same way. Starting from Equation (1), we have, for pixel j within ring k, and with $z_k \equiv \cos\theta_k$,

$$\begin{aligned} f(\theta_k, \phi_{k,j}) &= \sum_{m=-L}^{L} \left[\sum_{\ell=|m|}^{L} a_{\ell m} \widetilde{P}_\ell^m(z_k) \right] e^{im\phi_{k,j}} \\ &\equiv \sum_{m=-L}^{L} q_m(z_k) e^{im\phi_{k,j}}, \end{aligned} \tag{8}$$

where we introduce $q_m(z_k)$. Assuming that ring k contains J_k pixels, their equidistant longitude is given by

$$\phi_{k,j} = \phi_{k,0} + \frac{2\pi j}{J_k}. \tag{9}$$

Since e^{ix} has period 2π, and since $q_m(z_k) = 0$ whenever $|m| > L$, we find that

$$\sum_{m=-L}^{L} q_{k,m} e^{im\phi_{k,j}} = \sum_{j=0}^{J_k-1} \tau_j(z_k) e^{2\pi ji/J_k} \tag{10}$$

with

$$\tau_j(z_k) = \sum_{t=-\infty}^{\infty} q_{J_k t+j}(z_k) e^{i\phi_{k,0}(J_k t+j)}. \tag{11}$$

Thus, one can phase-shift the coefficients $q_m(z_k)$ to match the ring grid, wrap around or pad with zeros, and perform a regular backward FFT. The symmetries of the spherical harmonic coefficients of a real field carry over directly to the Hermitian property of real Fourier transforms.

This separation of sums represents a first step in speeding up the SHT and is implemented in all packages for high-resolution SHTs.

2.4. Legendre Transforms and Even/Odd Symmetry

The function $q_m(z)$ introduced in Equation (8) is known as the *(Associated) Legendre transform of order m*,

$$q_m(z_k) = \sum_{\ell=m}^{L} \widetilde{P}_\ell^m(z_k) a_{\ell m}, \tag{12}$$

assuming $m \geqslant 0$. The following symmetry cuts the arithmetic operations required in an SHT in half, as long as the spherical grid distributes the rings symmetrically around the equator. For any non-negative integer n, the functions $\widetilde{P}^m_{m+2n}(z)$ are even and $\widetilde{P}^m_{m+2n+1}(z)$ are odd. We define q_m^{even} and q_m^{odd} so that q_m^{even} contains the even-numbered and q_m^{odd} the odd-numbered terms of Equation (12), and so that

$$q_m(z) = q_m^{\text{even}}(z) + q_m^{\text{odd}}(z). \tag{13}$$

Then, since q_m^{even} and q_m^{odd} are weighted sums of even and odd functions, respectively, they are themselves even and odd, so that $q_m(-z)$ can be computed at the same time essentially for free,

$$q_m(-z) = q_m^{\text{even}}(z) - q_m^{\text{odd}}(z). \tag{14}$$

For spherical harmonic analysis, one uses the orthogonality property. Assuming $m \geqslant 0$,

$$\int \widetilde{P}_\ell^m(z) \widetilde{P}_{\ell'}^m(z) dz = \delta_{\ell\ell'}, \tag{15}$$

so that

$$a_{\ell m} = \int \widetilde{P}_\ell^m(z) q_m(z) dz. \tag{16}$$

As discussed in Section 2.2, the resulting quadrature used in calculations can be exact or approximate, depending on the placement of the pixel rings. One can also in this case cut computation time in half by treating even and odd $\ell - m$ separately.

3. FAST LEGENDRE TRANSFORMS

As the Fourier transform part is essentially a solved problem, efforts to accelerate SHTs revolve around speeding up the Legendre transforms. Let us write Equation (12) as

$$\mathbf{q} = \mathbf{\Lambda}^T \mathbf{a}, \tag{17}$$

where we leave m and the odd versus even case implicit. For a full SHT, such a product must be computed for each of $2(L+1)$ different $\mathbf{\Lambda}$ matrices. The backward Legendre transform required for spherical harmonic analysis is similarly

$$\mathbf{a} = \mathbf{\Lambda} \mathbf{q}, \tag{18}$$

give or take a set of quadrature weights.

The idea of fast Legendre transform algorithms is to compute Equations (17) and (18) faster than $O(LN_{\text{ring}})$. The approach of Tygert (2010) is to factor $\mathbf{\Lambda}$ as a product of block-diagonal matrices in a pre-computation step, which can significantly reduce the number of elements in total. This technique is known as *butterfly compression* and was introduced by Michielssen & Boag (1996). The accuracy of the compression is tunable, but even nearly loss-less compression with close to double-precision accuracy is able to yield significant gains as the resolution increases. We review the algorithm below but stress again that the reader should consult Tygert (2010) for the full details.

3.1. The Interpolative Decomposition

The core building block of the compression algorithm is the Interpolative Decomposition (ID), described in Cheng et al. (2005). Assume that an $m \times n$ matrix $\mathbf{A}$ has rank k; then, the ID is

$$\mathbf{A} = \mathbf{A}^{(k)} \widetilde{\mathbf{A}}. \tag{19}$$

The matrix $\mathbf{A}^{(k)}$, known as the *skeleton matrix*, consists of k columns of $\mathbf{A}$, whereas $\widetilde{\mathbf{A}}$, the *interpolation matrix*, interpolates the eliminated columns from the ones that are preserved. Of course, k of the columns of $\widetilde{\mathbf{A}}$ must form the identity matrix.

The ID is obviously not unique; the trick is to find a decomposition that is numerically stable. The algorithm of Cheng et al. (2005) finds an interpolation matrix $\widetilde{\mathbf{A}}$ so that no element has absolute value greater than 2, all singular values are larger than or equal to 1, and the spectral norm is bounded by $\sqrt{4k(n-k)+1}$. The numerical precision of the decomposition is tunable, as the decomposition found by the algorithm satisfies

$$\|\mathbf{A} - \mathbf{A}^{(k)} \widetilde{\mathbf{A}}\| \leqslant \sqrt{4k(n-k)+1}\, \sigma_{k+1}, \tag{20}$$

where σ_{k+1} is the $(k+1)$ greatest singular value of $\mathbf{A}$. Implementing lossy compression is simply a matter of reducing the accuracy required of the IDs we use.

3.2. Butterfly Matrix Compression

We now use the ID recursively to factor the matrix $\mathbf{\Lambda}$. After applying p levels of compression, we have

$$\mathbf{\Lambda} = \mathbf{R}\mathbf{S}_p\mathbf{P}_{p-1}\mathbf{S}_{p-1}\cdots\mathbf{P}_2\mathbf{S}_2\mathbf{P}_1\mathbf{S}_1, \tag{21}$$

where $\mathbf{R}$ is a block-diagonal *residual matrix* containing elements that were not compressed, the $\mathbf{S}_i$ are block-diagonal matrices containing compressed data, and the $\mathbf{P}_i$ are permutation matrices. See Figure 1 for an illustration. The structures of the permutations are very similar to the *butterflies* used in FFT algorithms, hence the name of the compression scheme. In fact, if one lets $\mathbf{S}_i$ contain a specific set of 2×2 blocks on their diagonals, one recovers the famous Cooley–Tukey FFT. In our case the blocks will be significantly larger, typically around 150×150, although with much variation.

We start by partitioning $\mathbf{\Lambda}$ into 2^p column blocks. The number of levels p is mainly determined by the number of columns in

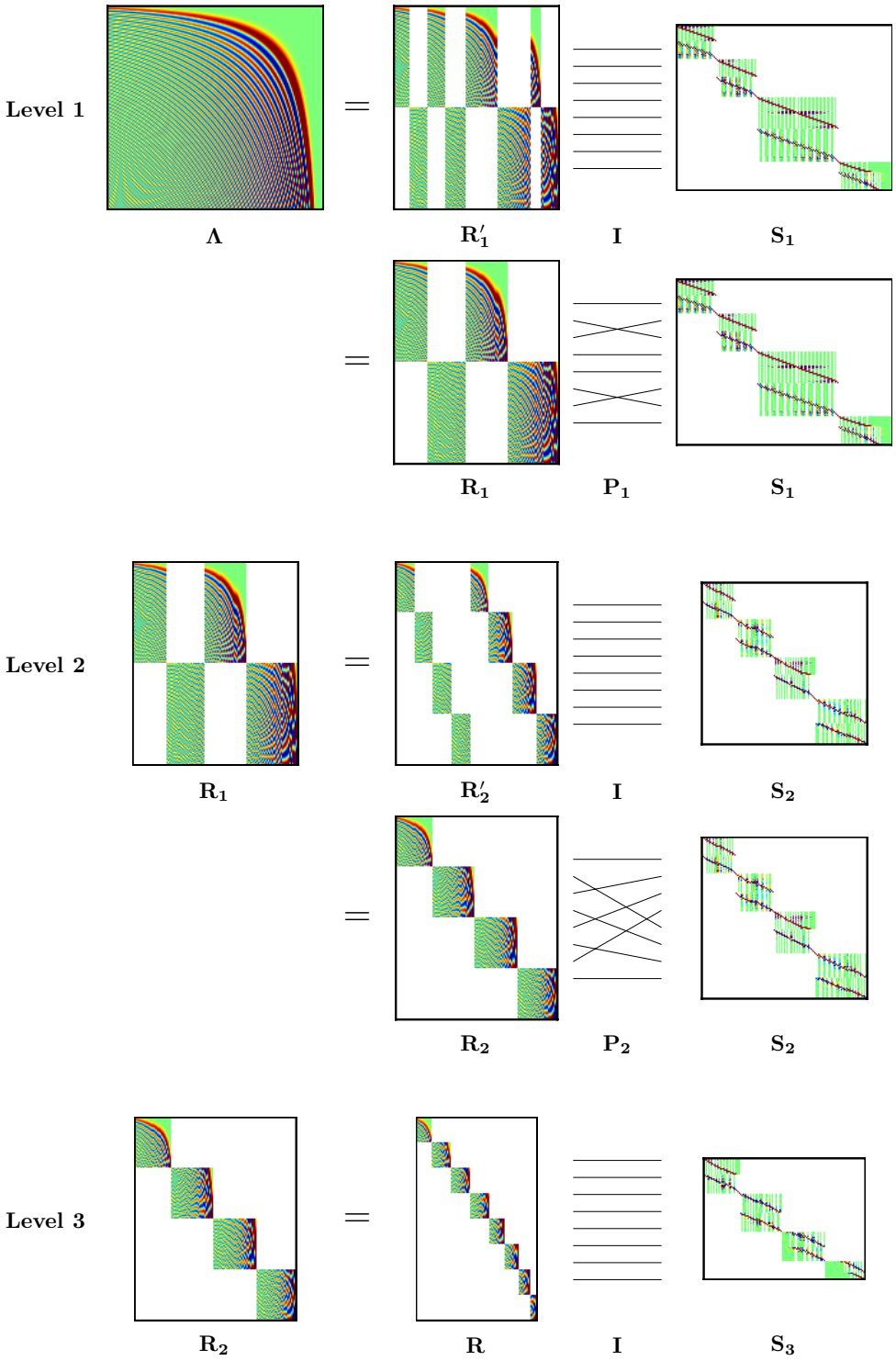


Figure 1. Illustration of the butterfly matrix compression scheme. On the first level, we use the Interpolative Decomposition to compress sub-blocks of the matrix $\mathbf{\Lambda}$ and produce the factorization $\mathbf{\Lambda} = \mathbf{R}'_1\mathbf{S}_1$, where all blocks in $\mathbf{R}'_1$ have full rank. We then proceed by permuting the columns of $\mathbf{R}'_1$ so that $\mathbf{\Lambda} = \mathbf{R}_1\mathbf{P}_1\mathbf{S}_1$, in order to create new rank-deficient blocks. The contents of the $\mathbf{S}_1$ matrix are saved as pre-computed data, while we carry $\mathbf{R}_1$ along for further compression on the next level. The algorithm continues in this fashion until the residual matrix $\mathbf{R}$ only consists of a single diagonal of full-rank blocks. The final factorization becomes $\mathbf{\Lambda} = \mathbf{R}\mathbf{S}_3\mathbf{P}_2\mathbf{S}_2\mathbf{P}_1\mathbf{S}_1$. The permutations involved are known in the FFT literature as *butterfly permutations*; the "butterfly" can be seen twice in the pattern of $\mathbf{P}_1$.

(A color version of this figure is available in the online journal.)

the matrix, so that the column blocks all are roughly of the same predetermined width. In our case, 64 columns worked well.

We then split each block roughly in half horizontally and compress each resulting block using the ID,

$$\mathbf{\Lambda} = \begin{bmatrix} \mathbf{T}_{1,1} & \mathbf{T}_{1,2} & \dots \\ \mathbf{B}_{1,1} & \mathbf{T}_{1,2} & \dots \end{bmatrix} = \begin{bmatrix} \left(\mathbf{T}^{(k)}_{1,1} \cdot \widetilde{\mathbf{T}}_{1,1}\right) & \left(\mathbf{T}^{(k)}_{1,2} \cdot \widetilde{\mathbf{T}}_{1,2}\right) & \dots \\ \left(\mathbf{B}^{(k)}_{1,1} \cdot \widetilde{\mathbf{B}}_{1,1}\right) & \left(\mathbf{B}^{(k)}_{1,2} \cdot \widetilde{\mathbf{B}}_{1,2}\right) & \dots \end{bmatrix},$$

where the first subscript of each matrix refers to this being the first iteration of the algorithm. It is useful to write the above matrix as

$$\mathbf{\Lambda} = \begin{bmatrix} \mathbf{T}^{(k)}_{1,1} & & \mathbf{T}^{(k)}_{1,2} & & \dots \\ & \mathbf{B}^{(k)}_{1,1} & & \mathbf{B}^{(k)}_{1,2} & \dots \end{bmatrix} \begin{bmatrix} \widetilde{\mathbf{T}}_{1,1} & & & \\ \widetilde{\mathbf{B}}_{1,1} & & & \\ & \widetilde{\mathbf{T}}_{1,2} & & \\ & \widetilde{\mathbf{B}}_{1,2} & & \\ & & & \ddots \end{bmatrix}.$$

We denote the right matrix $\mathbf{S}_1$. It cannot be further processed and its blocks are simply saved as pre-computed data, making use of the fact that each block embeds the identity matrix in a subset of its columns.

The left matrix can be permuted and further compressed. For some permutation matrix $\mathbf{P}_1$ we have

$$\begin{aligned} \mathbf{\Lambda} &= \begin{bmatrix} \mathbf{T}^{(k)}_{1,1} & & \mathbf{T}^{(k)}_{1,2} & & \dots \\ & \mathbf{B}^{(k)}_{1,1} & & \mathbf{B}^{(k)}_{1,2} & \dots \end{bmatrix} \mathbf{S}_1 \\ &= \begin{bmatrix} \mathbf{T}^{(k)}_{1,1} & \mathbf{T}^{(k)}_{1,2} & & & \dots \\ & & \mathbf{B}^{(k)}_{1,1} & \mathbf{B}^{(k)}_{1,2} & \dots \end{bmatrix} \mathbf{P}_1 \mathbf{S}_1. \end{aligned}$$

Then we join blocks horizontally, split them vertically, and compress each resulting block. For the top-left corner we have

$$\begin{bmatrix} \mathbf{T}^{(k)}_{1,1} & \mathbf{T}^{(k)}_{1,2} \end{bmatrix} = \begin{bmatrix} \mathbf{T}_{2,1} \\ \mathbf{B}_{2,1} \end{bmatrix} = \begin{bmatrix} \left(\mathbf{T}^{(k)}_{2,1} \cdot \widetilde{\mathbf{T}}_{2,1}\right) \\ \left(\mathbf{B}^{(k)}_{2,1} \cdot \widetilde{\mathbf{B}}_{2,1}\right) \end{bmatrix}. \quad (22)$$

Applying this to all blocks in the matrix, we get

$$\begin{aligned} \mathbf{\Lambda} &= \begin{bmatrix} \left(\mathbf{T}^{(k)}_{2,1} \cdot \widetilde{\mathbf{T}}_{2,1}\right) & & \dots \\ \left(\mathbf{B}^{(k)}_{2,1} \cdot \widetilde{\mathbf{B}}_{2,1}\right) & & \dots \\ & \left(\mathbf{T}^{(k)}_{2,2} \cdot \widetilde{\mathbf{T}}_{2,2}\right) & \dots \\ & \left(\mathbf{B}^{(k)}_{2,2} \cdot \widetilde{\mathbf{B}}_{2,2}\right) & \dots \end{bmatrix} \mathbf{P}_1 \mathbf{S}_1 \\ &= \begin{bmatrix} \mathbf{T}^{(k)}_{2,1} & & & \mathbf{T}^{(k)}_{2,3} & & \dots \\ & \mathbf{B}^{(k)}_{2,1} & & & \mathbf{B}^{(k)}_{2,3} & \dots \\ & & \mathbf{T}^{(k)}_{2,2} & & & \dots \\ & & & \mathbf{B}^{(k)}_{2,2} & & \dots \end{bmatrix} \\ &\quad \cdot \begin{bmatrix} \widetilde{\mathbf{T}}_{2,1} & & & \\ \widetilde{\mathbf{B}}_{2,1} & & & \\ & \widetilde{\mathbf{T}}_{2,2} & & \\ & \widetilde{\mathbf{B}}_{2,2} & & \\ & & \ddots & \end{bmatrix} \mathbf{P}_1 \mathbf{S}_1. \end{aligned}$$

And so the scheme continues. For each iteration the number of diagonals in the left matrix is halved, the number of blocks in each diagonal is doubled, and the height of each block is roughly halved. Eventually the left matrix consists only of a single diagonal band of blocks, and further compression is impossible. This becomes the residual matrix $\mathbf{R}$ of Equation (21).

The efficiency of the scheme relies on the non-trivial requirement that the $\mathbf{T}^{(k)}$ and $\mathbf{B}^{(k)}$ blocks are rank-deficient at every level of the algorithm. To get a handle on which matrices exhibit this behavior, we start with assuming the *rank property*, namely, that any contiguous rectangular sub-block of $\mathbf{\Lambda}$, up to the numerical precision chosen, has rank proportional to the number of elements in the sub-block. That is, the rank does not depend on the location or shape of the block. Now, each time the butterfly algorithm joins two skeletons, such as $[\mathbf{T}^{(k)}_{1,1}\ \mathbf{T}^{(k)}_{1,2}]$ in Equation (22), the resulting matrix has roughly $2k$ columns while spanning out a corresponding block of $\mathbf{\Lambda}$ of rank k. Therefore, half of the columns can be eliminated by applying the ID. Since the data volume is roughly halved at each compression level, and since $\mathbf{S}_i$ at each level has $O(L)$ interpolative matrices of size roughly $k \times 2k = O(1)$, the resulting compressed representation of $\mathbf{\Lambda}$ has $O(L \log L)$ elements. See Tygert (2010) for a more detailed argument.

O'Neil et al. (2010) prove the rank property in the case of Fourier transforms and Fourier–Bessel transforms. It is, however, not proven in the case of associated Legendre functions $\widetilde{P}^m_\ell(z)$. Figure 2 shows our results for resolutions up to $L \sim 130{,}000$; we discuss these results further in Section 4.3.

3.3. Notes on Interpolation

Tygert (2008) describes an elegant and exact interpolation scheme that, in the case of the HEALPix grid and $L = 2N_{\text{side}}$, reduces the number of required evaluation points for $q_m(z_k)$ by 2/3. Although our conclusion was not to include this step in our code, we include a brief discussion in order to motivate our decision.

We focus on the even Legendre functions; the odd case is similar. Let n be an integer such that $L < m + 2n$. The function $P^m_{m+2n}(x)$ has n roots in the interval $(0, 1)$, which we denote $z_1, \dots, z_n$. Now, assuming that we have evaluated q^{even}_m in these roots, we can interpolate to any other point $y \in (-1, 1)$ by using the formula

$$q^{\text{even}}_m(y) = \omega(y) \sum_{i=1}^{n} \frac{\gamma(z_i)}{y^2 - z_i^2} q^{\text{even}}_m(z_i), \quad (23)$$

for some pre-computed weights $\omega(y)$ and $\gamma(z_i)$. The proof relies on the Christoffel–Darboux identity for the normalized associated Legendre functions (Tygert 2008; Jakob-Chien & Alpert 1997). The Fast Multipole Method (FMM) allows the computation of Equation (23) for p points with operation count of order $O(p + n)$ rather than $O(pn)$. The FMM was originally developed for accelerating N-body simulations but is here motivated algebraically. For more information about one-dimensional FMM we refer to Yarvin & Rokhlin (1999) and Dutt et al. (1996).

The reason we did not include this step in our code is that much of the interpolation is already embedded in the butterfly matrix compression. Consider for instance $N_{\text{side}} = 2048$, $L = 3N_{\text{side}}$, and $m = 2000$. The full matrix $\mathbf{\Lambda}$ occupies 65 MiB when evaluated in the HEALPix colatitude nodes and only 49 MiB when evaluated in the optimal nodes as described above. However, after compression the difference is only 10.4 MiB versus 9.4 MiB. Thus, the butterfly compression compensates, at least partially, for the oversampling. Indeed, Martinsson

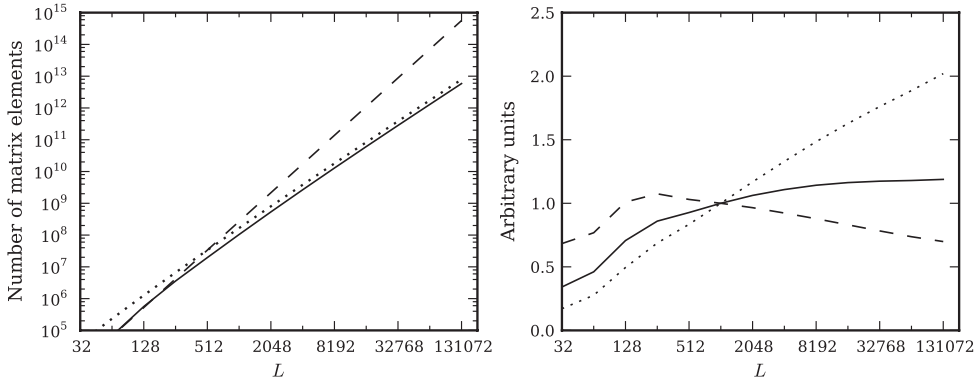


Figure 2. Efficiency of the butterfly compression scheme. Left panel: estimated size of compressed data (solid) compared with uncompressed matrix size (dashed). A line proportional to $O(L^2 \log^2 L)$ (dotted) is shown for comparison. In each case we use a HEALPix grid with $N_{\rm side} = L/2$. Right panel: a closer look at the computational scaling. The size of the compressed data is shown divided by $O(L^2 \log^3 L)$ (dashed), $O(L^2 \log^2 L)$ (solid), and $O(L^2 \log L)$ (dotted), using an arbitrary normalization.

& Rokhlin (2007) use a strongly related matrix compression technique to implement the FMM itself.

Interpolation also causes the pre-computed data to become independent of the chosen grid and resolution. However, we found the constant pre-factor in the FMM to be quite high, and including it only as a matter of convenience appears to be out of the question for our target resolutions. Since the FMM has a linear computational scaling, the question should be revisited for higher resolutions.

3.4. CPU and Memory Trade-offs

So far we have focused on reducing the number of floating point operations (FLOPs). However, during the past decade the speed of the CPU has increased much more rapidly than the system memory bandwidth, so that in current multi-core computers it is easy to get in a situation where the CPUs are starved for data to process. When processing only one or a few transforms concurrently, the volume of the pre-computed data is much larger than the volume of the maps being transformed, so that the limitation is moving the pre-computed data over the memory bus, not processing power. Note that in the case of very many simultaneous transforms the problem is alleviated since the movement of pre-computed data is amortized. Following in the footsteps of libpsht, and our own requirements in CMB analysis, we have restricted our attention to between 1 and 10 concurrent transforms. While the butterfly algorithm probably performs well in the face of many concurrent transforms, it would require additional blocking and optimization beyond what we have implemented, so that movement of the working set in memory is properly amortized. Note that as each m is processed independently, the working set is only about $1/L$ of the total input.

The considerations above motivate stopping compression early, after a significant reduction in the FLOP count has been achieved, but before the size of the pre-computed data becomes too large (see Figure 3). Butterfly compression has the convenient feature that the blocks in the residual matrix $\mathbf{R}$ consist of contiguous slices from columns of $\mathbf{\Lambda}$. By orienting $\mathbf{\Lambda}$ so that rows are indexed by ℓ and columns by z, the elements of the residual blocks can be computed on the fly from three-term recurrence formulas for the associated Legendre functions. We return to this topic in Appendix A.2.

As an example, consider $N_{\rm side} = 2048$ and $m = 400$. The uncompressed matrix $\mathbf{\Lambda}$ takes 64 MB in double precision. This

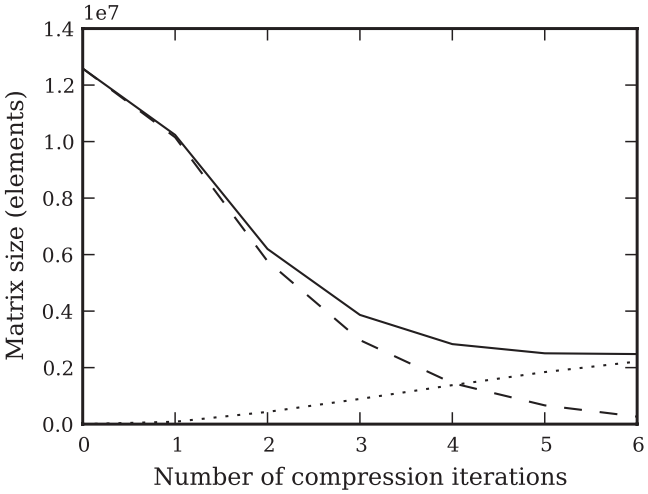


Figure 3. Effect of each level of butterfly compression. The size of the compressed data (solid) is the sum of elements in residual uncompressed blocks in $\mathbf{R}$ (dashed) and the interpolation matrices $\mathbf{S}_i$ (dotted). While $\mathbf{R}$ can be generated on the fly during transforms, the $\mathbf{S}_i$ needs to be stored as pre-computed data, so that the choice of compression level is a trade-off between CPU use and the size of the pre-computed data. Parameters for this figure are $N_{\rm side} = 2048$, $L = 3N_{\rm side}$, $m = 0$, and the initial chunk size 32 columns.

can be compressed to 20% of the original size by using five levels of compression, with the uncompressed residual $\mathbf{R}$ accounting for about 13% of the compressed data. If one instead stops after three levels of compression, then although the size of the compressed data has now grown to 24% of the original, 57% of this is made out of elements in $\mathbf{R}$. Since one only needs to store two elements for every column of 512 elements in $\mathbf{R}$ and can generate the rest on the fly, stopping compression after three levels reduces the memory bus traffic and size of pre-computed data by about 40%, at the cost of some extra CPU instructions. Note that the brute-force codes may simply be seen as the limit of zero levels of compression.

4. IMPLEMENTATION AND RESULTS

4.1. Technology

The Wavemoth library is organized in a core part and an auxiliary part. The core is primarily written in C and contains the routines for performing SHTs. The auxiliary shell around the core is written as a Python package and is

responsible for generating the pre-computed data using the butterfly compression algorithm, as well as the regression and unit tests.

By writing the core in pure C, we remain close to the hardware and make sure the library can be used without Python. C remains the easiest language to call from other languages such as Fortran, C++, Java, Python, MATLAB, and so on. By using Python in the auxiliary support code, we accelerate development of the parts that are not performance critical and make writing tests a pleasant experience. Being able to quickly write up unit tests is an indispensable tool, as it allows optimizing the C code iteratively without introducing bugs. Since individual pieces of the C core are tested, there is both a public API for end-users and a private API that is used from Python to test individual C routines in isolation. Much of the support code is implemented in the Cython language (Behnel et al. 2011), which bridges the worlds of Python and C.

The C core depends on files containing pre-computed data, a Fourier transform library, and a BLAS library. For the latter two we use FFTW3 (Frigo & Johnson 2005) and ATLAS (Whaley et al. 2001), respectively. Parts of the Wavemoth core are written using templates in order to generate many slight variations of the same C routine. We use Tempita,[2] a purely text-oriented templating language, and find this to be much more convenient for optimizing a computational core than the type-oriented templates of C++. During the pre-computations, we use the open source Fortran 77 library ID[3] to compute the Interpolative Decomposition and libpsht to generate the associated Legendre functions.

Unlike libpsht, we have not focused on portability, and Wavemoth is only tested on 64-bit Linux with the GCC compiler on Intel-platform CPUs. Computational cores are written using intrinsic SSE functions and 128-bit registers. More work is needed for optimal performance on the latest Intel micro-architecture, which supports 256-bit registers, or on non-Intel platforms. Beyond that, we expect no hurdles in improving portability.

4.2. Benchmarks

We include benchmarks for two different systems with different memory bandwidth, as Wavemoth's performance is deeply influenced by this aspect of the hardware. Figure 4 presents benchmarks taken on a 64-core 2.27 GHz Intel Xeon X7560 (Nehalem micro-architecture), which has a compute-to-bandwidth ratio of about 45:1. Figure 5 presents benchmarks taken on a 48-core 2.2 GHz AMD Opteron 6174. The compute-to-bandwidth ratio is in this case about 64:1, significantly worse than the Intel system.[4] The consequence is that butterfly compression gives less of an advantage, with only about four times speedup over libpsht at $L = 4096$, compared with the corresponding six times speedup achieved on the Intel system. In the case of 10 simultaneous transforms, libpsht achieves a very consistent $2\times$ speedup that Wavemoth is not able to fully match, as most of our tuning effort has been on the single transform path.

[2] http://pythonpaste.org/tempita/

[3] http://cims.nyu.edu/~tygert/software.html

[4] The Intel system supports transfer of 13 billion numbers per second and has theoretical peak compute power 580 GFLOPS, using all 64 cores. The AMD system supports transfer of 6.5 billion numbers per second and has theoretical peak compute power of 422 GFLOPS, using all 48 cores. All numbers refer to double-precision floating point.

The highest tested accuracy of $\epsilon = 10^{-13}$ for the Legendre transforms was chosen because current codes using the HEALPix grid only agree to this accuracy on high resolutions (Reinecke 2011).

An important aspect of the systems for our purposes is the non-uniform memory access (NUMA). On each system, the CPU cores are grouped into eight nodes, and the RAM chips evenly divided between the nodes. Each CPU only has direct access to RAM chips on the local node and must go through a CPU interconnect bus to access other RAM chips. For consistent performance we need to ensure that Wavemoth distributes the pre-computed data in such a way that each CPU finds the data it needs in its local RAM chips. In the benchmarks we always use a whole number of nodes, so that computation power and memory bandwidth scale together. The exception is benchmarks using a single core, but in those cases, Wavemoth's pre-computed data fit in cache anyway.

Table 1 list the sizes of the pre-computed data. To balance bandwidth and CPU requirements as described in Section 3.4, the pre-computation code takes a parameter ρ, specifying the cost of FLOPs in the bandwidth-intensive butterfly matrix application stage relative to the cost of FLOPs in the CPU-intensive brute-force Legendre transform stage. The parameter was then tuned for the single-transform case for $L = 4096$, resulting in optimal choices of $\rho = 7.5$ on the Intel system and $\rho = 18$ on the AMD system. Performing the pre-computations scales as $O(L^3)$. In the case of no compression, we still store the pre-computed quantities necessary for the Legendre recurrence relations in memory, as described in the Appendix. Loading these data from memory is not necessarily faster than computing them on the fly, but doing so saved some development time.

All methods involved are numerically stable and well understood, so we do not include a rigorous analysis of numerical accuracy. Table 2 lists the relative error from transforming a single set of standard Gaussian coefficients per configuration. We use the relative error

$$\epsilon = \sqrt{\sum_{i=1}^{N_{\mathrm{pix}}} (x_i - y_i)^2 / \sum_{i=1}^{N_{\mathrm{pix}}} x_i^2}, \tag{24}$$

where x_i denote the result of libpsht and y_i the result of our code. The discrepancies in the no-compression, high-L cases are due to using a different recurrence for the associated Legendre functions, as described in Appendix A.2. As we did not compare with higher precision results, it is not clear whether it is our code, libpsht, or both that lose precision with higher resolution. Note that the input data to the butterfly compression are generated using libpsht.

4.3. Higher Resolutions

Because of memory constraints we have not gone to higher resolutions than $L \sim 8000$. Instead, we provide estimates for the number of required FLOPs. Tygert (2010) provides similar estimates but focuses on the behavior for the Legendre transform for single m rather than the full SHT.

At each resolution, we compress $\mathbf{\Lambda}_m^{\mathrm{odd}}$ for 20 different m and fit the cost estimate

$$\hat{c}_m = \alpha + \sum_{p=0}^{2} \beta_p m \log(1+m)^p \tag{25}$$

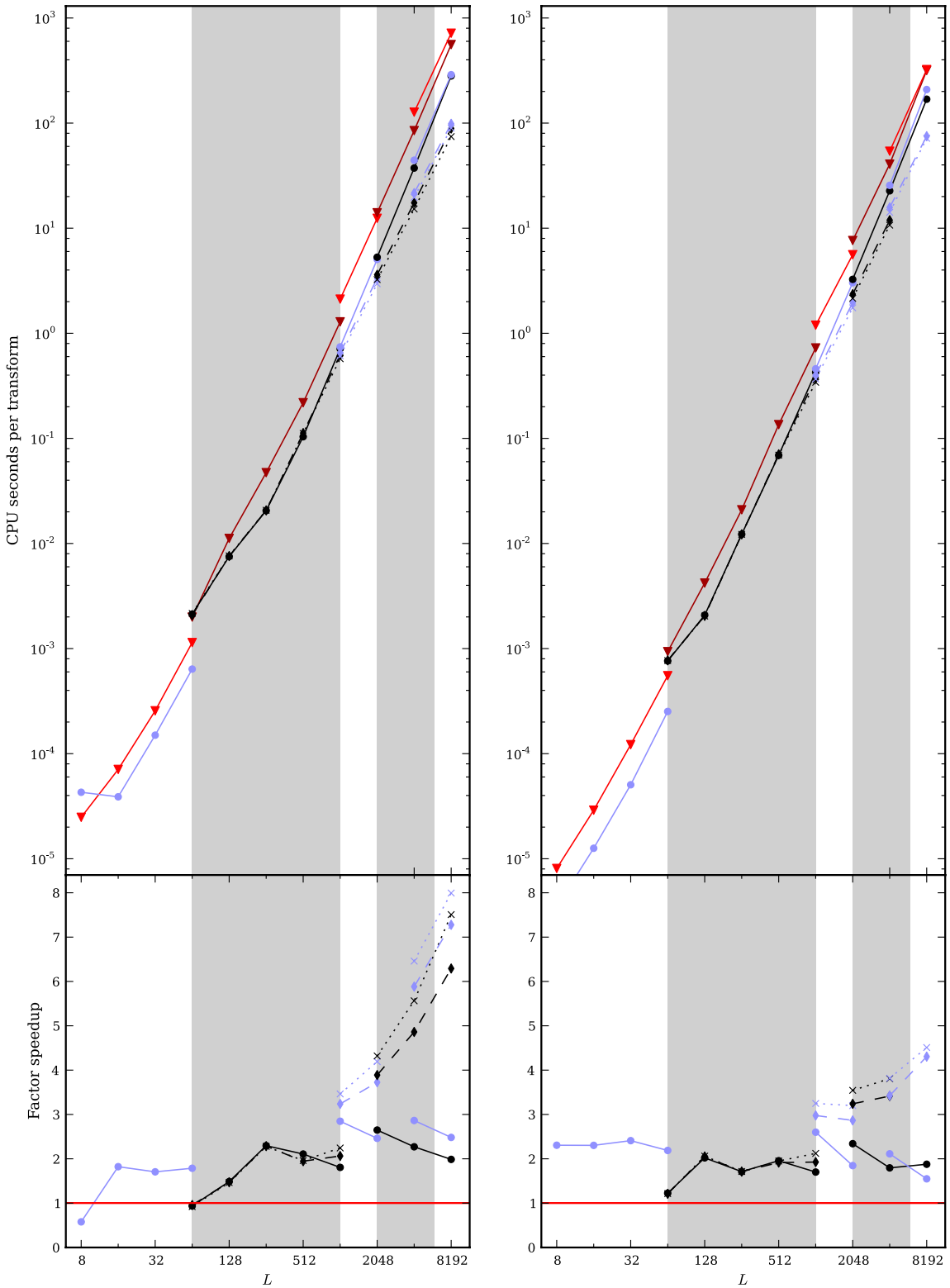


Figure 4. Benchmarks for full SHTs performed on the Intel system. The left panel shows timings for a single transform, the right for 10 simultaneous transforms. We scale up the number of CPU cores together with the resolution. Each pane is divided into four partially overlapping segments corresponding to 1, 8, 16, 32, and 64 CPU cores, respectively (indicated by white/gray backgrounds and changes in line colors). The libpsht code (red triangles) is compared with Wavemoth (blue/black) with no compression (solid, circles), compression with precision 10^{-13} (dashed, diamonds), and compression with precision 10^{-8} (dotted, crosses). In each case we use a HEALPix grid with resolution $N_{\mathrm{side}} = L/2$. Note for instance how both codes suffer from parallelization overhead at the transition from one to eight cores, but that libpsht suffers less and catches up with Wavemoth. For a single transform at high resolutions, the situation is the contrary, with Wavemoth parallelizing better at the jump from 16 to 32 cores and from 32 to 64 cores. We repeated each benchmark multiple times both with and without HyperThreading and report the fastest wall clock time achieved multiplied with the number of CPU cores used and divided by the number of simultaneous transforms. Some 32-core timings for 10 simultaneous transforms at $L = 8192$ could not be obtained owing to memory limitations. The load time of the pre-computed data from the hard drive is not included.

(A color version of this figure is available in the online journal.)

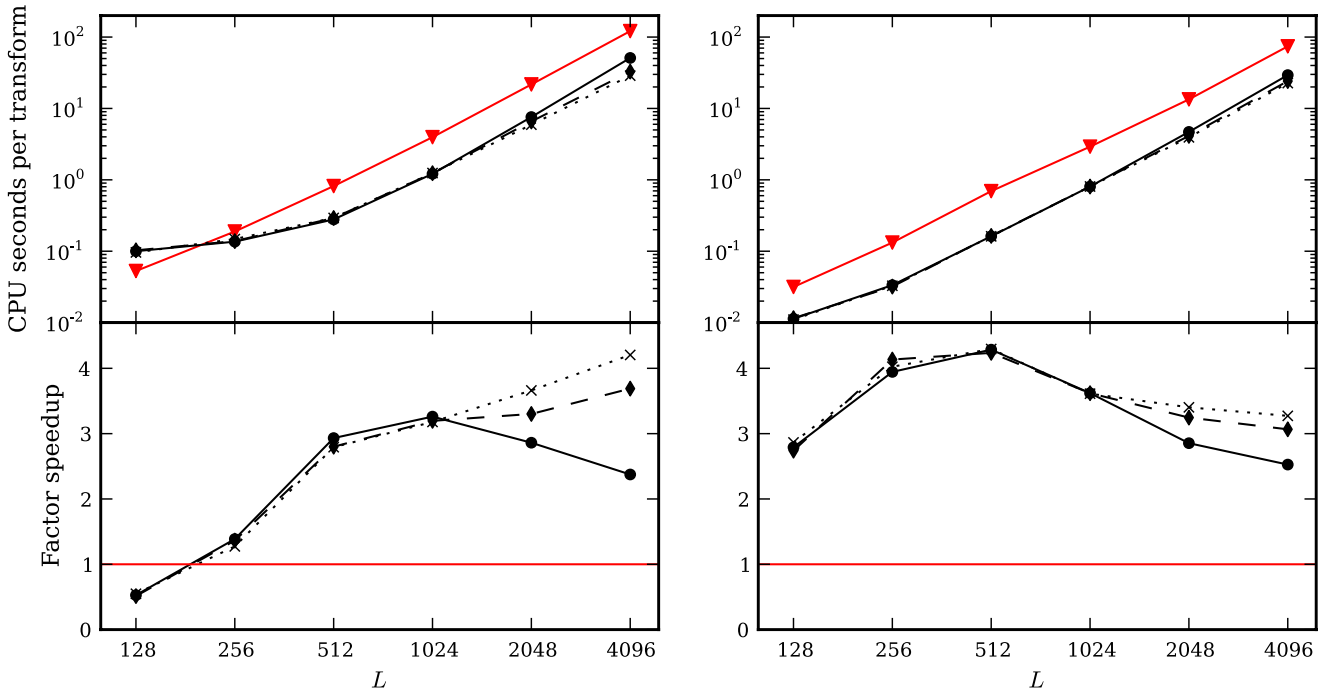


Figure 5. Benchmarks for full SHTs performed on the AMD system, using all 48 CPU cores. The libpsht code (red triangles) is compared with Wavemoth (blue) with no compression (solid, circles), compression with precision 10^{-13} (dashed, diamonds), and compression with precision 10^{-8} (dotted, crosses). Left panel shows a single transform and the right panel 10 simultaneous transforms. In each case we use a HEALPix grid with resolution $N_{\text{side}} = L/2$. The large speedup in the range $L = 256..1024$ is in part due to Wavemoth scaling better to all 48 cores and is closer to a 2× speedup when using fewer cores. We repeated each benchmark multiple times and report the fastest wall clock time achieved multiplied with the number of CPU cores used and divided by the number of simultaneous transforms.
(A color version of this figure is available in the online journal.)

Table 1
Size of Pre-computed Data

L	No Comp.	Intel ($\rho = 7.5$)		AMD ($\rho = 18$)		Pre-computation Time
		Tol. 10^{-13}	Tol. 10^{-8}	Tol. 10^{-13}	Tol. 10^{-8}	(CPU Minutes)
32	130 KiB	...	...	...	...	0.02
64	496 KiB	...	...	...	...	0.03
128	2.0 MiB	2.0 MiB	2.0 MiB	1.9 MiB	1.9 MiB	0.22
256	8.0 MiB	8.0 MiB	8.0 MiB	7.1 MiB	7.1 MiB	2.6
512	27 MiB	174 MiB	187 MiB	27 MiB	27 MiB	7.4
1024	102 MiB	937 MiB	988 MiB	102 MiB	170 MiB	12
2048	389 MiB	6.0 GiB	5.8 GiB	4.4 GiB	4.3 GiB	90
4096	1.5 GiB	38 GiB	35 GiB	28 GiB	27 GiB	536
8192	5.8 GiB	212 GiB	208 GiB	...	...	4380

Notes. The pre-computation time quoted is the wall time taken to compute at tolerance 10^{-13} on the Intel system, multiplied by the number of CPU cores used. We use 1 core for $L = 32$ and then scale up gradually to 64 cores at $L = 8192$. The pre-computed data are saved to a network file system.

by least-squares minimization in the parameters α, β_0, β_1, and β_2. The final cost is then estimated by

$$\hat{c}_{\text{total}} = 2 \sum_{m=0}^{L} \hat{c}_m, \tag{26}$$

since $\mathbf{\Lambda}_m^{\text{even}}$ and $\mathbf{\Lambda}_m^{\text{odd}}$ have almost identical behavior. The results can be seen in Figure 2. For $L \sim 130{,}000$, the butterfly algorithm requires only 1% of the arithmetic operations of a brute-force transform. The size of the pre-computed data at this resolution is around 45 TiB in double precision, although this can be reduced by using the hybrid approach of Section 3.4.

At low resolutions, the algorithm is bound by the $O(L^3)$ operations of the brute-force Legendre transform. At high resolutions, the $O(L^2 \log^2 L)$ trajectory is clearly a better fit than the $O(L^2 \log L)$ scaling conjectured by Tygert (2010). Note that the numerical evidence presented in Tygert (2010) shows that the average k increases monotonically with m, so it may indeed be the case that the rank property is not fully satisfied, or only satisfied conditionally on m. The benchmark results of Tygert (2010) seem to be in agreement with the $O(L^2 \log^2 L)$ hypothesis as well.

4.4. Comparison with Other Fast SHT Algorithms

A widely known scheme for fast SHTs is the $O(L^2 \log^2 L)$ transform of Healy et al. (2003), implemented in SpharmonicKit. It algebraically expresses a Legendre transform of degree L as a function of two Legendre transforms of degree $L/2$,

Table 2
Samples of Numerical Accuracy

L	No Compression	Tolerance 10^{-13}	Tolerance 10^{-8}
8	8.6e-16	...	...
16	1.4e-15	...	...
32	2.7e-15	...	...
64	5.8e-15	...	...
128	1.2e-14	...	...
512	5.1e-14	4.7e-14	1.9e-09
1024	1.3e-13	8.9e-14	2.4e-09
2048	2.7e-13	1.7e-13	3.1e-09
4096	6.4e-13	3.3e-13	3.7e-09
8192	2.2e-12	6.6e-13	4.2e-09

Notes. In each case, the transform of a single Gaussian sample is compared with libpsht double-precision results.

resulting in a divide-and-conquer scheme similar to the FFT algorithms. Unfortunately, the scheme is inherently numerically unstable, and special stabilization steps must be incorporated. Also, it is restricted to equiangular grids, so that it cannot be used directly with the HEALPix or GLESP grids. Wiaux et al. (2006) benchmark SpharmonicKit against the original HEALPix implementation (pre 2.20) and find that it is almost three times slower at $L = 1024$. Keep in mind that libpsht, used in present releases of HEALPix, is about twice as fast as the original HEALPix implementation. Considering the above, we stop short of a direct comparison between Wavemoth and SpharmonicKit. Note that while SpharmonicKit achieves much higher accuracy of an SHT round-trip than HEALPix does, this is an effect of the different sampling grids being used, not of the computational method, and it is straightforward to extend the Wavemoth code to use the same grid as SpharmonicKit.

Mohlenkamp (1999) uses a matrix compression technique similar to the one employed in this paper, which is independent of the pixel grid chosen. A matrix related to the $\mathbf{\Lambda}$ of the present paper is locally approximated by truncated trigonometric series. The resulting SHT algorithm scales as $O(L^{5/2} \log L)$. As shown in Figure 6, the code behaves very similarly to our code at medium resolution, as long as one does not require too much numerical accuracy. The size of the pre-computed data is also of the same order, sometimes half and sometimes double that of Wavemoth's data.

Note that libftsh appears to have potential for optimization for modern platforms, and this should be taken into account when comparing the algorithms. Owing to its age, libftsh makes assumptions about 32-bit array sizes that prevent comparison at higher resolutions without porting libftsh to 64-bit. The libftsh code contains an implementation of the Legendre transforms only, and not of the full SHTs. It should be straightforward to modify Wavemoth to use libftsh for its Legendre transforms in order to perform full SHTs using this algorithm.

The compression scheme of Mohlenkamp (1999) appears to be very competitive for low-accuracy transforms, but less so if higher precision is needed. It may be fruitful to hybridize the algorithms of Tygert (2010) and Mohlenkamp (1999) and use both together to compress a single matrix. Even if that does not work, one can simply use whichever performs best for a given m.

5. DISCUSSION

There is significant potential in speeding up SHTs beyond the codes in popular use today. We achieved a 2× speedup at low and medium resolutions simply due to restructuring how the brute-force computations are done, and we believe there is potential for even more speedup if time is spent on profiling and micro-optimization. In particular, our code is underoptimized for multiple simultaneous transforms.

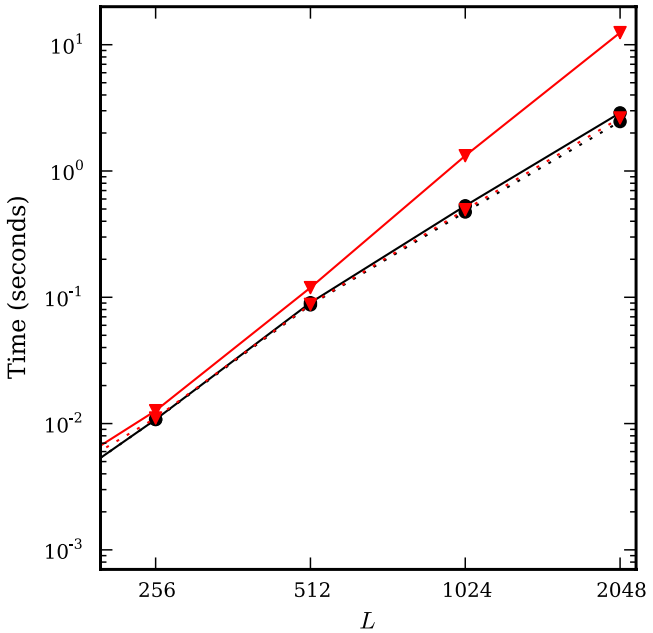


Figure 6. Comparison of Wavemoth (black circles) with the algorithm of Mohlenkamp (1999) as implemented in libftsh (red triangles), with accuracy 10^{-13} (solid) and 10^{-8} (dotted). Both codes are run on a single core. Only the Legendre transform part is benchmarked, as libftsh does not implement the full SHT. Wavemoth uses a HEALPix grid with $2L - 1$ rings, while libftsh uses a Gaussian grid with $2L$ rings. The placement of the rings should make little difference to the performance of either code.

(A color version of this figure is available in the online journal.)

At the highest resolutions in practical use in cosmology today, $L \sim 4000$, use of the butterfly compression is borderline. One the one hand, it does yield an additional 2× speedup, potentially much more if one needs less accuracy. On the other hand, it requires between 30 and 40 GiB of pre-computed data in memory and the transportation of those data over the memory bus for every set of transforms. The result is a delicate balance between bandwidth and achieved speedup; for every number stored in the pre-computed data, one might save 40 arithmetic operations, but then again computation is much cheaper than accessing system memory on present-day computer architectures.

In Section 3.3, we note the existence of interpolation schemes that cut the necessary sample points for brute-force codes by two-thirds in the case of the HEALPix grid, although performing the interpolation step does not come for free. It seems that the speedup from such interpolation alone could be on the same order as what the butterfly algorithm achieves for the current needs of CMB research. The advantage is that it does not require nearly as much pre-computed data and is so much less architecture-dependent and easier to micro-optimize. In going forward we therefore anticipate spending more effort on direct interpolation schemes and less effort on matrix compression. For resolutions higher than those needed in CMB analysis, matrix compression schemes seem like the most mature option at the moment.

We have not discussed spin-weighted SHTs, which are crucial to analyzing the polarization properties of the CMB. However, Kostelec et al. (2000) and Wiaux et al. (2007) describe how the transform of a polarized CMB map can be reduced to three scalar transforms. This would additionally help amortize the memory bus transfer of the pre-computed data. Alternatively, it may be possible to compress the spin-weighted spherical harmonic operators.

We consider Wavemoth an experimental code for the time being, and spherical harmonic analysis has been left out. This was done purely to save implementation time, and we know of no obstacles to implementing this using the same methods. The code also lacks support for MPI parallelization, although we expect adding such support to be straightforward. The only inter-node communication requirement is a global transpose of $q_m(z_k)$ between the Legendre transforms and the Fourier transforms.

The author thanks S. K. Næss, H. K. Eriksen, M. Tygert, M. Reinecke, and M. Mohlenkamp for useful discussions, and M. Omang and F. Hansen for lending the benchmark hardware. The author is funded by European Research Council grant StG2010-257080. The benchmark hardware is funded by the Norwegian Defence Estates Agency and the Research Council of Norway.

APPENDIX

IMPLEMENTATION DETAILS

A.1. Applying the Compressed Matrix Representation to a Vector

On modern computers, the primary bottleneck is often to move data around. Fundamental design decisions were made with this in mind. Looking at the compressed representations of $\mathbf{\Lambda}$ in Section 3.2, the immediate algorithm that comes to mind for computing $\mathbf{\Lambda x}$ or $\mathbf{\Lambda}^T\mathbf{x}$ is the breadth-first approach: First compute $\mathbf{S}_1\mathbf{x}$, then permute the result, then compute $\mathbf{S}_2(\mathbf{P}_1\mathbf{S}_1\mathbf{x})$, and so on. However, this leads to storing several temporary results for longer than they need to, since the rightmost permutations are very local permutations, and only the leftmost permutation is fully global. Therefore, we instead traverse the data dependency tree set up by the permutations in a depth-first manner. The advantage of this approach is that it is *cache oblivious* when transforming a few vectors at a time. That is, it automatically minimizes data movement for any cache hierarchy, whereas breadth-first traversal will always drop to the memory layer that is big enough to hold the entire set of input vectors. Note that for transforming many maps at the same time, cache-size-dependent blocking should be implemented in addition, but we have stopped short of this. Like Tygert (2010), we also do the compression during pre-computation depth-first, which ensures that, per m, memory requirements go as $O(L \log L)$ even though computation time goes as $O(L^2)$.

The core computation during tree traversal is to apply the interpolative matrices, e.g., $\widetilde{\mathbf{T}}\mathbf{x}$ or $\widetilde{\mathbf{T}}^T\mathbf{x}$. Keep in mind that the k-by-n matrix $\widetilde{\mathbf{T}}$ contains the k-by-k identity matrix in a subset of its columns; making use of this is important as it roughly halves the storage size and FLOP count. Given an ID $\mathbf{T} = \mathbf{T}^{(k)}\widetilde{\mathbf{T}}$, we can freely permute the rows of $\widetilde{\mathbf{T}}$, simply by permuting the columns of $\mathbf{T}^{(k)}$ correspondingly. We do this during pre-computation to avoid the unordered memory usage pattern of arbitrary permutations. Instead, we can simply filter the input or output vectors into the part that hits the identity sub-matrix and the part that hits the dense sub-matrix.

A.2. Efficient Code for Legendre Transforms

As mentioned in Section 3.4, it is necessary to balance the amount of pre-computed data to the memory bandwidth, so code is required to apply the residual blocks in $\mathbf{R}$ to vectors without actually storing $\mathbf{R}$ in memory. This means computing a cropped version of the Legendre transform,

$$q'(z_j) = \sum_{k=k_{\text{start}}}^{k_{\text{stop}}} \widetilde{P}^m_{m+2k+t}(z_j) a_{\ell m}, \tag{A1}$$

where $t = 0$ for the even transforms and $t = 1$ for the odd transforms. To compute $\widetilde{P}^m_\ell$, we use a relation that jumps two steps in ℓ for each iteration (Tygert 2010):

$$\begin{aligned}\widetilde{P}^m_{\ell+2}(z) &= \frac{z^2 - d^m_l}{c^m_\ell}\widetilde{P}^m_\ell(z) - \frac{c^m_{\ell-2}}{c^m_\ell}\widetilde{P}^m_{\ell-2}(z) \\ &\equiv (z^2 + \alpha^m_\ell)\beta^m_\ell \widetilde{P}^m_\ell(z) + \gamma^m_\ell \widetilde{P}^m_{\ell-2}(z),\end{aligned} \tag{A2}$$

with

$$c^m_\ell = \sqrt{\frac{(\ell - m + 1)(\ell - m + 2)(\ell + m + 1)(\ell + m + 2)}{(2\ell + 1)(2\ell + 3)^2(2\ell + 5)}}$$

and

$$d^m_\ell = \frac{2\ell(\ell + 1) - 2m^2 - 1}{(2\ell - 1)(2\ell + 3)}.$$

This recurrence relation requires five arithmetic operations per iteration, as opposed to a more widely used relation that takes one step in ℓ and only needs four arithmetic operations per step (see, e.g., Press et al. 2007). However, since $\mathbf{\Lambda}^{\text{even}}$ and $\mathbf{\Lambda}^{\text{odd}}$ may have different columns in the residual blocks of their compressed representations, relation (A2) is a better choice in our case.

For each block in $\mathbf{R}$ we pre-compute α^m_ℓ, β^m_ℓ and γ^m_ℓ, as well as $\widetilde{P}^m_{k_{\text{start}}}(z)$ and $\widetilde{P}^m_{k_{\text{start}}+1}(z)$ for each z for initial conditions. Note that $\widetilde{P}^m_\ell(z)$ in parts of its domain take values so close to zero that they cannot be represented in IEEE double precision. However, in these cases $\widetilde{P}^m_\ell(z)$ is always increasing in the direction of increasing ℓ, so we can simply increase k_{start} correspondingly. In fact, we follow libpsht and assume that the dynamic range of the input data is small enough, within each m, that values of $\widetilde{P}^m_\ell(z)$ smaller than 10^{-30} in magnitude can safely be neglected. As far as possible we group together six and six columns with the same k_{start} and k_{stop}, for reasons that will soon become clear.

For an efficient implementation, the first important point is to make sure the number of loads from cache into CPU registers is balanced with the number of FLOPs. The second is to make sure there are enough independent FLOPs in flight simultaneously, so that operations can be pipelined. Thus,

1. for performing a single transform with one real and one imaginary vector, the values of $\widetilde{P}^m_\ell$ should never need to leave the CPU registers. Rather, we fuse Equations (A1) and (A2) in the core loop. For multiple simultaneous transforms we save $\widetilde{P}^m_\ell$ to cache but make sure to process in small batches that easily fit in L1 cache.
2. We process for several z_j simultaneously. This amortizes the register loads of α^m_ℓ, β^m_ℓ, and γ^m_ℓ. It also ensures that there are multiple independent chains of computation going on so that pipelining works well.

In the single transform case with one real and one imaginary vector, we do the full summation for six z_j at a time (when

possible). The allocation of the 16 available 128-bit registers, each holding two double-precision numbers, then becomes three registers for $\widetilde{P}^m_\ell$, three for $\widetilde{P}^m_{\ell-2}$, three for the auxiliary data α^m_ℓ, β^m_ℓ, and γ^m_ℓ, six accumulation registers for $q'(z_j)$, and one work register. The z_j^2 values are, perhaps counterintuitively, read again from cache in each iteration, which conserves three registers and thus enables processing six z_j in each chunk instead of only four without register spills. Finally, when the time comes for multiplying $\widetilde{P}^m_\ell$ with $a_{\ell m}$, the auxiliary data are no longer needed, leaving room for loading $a_{\ell m}$.

On the Intel Xeon system, the routine performs at 6.46 GFLOP/s per core (71% of the theoretical maximum) when benchmarked on all the Legendre transforms necessary for a full SHT at $L = 4096$ using 32 cores. The effect of instruction pipelining is evident; reducing the number of columns processed in each iteration from six to four reduces performance to 5.69 GFLOP/s (63%), and when only processing two columns at a time, performance is only 4.28 GFLOP/s (47%).

We skip the details for the multiple transform case, but in short, it involves the same sort of blocking performed for matrix multiplication, including repacking the input data in blocks. Goto & van de Geijn (2008) provide an excellent introduction to blocking techniques. In this case the performance is 5.60 GFLOP/s (62%) per core when performing the Legendre transforms necessary for 10 simultaneous SHTs.

The considerations above guided the choice of loop structure, which was then implemented in pure C using SSE intrinsic. We did not spend much time on optimization, so there should be room for further improvements, in particular, for the multiple-transform path.

A.3. Data Layout

The butterfly compression algorithm naturally leads to the following code organization for spherical harmonic synthesis:

1. Since each m is processed independently, we request input in m-major ordering. Also, for multiple simultaneous transforms, the coefficients of each map are interleaved, which is optimal for both the butterfly algorithm and the brute-force cropped Legendre transforms. In most places, the real and complex parts of the input can be treated as two independent vectors, since $\mathbf{\Lambda}$ is a real matrix.
2. Compute all $q_m(z_j)$ into a two-dimensional array. Since each m is processed independently, this ends up in m-major ordering, like the input.
3. While transposing the $q_m(z_j)$ array into ring-major ordering, phase-shift and wrap around the coefficients, and perform FFTs on each ring. Rings must be processed in small batches in order to avoid loading cache lines multiple times.

A temporary work buffer with size of the same order as the input and output is used for $q_m(z_j)$. An in-place code should be feasible with the use of an in-place transpose.

A drawback compared with brute-force codes is that $q_m(z_j)$ needs to first be written to and then read from main memory. Here, libpsht is instead able to employ blocking, so that a few rings at a time are completely processed before moving on. Our benchmarks do, however, indicate that this is not a big problem in practice. Also, for cluster parallelization using MPI, it would be natural to follow S^{2}HAT (Hupca et al. 2010; Szydlarski et al. 2011) in distributing the input data by m and the output data by rings, which also leads to a global transpose operation.

Wavemoth stores the output maps in interleaved order, since FFTW3 is able to deal well with such transforms. The libpsht code is able to support any output ordering, although stacked, non-interleaved maps are slightly faster, so that is the ordering we use for libpsht in the benchmarks.

REFERENCES

Behnel, S., Bradshaw, R., Citro, C., et al. 2011, Comput. Sci. Eng., 13, 2
Cheng, H., Gimbutas, Z., Martinsson, P. G., & Rokhlin, V. 2005, SIAM J. Sci. Comput., 26, 4
Doroshkevich, A. G., Naselsky, P. D., Verkhodanov, O. V., et al. 2005, Int. J. Mod. Phys. D., 14, 275
Dutt, A., Gu, M., & Rokhlin, V. 1996, SIAM J. Numer. Anal., 33, 5
Eriksen, H. K., Jewell, J. B., Dickinson, C., et al. 2008, ApJ, 676, 1
Frigo, M., & Johnson, S. G. 2005, Proc. IEEE, 93, 2
Górski, K. M., Hivon, E., Banday, A. J., et al. 2005, ApJ, 622, 2
Goto, K., & van de Geijn, R. 2008, ACM Trans. Math. Softw., 34, 3
Healy, D. M., Rockmore, D. N., Kostelec, P. J., & Moore, S. 2003, J. Fourier Anal. Appl., 9, 4
Hupca, I. O, Falcou, J., Grigori, L., & Stompor, R. 2010, INRIA Technical Report, No. RR-7409, arXiv:1010.1260
Jakob-Chien, R., & Alpert, B. K. 1997, J. Comput. Phys., 136, 2
Kostelec, P. J., Maslen, D. K., Jr., Healy, D. M., & Rockmore, D. N. 2000, J. Comput. Phys., 162, 2
Kunis, S., & Potts, D. 2003, J. Comput. Appl. Math., 161, 1
Martinsson, P. G., & Rokhlin, V. 2007, SIAM J. Sci. Comput., 29, 3
McEwen, J. D., & Wiaux, Y. 2011, IEEE Trans. Signal Process., 59, 12
Michielssen, E., & Boag, A. 1996, IEEE Trans. Antennas Propag., 44, 8
Mohlenkamp, M. J. 1999, J. Fourier Anal. Appl., 5, 2
O'Neil, M., Woolfe, F., & Rokhlin, V. 2010, Appl. Comput. Harmon. Anal., 28, 2
Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. 2007, Numerical Recipes (3rd ed.; New York: Cambridge Univ. Press)
Reinecke, M. 2011, A&A, 526, A108
Rokhlin, V., & Tygert, M. 2006, SIAM J. Sci. Comput., 27, 6
Suda, R., & Takami, M. 2002, Math. Comput., 71, 238
Szydlarski, M., Esterie, P., Falcou, J., Grigori, L., & Stompor, R. 2011, INRIA Technical Report, No. RR-7635, arXiv:1106.0159
Tygert, M. 2008, J. Comput. Phys., 227, 8
Tygert, M. 2010, J. Comput. Phys., 229, 18
Whaley, R. C., Petitet, A., & Dongarra, J. 2001, Parallel Comput., 27, 1
Wiaux, Y., Jacques, L., & Vandergheynst, P. 2007, J. Comput. Phys., 226, 2
Wiaux, Y., Jacques, L., Vielva, P., & Vandergheynst, P. 2006, ApJ, 652, 1
Yarvin, N., & Rokhlin, V. 1999, SIAM J. Numer. Anal., 36, 2

Paper IV

Libsharp – spherical harmonic transforms revisited

When `libpsht` performance was beaten by Wavemoth and shtns, Martin Reinecke decided to reimplement it using ideas from the Wavemoth paper, and the result was `libsharp`: The *de facto* standard SHT code, used by thousands of researchers every day. During the development me and Martin was in correspondance about the strategies used to achieve high performance. I also contributed code for adjoint and real SHTs as described in section 4.5. Since the paper was published I have further contributed a couple of bugfixes, so that I am now the 2nd contributor/maintainer of the library, although Martin reigns supreme as the primary contributor.

A&A 554, A112 (2013)
DOI: 10.1051/0004-6361/201321494

Astronomy & Astrophysics

Libsharp – spherical harmonic transforms revisited

M. Reinecke[1] and D. S. Seljebotn[2]

[1] Max-Planck-Institut für Astrophysik, Karl-Schwarzschild-Str. 1, 85741 Garching, Germany
e-mail: martin@mpa-garching.mpg.de
[2] Institute of Theoretical Astrophysics, University of Oslo, PO Box 1029 Blindern, 0315 Oslo, Norway
e-mail: d.s.seljebotn@astro.uio.no

Received 18 March 2013 / Accepted 14 April 2013

ABSTRACT

We present libsharp, a code library for spherical harmonic transforms (SHTs), which evolved from the libpsht library and addresses several of its shortcomings, such as adding MPI support for distributed memory systems and SHTs of fields with arbitrary spin, but also supporting new developments in CPU instruction sets like the Advanced Vector Extensions (AVX) or fused multiply-accumulate (FMA) instructions. The library is implemented in portable C99 and provides an interface that can be easily accessed from other programming languages such as C++, Fortran, Python, etc. Generally, libsharp's performance is at least on par with that of its predecessor; however, significant improvements were made to the algorithms for scalar SHTs, which are roughly twice as fast when using the same CPU capabilities. The library is available at http://sourceforge.net/projects/libsharp/ under the terms of the GNU General Public License.

Key words. methods: numerical – cosmic background radiation – large-scale structure of Universe

1. Motivation

While the original libpsht library presented by Reinecke (2011) fulfilled most requirements on an implementation of spherical harmonic transforms (SHTs) in the astrophysical context at the time, it still left several points unaddressed. Some of those were already mentioned in the original publication: support for SHTs of arbitrary spins and parallelisation on computers with distributed memory.

Both of these features have been added to libpsht in the meantime, but other, more technical, shortcomings of the library have become obvious since its publication, which could not be fixed within the libpsht framework.

One of these complications is that the library design did not anticipate the rapid evolution of microprocessors during the past few years. While the code supports both traditional scalar arithmetic as well as SSE2 instructions, adding support for the newly released Advanced Vector Extensions (AVX) and fused multiply-accumulate instructions (FMA3/FMA4) would require adding significant amounts of new code to the library, which is inconvenient and very likely to become a maintenance burden in the long run. Using proper abstraction techniques, adding a new set of CPU instructions could be achieved by only very small changes to the code, but the need for this was unfortunately not anticipated when libpsht was written.

Also, several new, highly efficient SHT implementations have been published in the meantime; most notably Wavemoth (Seljebotn 2012) and shtns (Schaeffer 2013). These codes demonstrate that libpsht's computational core did not make the best possible use of the available CPU resources. Note that Wavemoth is currently an experimental research code not meant for general use.

To address both of these concerns, the library was redesigned from scratch. The internal changes also led to a small loss of functionality; the new code no longer supports multiple simultaneous SHTs of different type (i.e. having different directions or different spins). Simultaneous transforms of identical type are still available, however.

As a fortunate consequence of this slight reduction in functionality, the user interface could be simplified dramatically, which is especially helpful when interfacing the library with other programming languages.

Since backward compatibility is lost, the new name libsharp was chosen for the resulting code, as a shorthand for "Spherical HARmonic Package".

The decision to develop libsharp instead of simply using shtns was taken for various reasons: shtns does not support spin SHTs or allow MPI parallelisation, it requires more main memory than libsharp, which can be problematic for high-resolution runs, and it relies on the presence of the FFTW library. Also, shtns uses a syntax for expressing SIMD operations, which is currently only supported by the gcc and clang compilers, thereby limiting its portability at least for the vectorised version. Finally, libsharp has support for partial spherical coverage and a wide variety of spherical grids, including Gauss-Legendre, ECP, and HEALPix.

2. Problem definition

This section contains a quick recapitulation of equations presented in Reinecke (2011), for easier reference.

We assume a spherical grid with N_ϑ iso-latitude rings (indexed by y). Each of these in turn consists of $N_{\varphi,y}$ pixels (indexed by x), which are equally spaced in φ, the azimuth of the first ring pixel being $\varphi_{0,y}$.

A continuous spin-s function defined on the sphere with a spectral band limit of $l_{\max}$ can be represented either as a set of spherical harmonic coefficients ${}_sa_{lm}$, or a set of pixels $\boldsymbol{p}_{xy}$. These

two representations are connected by spherical harmonic synthesis (or backward SHT):

$$\boldsymbol{p}_{xy} = \sum_{m=-l_{\max}}^{l_{\max}} \sum_{l=|m|}^{l_{\max}} {}_s a_{lm}\, {}_s\lambda_{lm}(\boldsymbol{\vartheta}_y) \exp\left(\mathrm{i}m\varphi_{0,y} + \frac{2\pi\mathrm{i}mx}{N_{\varphi,y}}\right) \tag{1}$$

and spherical harmonic analysis (or forward SHT):

$${}_s\hat{a}_{lm} = \sum_{y=0}^{N_\vartheta-1} \sum_{x=0}^{N_{\varphi,y}-1} \boldsymbol{p}_{xy}\, \boldsymbol{w}_y\, {}_s\lambda_{lm}(\boldsymbol{\vartheta}_y) \exp\left(-\mathrm{i}m\varphi_{0,y} - \frac{2\pi\mathrm{i}mx}{N_{\varphi,y}}\right), \tag{2}$$

where ${}_s\lambda_{lm}(\vartheta) := {}_sY_{lm}(\vartheta, 0)$ and $\boldsymbol{w}_y$ are quadrature weights.

Both transforms can be subdivided into two stages:

$$\boldsymbol{p}_{xy} = \sum_{m=-l_{\max}}^{l_{\max}} F_{m,y} \exp\left(\mathrm{i}m\varphi_{0,y} + \frac{2\pi\mathrm{i}mx}{N_{\varphi,y}}\right) \quad \text{with} \tag{3}$$

$$F_{m,y} := \sum_{l=|m|}^{l_{\max}} {}_s a_{lm}\, {}_s\lambda_{lm}(\boldsymbol{\vartheta}_y), \quad \text{and} \tag{4}$$

$${}_s\hat{a}_{lm} = \sum_{y=0}^{N_\vartheta-1} G_{m,y}\, {}_s\lambda_{lm}(\boldsymbol{\vartheta}_y) \quad \text{with} \tag{5}$$

$$G_{m,y} := \boldsymbol{w}_y \sum_{x=0}^{N_{\varphi,y}-1} \boldsymbol{p}_{xy} \exp\left(-\mathrm{i}m\varphi_{0,y} - \frac{2\pi\mathrm{i}mx}{N_{\varphi,y}}\right). \tag{6}$$

Equations (3) and (6) can be computed using fast Fourier transforms (FFTs), while Eqs. (4) and (5), which represent the bulk of the computational load, are the main target for optimised implementation in `libsharp`.

3. Technical improvements

3.1. General remarks

As the implementation language for the new library, ISO C99 was chosen. This version of the C language standard is more flexible than the C89 one adopted for `libpsht` and has gained ubiquitous compiler support by now. Most notably, C99 allows definition of new variables anywhere in the code, which improves readability and eliminates a common source of programming mistakes. It also provides native data types for complex numbers, which allows for a more concise notation in many places. However, special care must be taken not to make use of these data types in the library's public interface, since this would prevent interoperability with C++ codes (because C++ has a different approach to complex number support). Fortunately, this drawback can be worked around fairly easily.

A new approach was required for dealing with the growing variety of instruction sets for arithmetic operations, such as traditional scalar instructions, SSE2 and AVX. Rewriting the library core for each of these alternatives would be cumbersome and error-prone. Instead we introduced the concept of a generic "vector" type containing a number of double-precision IEEE values and defined a set of abstract operations (basic arithmetics, negation, absolute value, multiply-accumulate, min/max, comparison, masking and blending) for this type. Depending on the concrete instruction set used when compiling the code, these operations are then expressed by means of the appropriate operators and intrinsic function calls. The only constraint on the number of values in the vector type is that it has to be a multiple of the underlying instruction set's native vector length (1 for scalar arithmetic, 2 for SSE2, 4 for AVX).

Using this technique, adding support for new vector instruction sets is straightforward and carries little risk of breaking existing code. As a concrete example, support for the FMA4 instructions present in AMD's Bulldozer CPUs was added and successfully tested in less than an hour.

3.2. Improved loop structure

After publication of SHT implementations, which perform significantly better than `libpsht`, especially for $s = 0$ transforms (Seljebotn 2012; Schaeffer 2013), it became obvious that some bottleneck must be present in `libpsht`'s implementation. This was identified with `libpsht`'s approach of first computing a whole l-vector of ${}_s\lambda_{lm}(\vartheta)$ in one go, storing it to main memory, and afterwards re-reading it sequentially whenever needed. While the l-vectors are small enough to fit into the CPU's Level-1 cache, the store and load operations nevertheless caused some latency. For $s = 0$ transforms with their comparatively low arithmetic operation count (compared to the amount of memory accesses), this latency could not be hidden behind floating point operations and so resulted in a slow-down. This is the most likely explanation for the observation that `libpsht`'s $s = 0$ SHTs have a much lower FLOP rate compared to those with $s \neq 0$.

It is possible to avoid the store/load overhead for the ${}_s\lambda_{lm}(\vartheta)$ by applying each value immediately after it has been computed, and discarding it as soon as it is not needed any more. This approach is reflected in the loop structure shown in Figs. 1 and 2, which differs from the one in Reinecke (2011) mainly by the fusion of the central loops over l.

In this context another question must be addressed: the loops marked as "SSE/AVX" in both figures are meant to be executed in "blocks", i.e. by processing several y indices simultaneously. The block size is equivalent to the size of the generic vector type described in Sect. 3.1. The best value for this parameter depends on hardware characteristics of the underlying computer and therefore cannot be determined a priori. `Libsharp` always uses a multiple of the system's natural vector length and estimates the best value by running quick measurements whenever a specific SHT is invoked for the first time. This auto-tuning step approximately takes a tenth of a wall-clock second.

Due to the changed central loop of the SHT implementation, it is no longer straightforward to support multiple simultaneous transforms with differing spins and/or directions, as `libpsht` did – this would lead to a combinatorial explosion of loop bodies that have to be implemented. Consequently, `libsharp`, while still supporting simultaneous SHTs, restricts them to have the same spin and direction. Fortunately, this is a very common case in many application scenarios.

3.3. Polar optimisation

As previously mentioned in Reinecke (2011), much CPU time can be saved by simply not computing terms in Eqs. (4) and (5) for which ${}_s\lambda_{lm}(\vartheta)$ is so small that their contribution to the result lies well below the numerical accuracy. Since this situation occurs for rings lying close to the poles and high values of m, Schaeffer (2013) referred to it as "polar optimisation".

To determine which terms can be omitted, `libsharp` uses the approach described in Prézeau & Reinecke (2010). In short, all terms for which

$$\sqrt{m^2 + s^2 - 2ms\cos\vartheta} - l_{\max}\sin\vartheta > T \tag{7}$$

```
for b = <all submaps or "blocks">
  for m = [0;mmax]                          // OpenMP
    for l = [m;lmax]
      precompute recursion coefficients
    end l
    for y = <all rings in submap b>         // SSE/AVX
      for l = [m;lmax]
        compute s_lambda_lm(theta_y)
        for j = <all jobs>
          accumulate F(m,theta_y,j)
        end j
      end l
    end y
  end m

  for y = <all rings in submap b>           // OpenMP
    for j = <all jobs>
      compute map(x,y,j) using FFT
    end j
  end y
end b
```

Fig. 1. Schematic loop structure of `libsharp`'s shared-memory synthesis code.

```
for b = <all submaps or "blocks">
  for y = <all rings in submap b>           // OpenMP
    for j = <all jobs>
      compute G(m,theta_y,j) using FFT
    end j
  end y

  for m = [0;mmax]                          // OpenMP
    for l = [m;lmax]
      precompute recursion coefficients
    end l
    for y = <all rings in submap b>         // SSE/AVX
      for l = [m;lmax]
        compute s_lambda_lm(theta_y)
        for j = <all jobs>
          compute contribution to a_lm(j)
        end j
      end l
    end y
  end m
end b
```

Fig. 2. Schematic loop structure of `libsharp`'s shared-memory analysis code.

are skipped. The parameter T is tunable and determines the overall accuracy of the result. `Libsharp` models it as

$$T = \max(100, 0.01 l_{\max}), \tag{8}$$

which has been verified to produce results equivalent to those of SHTs without polar optimisation.

4. Added functionality

4.1. SHTs with arbitrary spin

While the most widely used SHTs in cosmology are performed on quantities of spins 0 and 2 (i.e. sky maps of Stokes I and Q/U parameters), there is also a need for transforms at other spins. Lensing computations require SHTs of spin-1 and spin-3 quantities (see, e.g., Lewis 2005). The most important motivation for high-spin SHTs, however, are all-sky convolution codes (e.g. Wandelt & Górski 2001; Prézeau & Reinecke 2010) and deconvolution map-makers (e.g. Keihänen & Reinecke 2012). The computational cost of these algorithms is dominated by calculating expressions of the form

$$R_{mk}(\vartheta) = \sum_{l=\max(|m|,|k|)}^{l_{\max}} a_{lm} b^*_{lk} d^l_{mk}(\vartheta), \tag{9}$$

where a and b denote two sets of spherical harmonic coefficients (typically of the sky and a beam pattern) and d are the Wigner d matrix elements. These expressions can be interpreted and solved efficiently as a set of (slightly modified) SHTs with spins ranging from 0 to $k_{\max} \le l_{\max}$, which in today's applications can take on values much higher than 2.

As was discussed in Reinecke (2011), the algorithms used by `libpsht` for spin-1 and spin-2 SHTs become inefficient and inaccurate for higher spins. To support such transforms in `libsharp`, another approach was therefore implemented, which is based on the recursion for Wigner d matrix elements presented in Prézeau & Reinecke (2010).

Generally, the spin-weighted spherical harmonics are related to the Wigner d matrix elements via

$${}_s\lambda_{lm}(\vartheta) = (-1)^m \sqrt{\frac{2l+1}{4\pi}} d^l_{-ms}(\vartheta) \tag{10}$$

(Goldberg et al. 1967). It is possible to compute the $d^l_{mm'}(\vartheta)$ using a three-term recursion in l very similar to that for the scalar $Y_{lm}(\vartheta)$:

$$\left[\cos\vartheta - \frac{mm'}{l(l+1)}\right] d^l_{mm'}(\vartheta) = \frac{\sqrt{(l^2-m^2)(l^2-m'^2)}}{l(2l+1)} d^{l-1}_{mm'}(\vartheta) + \frac{\sqrt{[(l+1)^2-m^2][(l+1)^2-m'^2]}}{(l+1)(2l+1)} d^{l+1}_{mm'}(\vartheta) \tag{11}$$

(Kostelec & Rockmore 2008). The terms depending only on l, m, and m' can be re-used for different colatitudes, so that the real cost of a recursion step is two additions and three multiplications.

In contrast to the statements made in McEwen & Wiaux (2011), this recursion is numerically stable when performed in the direction of increasing l; see, e.g., Sect. 5.1.2 for a practical confirmation. It is necessary, however, to use a digital floating-point representation with enhanced exponent range to avoid underflow during the recursion, as is discussed in some detail in Prézeau & Reinecke (2010).

4.2. Distributed memory parallelisation

When considering that, in current research, the required band limit for SHTs practically never exceeds $l_{\max} = 10^4$, it seems at first glance unnecessary to provide an implementation supporting multiple nodes. Such SHTs fit easily into the memory of a single typical compute node and are carried out within a few seconds of wall clock time. The need for additional parallelisation becomes apparent, however, as soon as the SHT is no longer considered in isolation, but as a (potentially small) part of another algorithm, which is `libsharp`'s main usage scenario. In such a situation, large amounts of memory may be occupied

```
for m = <all local m>                // OpenMP
  for l = [m;lmax]
    precompute recursion coefficients
  end l
  for y = <all rings in the map>     // SSE/AVX
    for l = [m;lmax]
      compute s_lambda_lm(theta_y)
      for j = <all jobs>
        accumulate F(m,theta_y,j)
      end j
    end l
  end y
end m

rearrange F(m,theta_y,j) among tasks // MPI

for y = <all local rings>            // OpenMP
  for j = <all jobs>
    compute map(x,y,j) using FFT
  end j
end y
```

Fig. 3. Schematic loop structure of `libsharp`'s distributed-memory synthesis code.

```
for y = <all local rings>            // OpenMP
  for j = <all jobs>
    compute G(m,theta_y,j) using FFT
  end j
end y

rearrange G(m,theta_y,j) among tasks // MPI

for m = <all local m>                // OpenMP
  for l = [m;lmax]
    precompute recursion coefficients
  end l
  for y = <all rings in the map>     // SSE/AVX
    for l = [m;lmax]
      compute s_lambda_lm(theta_y)
      for j = <all jobs>
        compute contribution to a_lm(j)
      end j
    end l
  end y
end m
```

Fig. 4. Schematic loop structure of `libsharp`'s distributed-memory analysis code.

by data sets unrelated to the SHT, therefore requiring distribution over multiple nodes. Moreover, there is sometimes the need for very many SHTs in sequence, e.g. if they are part of a sampling process or an iterative solver. Here, the parallelisation to a very large number of CPUs may be the only way of reducing the time-to-solution to acceptable levels. Illustrative examples for this are the Commander code (Eriksen et al. 2008) and the `artDeco` deconvolution mapmaker (Keihänen & Reinecke 2012); for the processing of high-resolution *Planck* data, the latter is expected to require over 100GB of memory and several hundred CPU cores.

`Libsharp` provides an interface that allows collective execution of SHTs on multiple machines with distributed memory. It makes use of the MPI[1] interface to perform the necessary interprocess communication.

In contrast to the standard, shared-memory algorithms, it is the responsibility of the library user to distribute map data and a_{lm} over the individual computers in a way that ensures proper load balancing. A very straightforward and reliable way to achieve this is a "round robin" strategy: assuming N computing nodes, the map is distributed such that node i hosts the map rings i, $i + N$, $i + 2N$, etc. (and their counterparts on the other hemisphere). Similarly, for the spherical harmonic coefficients, node i would hold all a_{lm} for $m = i$, $i + N$, $i + 2N$, etc. Other efficient distribution strategies do of course exist and may be advantageous, depending on the circumstances under which `libsharp` is called. The only requirement the library has is that the a_{lm} are distributed by m and that the map is distributed by rings, as described in Figs. 3 and 4.

The SHT algorithm for distributed memory architectures is analogous to the one used in the S^{2}HAT package[2] and first published in Szydlarski et al. (2013); its structure is sketched in Figs. 3 and 4. In addition to the S^{2}HAT implementation, the SHT will be broken down into smaller chunks if the average number of map rings per MPI task exceeds a certain threshold. This is analogous to the use of chunks in the scalar and OpenMP-parallel implementations and reduces the memory overhead caused by temporary variables.

It should be noted that `libsharp` supports hybrid MPI and OpenMP parallelisation, which allows, e.g., running an SHT on eight nodes with four CPU cores each, by specifying eight MPI tasks, each of them consisting of four OpenMP threads. In general, OpenMP should be preferred over MPI whenever shared memory is available (i.e. at the computing node level), since the OpenMP algorithms contain dynamic load balancing and have a smaller communication overhead.

4.3. Map synthesis of first derivatives

Generating maps of first derivatives from a set of a_{lm} is closely related to performing an SHT of spin 1. A specialised SHT mode was added to `libsharp` for this purpose; it takes as input a set of spin-0 a_{lm} and produces two maps containing $\partial f/\partial\vartheta$ and $\partial f/(\partial\varphi \sin\vartheta)$, respectively.

4.4. Support for additional spherical grids

Direct support for certain classes of spherical grids has been extended in comparison to `libpsht`; these additions are listed below in detail. It must be stressed, however, that `libsharp` can – very much as `libpsht` does – perform SHTs on any iso-latitude grid with equidistant pixels on each ring. This very general class of pixelisations includes, e.g., certain types of partial spherical coverage. For these general grids, however, the user is responsible for providing correct quadrature weights when performing a spherical harmonic analysis.

[1] http://en.wikipedia.org/wiki/Message_Passing_Interface

[2] http://code.google.com/p/s2hat-library/

4.4.1. Extended support for ECP grids

`Libpsht` provides explicit support for HEALPix grids, Gauss-Legendre grids, and a subset of equidistant cylindrical projection (ECP) grids. The latter are limited to an even number of rings at the colatitudes

$$\vartheta_n = \frac{(n+0.5)\pi}{N}, \quad n \in [0; N-1]. \tag{12}$$

The associated quadrature weights are given by Fejér's first rule (Fejér 1933; Gautschi 1967).

`Libsharp` extends ECP grid support to allow even and odd numbers of rings, as well as the colatitude distributions

$$\vartheta_n = \frac{n\pi}{N}, \quad n \in [1; N-1] \tag{13}$$

(corresponding to Fejér's second rule), and

$$\vartheta_n = \frac{n\pi}{N}, \quad n \in [0; N] \tag{14}$$

(corresponding to Clenshaw-Curtis quadrature). This last pixelisation is identical to the one adopted in Huffenberger & Wandelt (2010).

Accurate computation of the quadrature weights for these pixelisations is nontrivial; `libsharp` adopts the FFT-based approach described in Waldvogel (2006) for this purpose.

4.4.2. Reduced Gauss-Legendre grid

The polar optimisation described in Sect. 3.3 implies that it is possible to reduce the number of pixels per ring below the theoretically required value of $2l_{\max} + 1$ close to the poles. Equation (7) can be solved for m (at a given s, $l_{\max}$ and ϑ), and using $2m + 1$ equidistant pixels in the corresponding map ring results in a pixelisation that can represent a band-limited function up to the desired precision, although it is no longer exact in a mathematical sense. If this number is further increased to the next number composed entirely of small prime factors (2, 3, and 5 are used in `libsharp`'s case), this has the additional advantage of allowing very efficient FFTs.

`Libsharp` supports this pixel reduction technique in the form of a thinned-out Gauss-Legendre grid. At moderate to high resolutions ($l_{\max} > 1000$), more than 30% of pixels can be saved, which can be significant in various applications.

It should be noted that working with reduced Gauss-Legendre grids, while saving considerable amounts of memory, does not change SHT execution times significantly; all potential savings are already taken into account, for all grids, by `libsharp`'s implementation of polar optimisation.

4.5. Adjoint and real SHTs

Since Eq. (1) is a linear transform, we can introduce the notation

$$\boldsymbol{p} = \mathbf{Y}\boldsymbol{a} \tag{15}$$

for a vector $\boldsymbol{a}$ of spherical harmonic coefficients and corresponding vector $\boldsymbol{p}$ of pixels. Similarly, one can write Eq. (2) as

$$\boldsymbol{a} = \mathbf{Y}^\dagger \mathbf{W} \boldsymbol{p}, \tag{16}$$

where $\mathbf{W}$ is a diagonal matrix of quadrature weights. When including SHTs as operators in linear systems, one will often need the *adjoint spherical harmonic synthesis*, $\mathbf{Y}^\dagger$, and the *adjoint spherical harmonic analysis*, $\mathbf{WY}$. For instance, if $\boldsymbol{a}$ is a random vector with covariance matrix $\mathbf{C}$ in the spherical harmonic domain, then its pixel representation $\mathbf{Y}\boldsymbol{a}$ has the covariance matrix $\mathbf{YCY}^\dagger$. Multiplication by this matrix requires the use of the adjoint synthesis, which corresponds to analysis with a different choice of weights. `Libsharp` includes routines for adjoint SHTs, which is more user-friendly than having to compensate for the wrong choice of weights in user code, and also avoids an extra pass over the data.

For linear algebra computations, the vector $\boldsymbol{a}$ must also include a_{lm} with $m < 0$, even if `libsharp` will only compute the coefficients for $m \geq 0$. The use of the *real spherical harmonics* convention is a convenient way to include negative m without increasing the computational workload by duplicating all coefficients. For the definition we refer to the appendix of de Oliveira-Costa et al. (2004). The convention is supported directly in `libsharp`, although with a restriction in the storage scheme: The coefficients for $m < 0$ must be stored in the same locations as the corresponding imaginary parts of the complex coefficients, so that the pattern in memory is $[a_{l,m}, a_{l,-m}]$.

5. Evaluation

Most tests were performed on the SuperMUC Petascale System located at the Leibniz-Rechenzentrum Garching. This system consists of nodes containing 32GB of main memory and 16 Intel Xeon E5-2680 cores running at 2.7GHz. The exception is the comparison with Wavemoth, which was performed on the Abel cluster at the University of Oslo on very similar hardware; Xeon E5-2670 at 2.6 GHz.

The code was compiled with `gcc` version 4.7.2. The Intel compiler (version 12.1.6) was also tested, but produced slightly inferior code.

Except where noted otherwise, test calculations were performed using the reduced Gauss-Legendre grid (see Sect. 4.4.2) to represent spherical map data. This was done for the pragmatic purpose of minimising the tests' memory usage, which allowed going to higher band limits in some cases, as well as to demonstrate the viability of this pixelisation.

The band limits adopted for the tests all obey $l_{\max} = 2^n - 1$ with $n \in \mathbb{N}$ (except for those presented in Sect. 5.2.2). This is done in analogy to most other papers on the subject, but leads to some unfamiliar numbers especially at very high $l_{\max}$.

The number of cores used for any particular run always is a power of 2.

5.1. Accuracy tests

5.1.1. Comparison with other implementations

The numerical equivalence of `libsharp`'s SHTs to existing implementations was verified by running spherical harmonic synthesis transforms on a Gauss-Legendre grid at $l_{\max} = 50$ and spins 0, 1, and 2 with both `libsharp` and `libpsht`, and comparing the results. The differences of the results lay well within the expected levels of numerical errors caused by the finite precision of IEEE numbers. Spherical harmonic analysis is implicitly tested by the experiments in the following sections.

5.1.2. Evaluation of SHT pairs

To test the accuracy of `libsharp`'s transforms, sets of spin = 0 a_{lm} coefficients were generated by setting their real and imaginary parts to numbers drawn from a uniform random

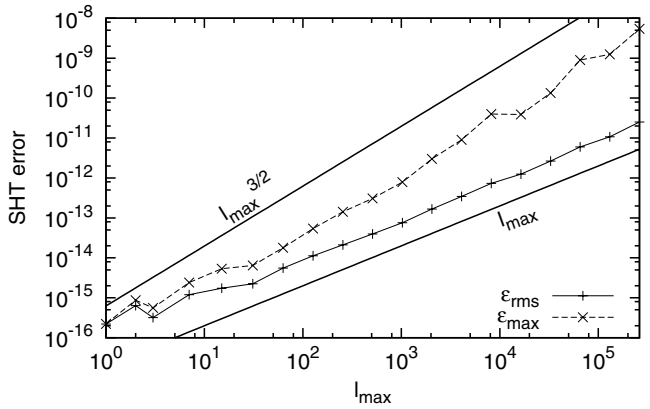


Fig. 5. Maximum and rms errors for inverse/forward spin = 0 SHT pairs at different l_{max}.

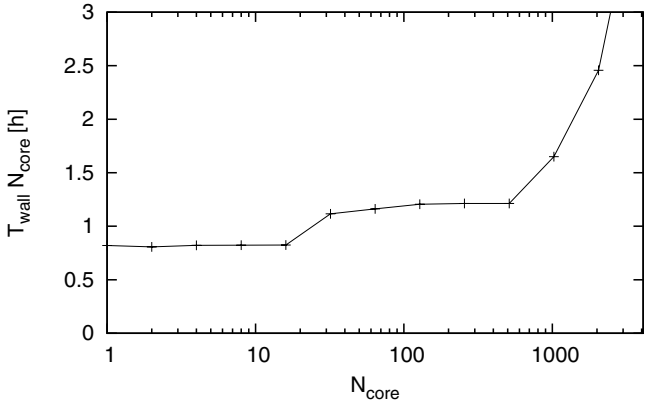


Fig. 6. Strong-scaling scenario: accumulated wall-clock time for a spin = 2 SHT pair with $l_{max} = 16\,383$ run on various numbers of cores.

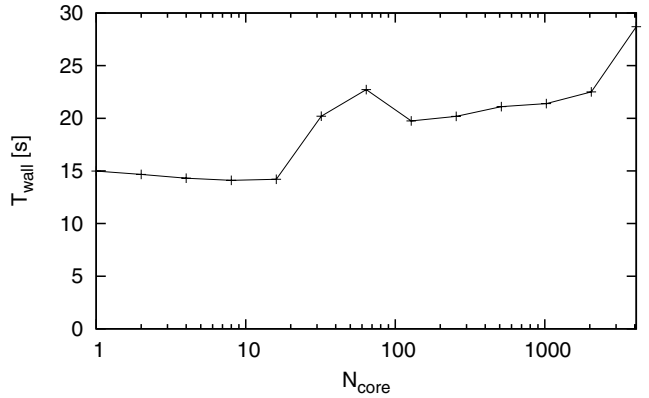


Fig. 7. Weak-scaling scenario: wall-clock time for a spin = 0 SHT pair run on various numbers of cores, with constant amount of work per core ($l_{max}(N) = 4096N^{1/3} - 1$).

distribution in the range $[-1; 1[$ (with exception of the imaginary parts for $m = 0$, which have to be zero for symmetry reasons). This data set was transformed onto a reduced Gauss-Legendre grid and back to spherical harmonic space again, resulting in $\hat{a}_{lm}$.

The rms and maximum errors of this inverse/forward transform pair can be written as

$$\varepsilon_{\rm rms} := \sqrt{\frac{\sum_{lm} |{}_s a_{lm} - {}_s\hat{a}_{lm}|^2}{\sum_{lm} |{}_s a_{lm}|^2}} \quad \text{and} \tag{17}$$

$$\varepsilon_{\rm max} := \max_{lm} |{}_s a_{lm} - {}_s\hat{a}_{lm}| \,. \tag{18}$$

Figure 5 shows the measured errors for a wide range of band limits. As expected, the numbers are close to the accuracy limit of double precision IEEE numbers for low l_{max}; rms errors increase roughly linearly with the band limit, while the maximum error seems to exhibit an $l_{max}^{3/2}$ scaling. Even at $l_{max} = 262\,143$ (which is extremely high compared to values typically required in cosmology), the errors are still negligible compared with the uncertainties in the input data in today's experiments.

Analogous experiments were performed for spins 2 and 37, with very similar results (not shown).

5.2. Performance tests

Determining reliable execution times for SHTs is nontrivial at low band limits, since intermittent operating system activity can significantly distort the measurement of short time scales. All `libsharp` timings shown in the following sections were obtained using the following procedure: the SHT pair in question is executed repeatedly until the accumulated wall-clock time exceeds 2 s. Then the shortest measured wall-clock time for synthesis and analysis is selected from the available set.

5.2.1. Strong-scaling test

To assess strong-scaling behaviour (i.e. run time scaling for a given problem with fixed total workload), a spin = 2 SHT with $l_{max} = 16\,383$ was carried out with differing degrees of parallelisation. The accumulated wall-clock time of these transforms (synthesis + analysis) is shown in Fig. 6. It is evident that the scaling is nearly ideal up to 16 cores, which implies that parallelisation overhead is negligible in this range. Beyond 16 cores, MPI communication has to be used for inter-node communication, and this most likely accounts for the sudden jump in accumulated time. At even higher core counts, linear scaling is again reached, although with a poorer proportionality factor than in the intra-node case. Finally, for 1024 and more cores, the communication time dominates the actual computation, and scalability is lost.

5.2.2. Weak-scaling test

Weak-scaling behaviour of the algorithm is investigated by choosing problem sizes that keep the total work *per core* constant, in contrast to a fixed *total* workload. Assuming an SHT complexity of $O(l_{max}^3)$, the band limits were derived from the employed number of cores N via $l_{max}(N) = 4096N^{1/3} - 1$. The results are shown in Fig. 7. Ideal scaling corresponds to a horizontal line. Again, the transition from one to several computing nodes degrades performance, whereas scaling on a single node, as well as in the multi-node range, is very good. By keeping the amount of work per core constant, the breakdown of scalability is shifted to 4096 cores, compared with 1024 in the strong-scaling test.

It is interesting to note that the scaling within a single node is actually slightly superlinear; this is most likely because in this setup, the amount of memory per core decreases with increasing problem size, which in turn can improve the amount of cache re-use and reduce memory bandwidth per core.

5.2.3. General scaling and efficiency

The preceding two sections did not cover cases with small SHTs. This scenario is interesting, however, since in the limit

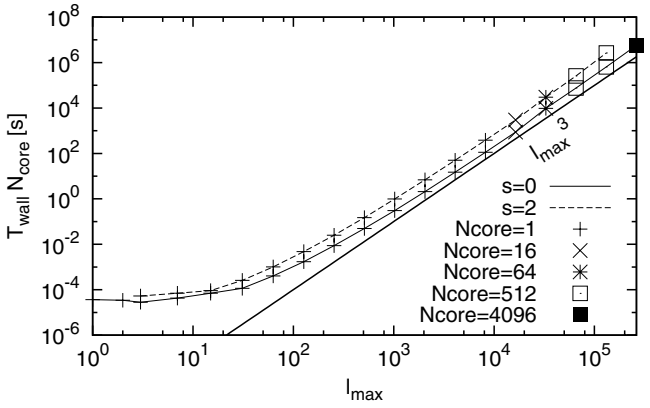


Fig. 8. Accumulated wall-clock time for spin = 0 and spin = 2 SHT pairs at a wide range of different band limits. For every run the number of cores was chosen sufficiently small to keep parallelisation overhead low.

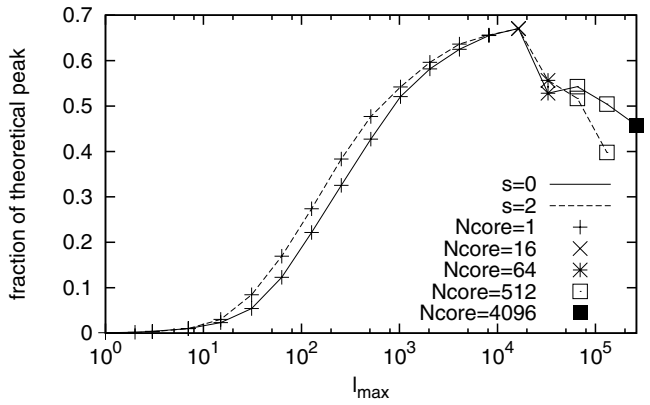


Fig. 9. Fraction of theoretical peak-performance reached by various SHT pairs. For every run the number of cores was chosen sufficiently small to keep parallelisation overhead low.

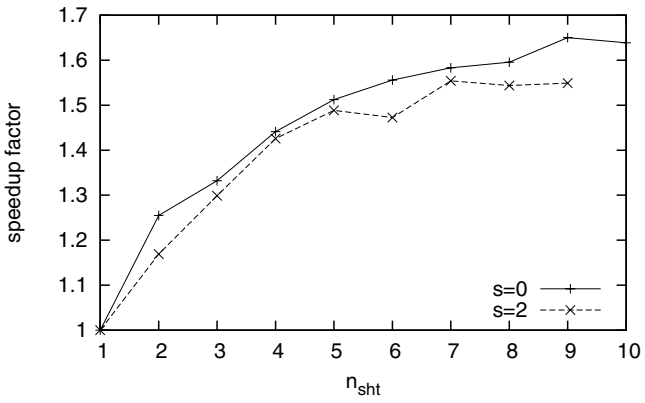


Fig. 10. Relative speed-up when performing several SHTs simultaneously, compared with sequential execution. The SHT had a band limit of $l_{max} = 8191$.

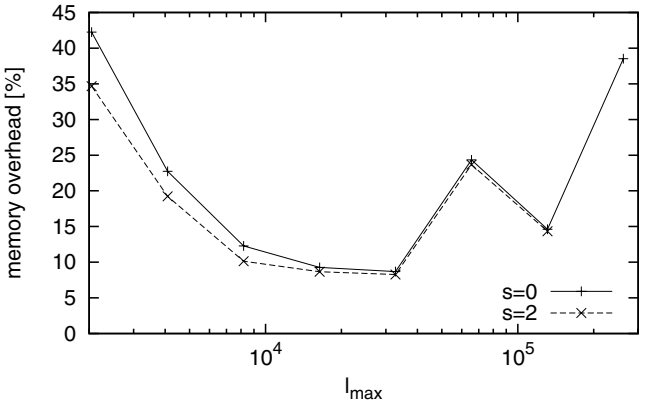


Fig. 11. Relative memory overhead, i.e. the fraction of total memory that is not occupied by input and output data of the SHT. For low l_{max} this is dominated by the program binary, for high l_{max} by temporary arrays.

of small l_{max} those components of the SHT implementation with complexities lower than $O(l_{max}^3)$ (like the FFT steps of Eqs. (3) and (6)) may begin to dominate execution time. Figure 8 shows the total wall-clock time for SHT pairs over a very wide range of band limits; to minimise the impact of communication, the degree of parallelisation was kept as low as possible for the runs in question. As expected, the l_{max}^3 scaling is a very good model for the execution times at $l_{max} \geq 511$. Below this limit, the FFTs, precomputations for the spherical harmonic recursion, memory copy operations and other parts of the code begin to dominate.

In analogy to one of the tests described in Reinecke (2011), we computed a lower limit for the number of executed floating-point operations per second in `libsharp`'s SHTs and compared the result with the theoretical peak performance achievable on the given hardware, which is eight operations per clock cycle (four additions and four multiplications) or 21.6 GFlops/s per core. Figue 9 shows the results. In contrast to `libpsht`, which reached approximately 22% for $s = 0$ and 43% for $s = 2$, both scalar and tensor harmonic transforms exhibit very similar performance levels and almost reach 70% of theoretical peak in the most favourable regime, thanks to the changed structure of the inner loops. For the l_{max} range that is typically required in cosmological applications, performance exceeds 50% (even when MPI is used), which is very high for a practically useful algorithm on this kind of computer architecture.

5.2.4. Multiple simultaneous SHTs

The computation of the ${}_s\lambda_{lm}(\vartheta)$ coefficients accounts for roughly half the arithmetic operations in an SHT. If several SHTs with identical grid geometry and band limit are computed simultaneously, it is possible to re-use these coefficients, thereby reducing the overall operation count. Figure 10 shows the speed-ups compared to sequential execution for various scenarios, which increase with the number of transforms and reach saturation around a factor of 1.6. This value is lower than the naïvely expected asymptotic factor of 2 (corresponding to avoiding half of the arithmetic operations), since the changed algorithm requires more memory transfers between Level-1 cache and CPU registers and therefore operates at a lower percentage of theoretical peak performance. Nevertheless, running SHTs simultaneously is evidently beneficial and should be used whenever possible.

5.2.5. Memory overhead

Especially at high band limits, it is important that the SHT library does not consume a large amount of main memory, to avoid memory exhaustion and subsequent swapping or code crashes. `Libsharp` is designed with the goal to keep the size of its auxiliary data structures much lower than the combined size of any SHT's input and output arrays. A measurement is shown in Fig. 11. Below the lowest shown band limit of 2047, memory

Table 1. Performance comparison with other implementations at $l_{max} = 2047$, $n_{core} = 1$.

Code	Version	Grid	Spin	n_{SHT}	R_{AVX}	R_{SSE2}	R_{scalar}
`shtns`	2.31	Gauss-Legendre	0	1	0.84	0.88	0.91
Wavemoth (brute-force)	Nov. 2011	HEALPix ($N_{side} = 1024$)	0	1	1.63	0.98	–
″	″	″	0	5	1.59	1.09	–
Wavemoth (butterfly)	Nov. 2011	HEALPix ($N_{side} = 1024$)	0	5	0.96	0.66	–
`libpsht`	Jan. 2011	Gauss-Legendre	0	1	4.06	2.30	2.30
″	″	″	0	5	2.66	1.75	1.62
″	″	″	2	1	2.50	1.48	1.20
″	″	″	2	5	2.15	1.44	1.08
`spinsfast`	r104	ECP (Clenshaw-Curtis)	0	1	57.04	32.12	15.31
″	″	″	0	5	28.39	18.72	9.38
″	″	″	2	1	16.99	10.20	4.73
″	″	″	2	5	8.60	5.66	2.56
SSHT	1.0b1	MW sampling theorem	0	1	20.91	15.60	9.46
″	″	″	2	1	13.40	9.29	4.99
S^{2}HAT	2.55beta	HEALPix ($N_{side} = 1024$)	0	1	12.33	7.33	3.60
Glesp	2	Gauss-Legendre	0	1	55.32	31.26	14.95

Notes. All tests had a band limit of $l_{max} = 2047$ and were carried out on a single core. The grids used by `libsharp` and the respective comparison code were identical in each run. R_{AVX} denotes the quotient of wall-clock times for the respective code and `libsharp` in the presence of the AVX instruction set, R_{SSE2} is the quotient when SSE2 (but not AVX) is supported, and R_{scalar} was measured with both SSE2 and AVX disabled. The `libsharp` support for the MW sampling theorem used for the SSHT comparisons is experimental. For Wavemoth, butterfly matrix compression can optionally be enabled. In the benchmark given we requested an accuracy of 10^{-4}, which led to an extra requirement of 4 GB of precomputed data in memory. Note that when running on a single core, Wavemoth is at an advantage compared to the normal situation where the memory bus is shared between multiple cores.

overhead quickly climbs to almost 100%, since in this regime memory consumption is dominated by the executable and the constant overhead of the communication libraries, which on the testing machine amounts to approximately 50MB. In the important range ($l_{max} \geq 2047$), memory overhead lies below 45%.

5.2.6. Comparison with existing implementations

Table 1 shows a performance comparison of synthesis/analysis SHT pairs between `libsharp` and various other SHT implementations. In addition to the already mentioned `shtns`, Wavemoth, S^{2}HAT and `libpsht` codes, we also included `spinsfast` (Huffenberger & Wandelt 2010), SSHT (McEwen & Wiaux 2011) and Glesp (Doroshkevich et al. 2005) in the comparison. All computations shared a common band limit of 2047 and were executed on a single core, since the corresponding SHTs are supported by all libraries and are very likely carried out with a comparatively high efficiency by all of them. The large overall number of possible parameters (l_{max}, spin, number of simultaneous transforms, degree and kind of parallelisation, choice of grid, etc.) prevented a truly comprehensive study.

Overall, `libsharp`'s performance is very satisfactory and exhibits speed-ups of more than an order of magnitude in several cases. The table also demonstrates `libsharp`'s flexibility, since it supports all of the other codes' "native" grid geometries, which is required for direct comparisons.

The three last columns list time ratios measured under different assumptions: R_{AVX} reflects values that can be expected on modern (2012 and later) AMD/Intel CPUs supporting AVX, R_{SSE2} applies to older (2001 and later) CPUs with the SSE2 instruction set. R_{scalar} should be used for CPUs from other vendors like IBM or ARM, since `libsharp` does not yet support vectorisation for these architectures.

Figure 12 shows the relative performance of identical SHT pairs on a full Gauss-Legendre grid with $s = 0$ for `libsharp` and `shtns`. For these measurements the benchmarking code delivered with `shtns` was adjusted to measure

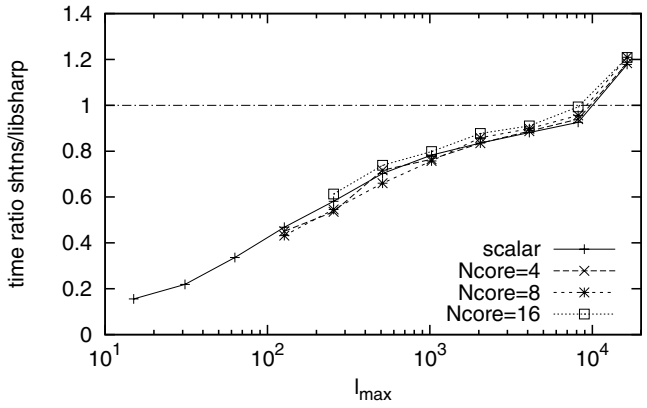


Fig. 12. Performance comparison between `libsharp` and `shtns` for varying l_{max} and number of OpenMP threads. Note that reduced autotuning was used for `shtns` at l_{max}=16 383 (see text).

SHT times in a similar fashion as was described above. The plotted quantity is `shtns` wall-clock time divided by `libsharp` wall-clock time for varying l_{max} and number of OpenMP threads. It is evident that `shtns` has a significant advantage for small band limits (almost an order of magnitude) and maintains a slight edge up to $l_{max} = 8191$. It must be noted, however, that the measured times do not include the overhead for auto-tuning and necessary precalculations, which in the case of `shtns` are about an order of magnitude more expensive than the SHTs themselves. As a consequence, its performance advantage only pays off if many identical SHT operations are performed within one run. The origin of `shtns`'s performance advantage has not been studied in depth; however, a quick analysis shows that the measured time differences scale roughly like l_{max}^2, so the following explanations are likely candidates:

- `libsharp` performs all of its precomputations as part of the time-measured SHT;

- `libsharp`'s flexibility with regard to pixelisation and storage arrangement of input and output data requires some additional copy operations;
- at low band limits the inferior performance of `libsharp`'s FFT implementation has a noticeable impact on overall run times.

The relative performance of both libraries is remarkably insensitive to the number of OpenMP threads; this indicates that the performance differences are located in parallel code regions as opposed to sequential ones.

For $l_{\max} = 16\,383$, the time required by the default `shtns` autotuner becomes very long (on the order of wall-clock hours), so that we decided to invoke it with an option for reduced tuning. It is likely a consequence of this missed optimisation that, at this band limit, `libsharp` is the better-performing code.

6. Conclusions

Judging from the benchmarks presented in the preceding section, the goals that were set for the `libsharp` library have been reached: it exceeds `libpsht` in terms of performance, supports recent developments in microprocessor technology, allows using distributed memory systems for a wider range of applications, and is slightly easier to use. On the developer side, the modular design of the code makes it much more straightforward to add support for new instruction sets and other functionality.

In some specific scenarios, especially for SHTs with comparatively low band limits, `libsharp` does not provide the best performance of all available implementations, but given its extreme flexibility concerning grid types and the memory layout of its input/output data, as well as its compactness (≈8000 lines of portable and easily maintainable source code without external dependencies), this compromise certainly seems acceptable.

The library has been successfully integrated into version 3.1 of the HEALPix C++ and Fortran packages. There also exists an experimental version of the SSHT[3] package with `libsharp` replacing the library's original SHT engine. `Libsharp` is also used as SHT engine in an upcoming version of the Python package NIFT [4] for signal inference (Selig et al. 2013). Recently, the total convolution code `conviqt` (Prézeau & Reinecke 2010), which is a central component of the *Planck* simulation pipeline (Reinecke et al. 2006), has been updated and is now based on `libsharp` SHTs. There are plans for a similar update of the `artDeco` deconvolution map maker (Keihänen & Reinecke 2012).

A potential future field of work is porting `libsharp` to Intel's "many integrated cores" architecture[5], once sufficient compiler support for this platform has been established. The hardware appears to be very well suited for running SHTs, and the porting by itself would provide a welcome test for the adaptability of the library's code design.

Acknowledgements. We thank our referee Nathanaël Schaeffer for his constructive remarks and especially for pointing out a missed optimisation opportunity in our `shtns` installation, which had a significant effect on some benchmark results. M.R. is supported by the German Aeronautics Center and Space Agency (DLR), under program 50-OP-0901, funded by the Federal Ministry of Economics and Technology. D.S.S. is supported by the European Research Council, grant StG2010-257080. The presented benchmarks were performed as project *pr89yi* at the Leibniz Computing Center Garching.

References

de Oliveira-Costa, A., Tegmark, M., Zaldarriaga, M., & Hamilton, A. 2004, Phys. Rev. D, 69, 063516
Doroshkevich, A. G., Naselsky, P. D., Verkhodanov, O. V., et al. 2005, Int. J. Mod. Phys. D, 14, 275
Eriksen, H. K., Jewell, J. B., Dickinson, C., et al. 2008, ApJ, 676, 10
Fejér, L. 1933, Mathematische Zeitschrift, 37, 287
Gautschi, W. 1967, SIAM J. Numerical Analysis, 4, 357
Goldberg, J. N., Macfarlane, A. J., Newman, E. T., Rohrlich, F., & Sudarshan, E. C. G. 1967, J. Math. Phys., 8, 2155
Huffenberger, K. M., & Wandelt, B. D. 2010, ApJS, 189, 255
Keihänen, E., & Reinecke, M. 2012, A&A, 548, A110
Kostelec, P., & Rockmore, D. 2008, J. Fourier Analysis and Applications, 14, 145
Lewis, A. 2005, Phys. Rev. D, 71, 083008
McEwen, J. D., & Wiaux, Y. 2011, IEEE Trans. Signal Proc., 59, 5876
Prézeau, G., & Reinecke, M. 2010, ApJS, 190, 267
Reinecke, M. 2011, A&A, 526, A108
Reinecke, M., Dolag, K., Hell, R., Bartelmann, M., & Enßlin, T. A. 2006, A&A, 445, 373
Schaeffer, N. 2013, Geochem. Geophys. Geosyst., 14
Selig, M., Bell, M. R., Junklewitz, H., et al. 2013, IEEE Trans. Inf. Theory, submitted [arXiv:1301.4499]
Seljebotn, D. S. 2012, ApJS, 199, 5
Szydlarski, M., Esterie, P., Falcou, J., Grigori, L., & Stompor, R. 2013, Concurrency and Computation: Practice and Experience
Waldvogel, J. 2006, BIT Numerical Mathematics, 46, 195
Wandelt, B. D., & Górski, K. M. 2001, Phys. Rev. D, 63, 123002

[3] http://www.mrao.cam.ac.uk/~jdm57/ssht/index.html
[4] http://www.mpa-garching.mpg.de/ift/nifty/
[5] http://en.wikipedia.org/wiki/Intel_MIC

Paper V

SymPix: A spherical grid for efficient sampling of rotationally invariant operators

The algorithm of Paper I took a very long time for me to tune correctly, not least because every time I changed any parameters I had to wait until next day for the precomputations to finish. After creating SymPix these debug cycles were reduced to a small coffee break. The final component seperation algorithm in Paper II does however not make use of the results of this paper.

The Astrophysical Journal Supplement Series, 222:17 (10pp), 2016 February doi:10.3847/0067-0049/222/2/17

SYMPIX: A SPHERICAL GRID FOR EFFICIENT SAMPLING OF ROTATIONALLY INVARIANT OPERATORS

D. S. Seljebotn and H. K. Eriksen
Institute of Theoretical Astrophysics, University of Oslo, P.O. Box 1029 Blindern, NO-0315 Oslo, Norway; d.s.seljebotn@astro.uio.no
Received 2015 April 20; accepted 2015 December 1; published 2016 February 10

ABSTRACT

We present SymPix, a special-purpose spherical grid optimized for efficiently sampling rotationally invariant linear operators. This grid is conceptually similar to the Gauss–Legendre (GL) grid, aligning sample points with iso-latitude rings located on Legendre polynomial zeros. Unlike the GL grid, however, the number of grid points per ring varies as a function of latitude, avoiding expensive oversampling near the poles and ensuring nearly equal sky area per grid point. The ratio between the number of grid points in two neighboring rings is required to be a low-order rational number (3, 2, 1, 4/3, 5/4, or 6/5) to maintain a high degree of symmetries. Our main motivation for this grid is to solve linear systems using multi-grid methods, and to construct efficient preconditioners through pixel-space sampling of the linear operator in question. As a benchmark and representative example, we compute a preconditioner for a linear system that involves the operator $\widehat{\boldsymbol{D}} + \widehat{\boldsymbol{B}}^T \boldsymbol{N}^{-1} \widehat{\boldsymbol{B}}$, where $\widehat{\boldsymbol{B}}$ and $\widehat{\boldsymbol{D}}$ may be described as both local and rotationally invariant operators, and $\boldsymbol{N}$ is diagonal in the pixel domain. For a bandwidth limit of $\ell_{\rm max} = 3000$, we find that our new SymPix implementation yields average speed-ups of 360 and 23 for $\widehat{\boldsymbol{B}}^T \boldsymbol{N}^{-1} \widehat{\boldsymbol{B}}$ and $\widehat{\boldsymbol{D}}$, respectively, compared with the previous state-of-the-art implementation.

Key words: cosmic background radiation – cosmology: miscellaneous – methods: data analysis – methods: numerical – methods: statistical

1. INTRODUCTION

Unlike the plane, it is impossible to construct a regular discretization of the sphere. Instead, every conceivable spherical grid comes with its own set of trade-offs, emphasizing one or more features at the cost of others. Thus, there is no such thing as a perfect spherical grid, but the optimal grid instead depends sensitively on the application under consideration.

In this paper, we will restrict our attention to high-resolution grids designed for fast and accurate spherical harmonic transforms (SHTs). In such cases, the primary consideration is that the grid must allow for efficient $\mathcal{O}(\ell_{\rm max}^3)$ SHTs, where $\ell_{\rm max}$ denotes the upper harmonic space bandwidth limit of the field in question, as opposed to the $\mathcal{O}(\ell_{\rm max}^4)$ scaling resulting from naive brute-force summation. This requires the use of FastFourier Transforms (FFTs) in the longitudinal direction, which in turn implies that (i) sample points must be placed on a set of iso-latitude rings, and (ii) sample points within each ring must be equidistant. However, there is still flexibility in choosing the latitude of each ring ($\theta_j \in [0, \pi]$), the number of grid points along each ring (n_j), and the initial offset of each ring ($\phi_{0,j}$).

Three popular spherical grids are the equiangular grid, the Gauss–Legendre (GL) grid (e.g., Doroshkevich et al. 2005), and HEALPix[1] (Górski et al. 2005). Of these, the equiangular grid is the most straightforward, simply defined by evenly spaced grid points (θ_i, ϕ_i) in both directions. This grid is typically used for geographical maps, and it is therefore also called a geographical grid.

Similarly, the standard GL grid has a constant number of grid points per ring. However, the ring latitudes θ_j are defined such that $P_{N_{\rm rings}}(\cos\theta_j) = 0$, where P_n is the Legendre polynomial of degree n. This simple modification allows efficient spherical harmonic analysis to machine precision, and the grid is thus optimized for spherical harmonics transforms.

[1] http://healpix.sourceforge.net

Both of these grids suffer from a massive oversampling of the polar regions (θ close to 0 or π) compared to the equatorial region ($\theta \approx \pi/2$), and this renders them sub-optimal, and sometimes even useless, for certain practical applications. An important example is the solution of discretized and bandwidth-limited linear systems. If there is a large number of sample points within the correlation length implied by $\ell_{\rm max}$, the system becomes degenerate and numerically unstable. Grids with nearly constant pixel areas perform much better than grids with strongly varying pixel areas for these types of applications.

One example of such grids is HEALPix, which is short for "Hierarchical Equal Area and Latitude Pixelization." By construction, this grid has both constant area pixel area per pixel and grid points located on iso-latitude, so it is a good general-purpose grid. However, this generality comes at the cost of spherical harmonics precision, as well as a low level of internal pixel symmetries.

The latter point is particularly important for our applications. Consider a function of two grid points, $\hat{n}_1$ and $\hat{n}_2$, that is both localized and rotationally invariant,

$$f(\hat{n}_1, \hat{n}_2) = \begin{cases} f(\hat{n}_1 \cdot \hat{n}_2) & \text{if } \arccos(\hat{n}_1 \cdot \hat{n}_2) < k\Delta \\ 0 & \text{otherwise,} \end{cases} \quad (1)$$

where Δ denotes the average distance between two neighboring grid points. Thus, f is assumed to be identically zero if the two grid points are separated by more than k grid units. In our applications, which employ multi-grid and/or preconditioning methods, we need to evaluate f for all relevant pairs $(\hat{n}_1, \hat{n}_2)$. Furthermore, because f typically is computationally expensive, it is important to minimize the total number of function evaluations, and large speed-ups can be gained by exploiting symmetries and caching.

For HEALPix, f needs to be evaluated $\mathcal{O}(k^2 N_{\rm pix})$ times, because the angular distances between neighboring grid points are all different, up to a handful of overall symmetries. In contrast, for the equiangular and GL grids only $\mathcal{O}(k^2\sqrt{N_{\rm pix}})$

evaluations are needed. Since the number of grid points is constant for every ring, we only need to evaluate f for the first grid point on every ring, accounting for all its neighbors, after which all function evaluations along the same ring will be given by symmetry.

In this paper, we construct a novel spherical grid called SymPix that combines the spherical harmonics transform precision of the GL grid with the nearly uniform sample point distances of HEALPix, while at the same time maintaining a high degree of symmetries within each ring, ensuring that fully sampling $f(\hat{n}_1 \cdot \hat{n}_2)$ scales as $\mathcal{O}(k^2\sqrt{N_{\rm pix}})$.

2. THE SYMPIX GRID

2.1. Ring Layout Basics

The main role of the SymPix grid is that of a supporting grid in internal multi-grid and/or preconditioning calculations, so maintaining high numerical precision is therefore essential. For this reason, we adopt the GL latitudinal ring layout as the basis of our grid. This provides support for both spherical harmonic *synthesis* (i.e., transforming from harmonic coefficients to pixel space) and *analysis* (transforming from pixel space to harmonic coefficients) to machine precision, by virtue of having an exact quadrature rule on the form

$$a_{\ell m} = \int_\Omega Y^*(\hat{n}) f(\hat{n}) d\Omega = \sum_i Y^*(\hat{n}_i) f(\hat{n}_i) w_i, \qquad (2)$$

where w_i is a set of quadrature weights. By placing rings exclusively on the zeros of the $\ell_{\rm max}$'th polynomial, one is guaranteed that $P_{\ell_{\rm max}+1}(\cos\theta_i) = 0$, and the discretized field is algebraically bandwidth-limited to harmonic modes with $\ell \leqslant \ell_{\rm max}$.

Next, we need to include enough sample points along each ring to fully resolve all spherical harmonic modes with $\ell \leqslant \ell_{\rm max}$. Formally speaking, this requires $2N_{\rm rings}$ grid points per ring. However, this requirement is somewhat counter-intuitive because it suggests massive oversampling of the polar regions compared to the equatorial region. And indeed, our intuition is correct: the spherical harmonic modes $Y_{\ell m}(\theta, \phi)$ are very close to zero in the polar regions for high ℓ and m, and these are the only modes that can cause high-frequency variation in the longitudinal direction. For this reason, the `libsharp` SHT package (Reinecke & Seljebotn 2013; Reinecke 2011) omits $Y_{\ell m}(\theta, \phi)$ whenever

$$\sqrt{m^2 - 2\,m\cos\theta} - \ell_{\rm max}\sin\theta > \max(100, 0.01\ell_{\rm max}), \qquad (3)$$

exploiting the fact that contributions from higher-ordered harmonics are numerically irrelevant. An explicit bound on the number of pixels required for machine precision was derived by Prézeau & Reinecke (2010), and Reinecke & Seljebotn (2013) used this to construct the *reduced GL grid*. Explicitly, for a given ring located at some latitude θ, Equation (3) defines the maximum m such that $Y_{\ell m}(\theta, \phi)$ does not vanish. The minimum number of pixels on that ring is then given by $2m + 1$, resulting in a longitudinal sample frequency that exceeds the Nyquist frequency.

2.2. Tiling

As discussed in Section 1, our primary usecase is evaluating a function $f(\hat{n}_1, \hat{n}_2)$ for all possible pairs $(\hat{n}_1, \hat{n}_2)$, but with the restriction that f is zero unless $\hat{n}_1$ and $\hat{n}_2$ are close together. To avoid unnecessary searches over vanishing pairs, we therefore partition our grid into a set of $k \times k$-sized tiles, where k is chosen such that $f(\hat{n}_1, \hat{n}_2) = 0$ unless $\hat{n}_1$ and $\hat{n}_2$ are either in the same tile or in two neighboring tiles. Thus, finding all relevant partner points for a given grid point simply amounts to a closest neighbor tile look-up. However, this also requires that the number of rings is divisible by k (letting $N_{\rm rings} > \ell_{\rm max} + 1$ if necessary), and that a set of k consecutive rings must have the same number of sample points. We will refer to each such set of k rings as a *band*.

2.3. Enforcing Symmetries

The main remaining step is to define the number of tiles per band. On the one hand, it must satisfy the minimum number of pixels given by Equation (3). On the other hand, it may be beneficial to increase it beyond this, in order to increase symmetries within and across bands. For instance, if we sample f from Equation (1) for all point pairs within a tile, the result can obviously be reused for all tiles in that band, since all between-point angular distances are conserved between tiles. Similarly, we can reuse results between neighboring tiles within the same band due to longitudinal symmetry.

In addition, we exploit the additional degrees of freedom in choosing the number of tiles to ensure symmetries with respect to latitudinally neighboring tiles. Specifically, we require that the number of tiles can increase from one band to the next only by a factor of exactly 3, 2, 1, 4/3, 5/4, or 6/5. Additionally, at least two bands in a row must have the same number of tiles, except for the polar bands. Finally, in order to avoid special cases we allow no equatorial ring (i.e., we insist that $N_{\rm rings}$ is an even number), and, purely conventionally, the location of the first grid point in a given ring is chosen to be half the pixel distance within that same ring. Together, these requirements ensure that the pattern of neighboring tiles repeats itself with a short period, and the total number of different cases evaluates scales as $\mathcal{O}(N_{\rm ring})$ rather than $\mathcal{O}(N_{\rm pix})$. We employ a dynamic programming algorithm to find the optimal number of tiles per band, subject to the constraints defined above, as detailed in Section 2.5. An example grid corresponding to $k = 2$ tiling is illustrated in Figure 1.

2.4. Memory Layout and Pixel Ordering

While the above constraints fully define the geometric properties of the SymPix grid, they do not imply a canonical memory layout or "pixel ordering." To fix this, we adopt two additional rules, both designed to maximize memory access efficiency and programming convenience.

First, the northern and southern hemispheres are band-wise interleaved. That is, we first list the northernmost polar band, followed by the southernmost polar band, followed by the second northern band and so on. The main advantage of this organization lies in convenient distributed programming across multiple computing nodes; interleaving the two hemispheres ensures that the same node can readily exploit north–south symmetries.

Second, grid points are latitudinally major-ordered within a given tile, i.e., the pixel ordering increases most rapidly along the θ direction. While the order within each tile could have been in any direction, this choice implies that pixel ordering is

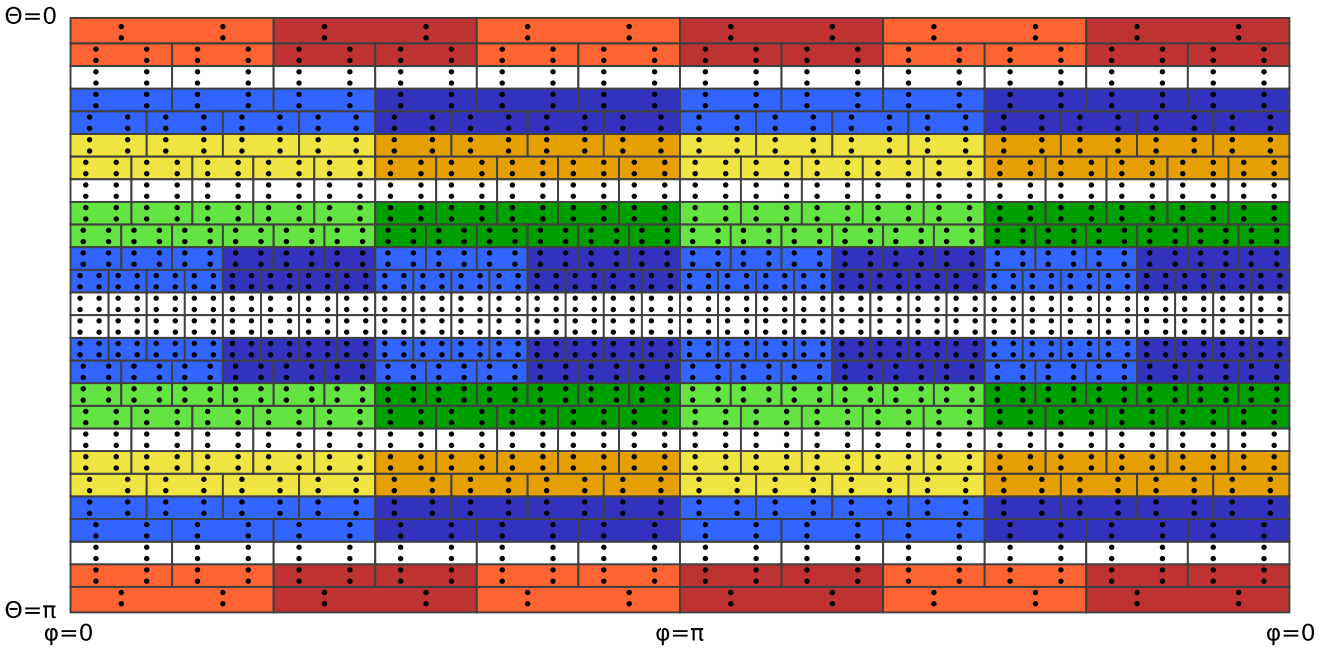


Figure 1. Geometric layout of SymPix sample points, implementing a cylindrical projection of the sphere. Each rectangle indicates a *tile* of (in this case) 2×2 sample points. For white tile-bands, the bands above and below have the same number of tiles, and angular distances between sample points in a given tile and sample points in the neighboring tiles are therefore constant throughout the band. Function evaluations depending only on angular distances may therefore be cached and reused. Colored tile-bands increment the number of tiles by a factor of 2 (red), $4/3$ (blue), $5/4$ (yellow), $6/5$ (green), and $4/3$ again (blue) toward the equator. For these bands, the neighboring tile relationship repeats itself (as indicated by shading), and there are still only a few cases that need to be computed and cached for each band.

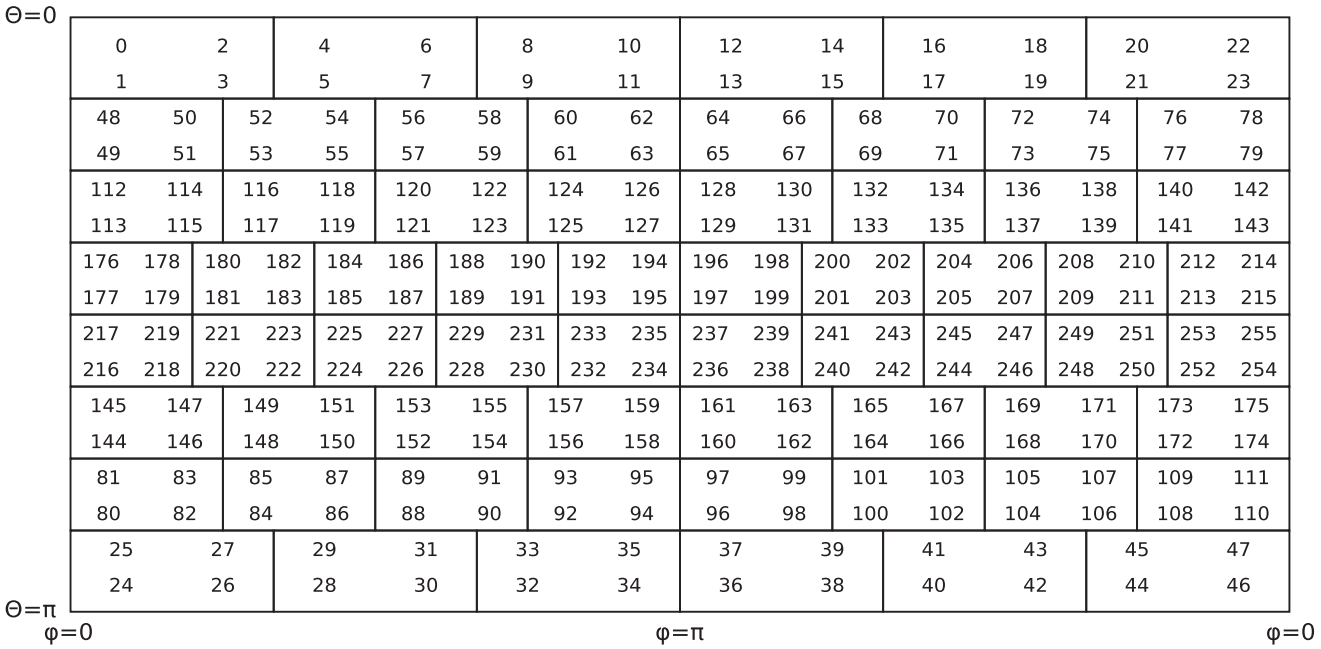


Figure 2. Memory ordering of SymPix sample points. Note that the resolution is lower than in Figure 1. Within each band the pixel order increases first latitudinally, i.e., along the θ direction. This ensures that access within the same tile is local in memory, and there are no discontinuities along each ring, which is convenient for SHTs. Additionally, to support efficient distributed programming, we interleave northern and southern bands, such that they naturally are assigned to the same node without explicit additional bookkeeping.

continuous across longitudinal tile borders, which is particularly convenient for SHTs.

Figure 2 provides an example of the resulting pixel ordering. Note that the resolution is lower than the corresponding illustration in Figure 1.

2.5. Grid Optimization

We end this section by describing the algorithm used to optimize the number of of tiles in each band, subject to the constraints defined in Section 2.3. We will only discuss the northern hemisphere, as the southern hemisphere is given directly by symmetry.

To initialize the algorithm, the user must provide a tile size k and a total number of rings $N_{\rm rings}$, where $N_{\rm rings}$ must be divisible by both 2 and k. The grid will be able to accurately represent fields that are band-limited at $\ell_{\rm max} = N_{\rm rings} - 1$. Together, these parameters specify the angular resolution of the grid, and correspond in principle to the HEALPix $N_{\rm side}$ parameter. We then number the bands by $i = 0, \ldots N_{\rm bands} - 1 \equiv N_{\rm rings}/(2k) - 1$, such that each band consists of k rings. We also define α_i to be the minimum number of tiles in each band subject to the constraint that the southmost ring within the band fulfills Equation (3).

Deriving the optimal SymPix grid is now equivalent to determining the number of tiles, T_i, for each band. For this optimization process we adopt the following cost function,

$$c(T_0, \ldots, T_{N_{\rm bands}-1}) \equiv \sum_i c_i(T_i) \equiv \sum_i (T_i - \alpha_i)^2, \quad (4)$$

which must be minimized, subject to

$$\frac{T_{i+1}}{T_i} \in \left\{ \frac{6}{5}, \frac{5}{4}, \frac{4}{3}, 1, 2, 3 \right\}. \quad (5)$$

Additionally, we initialize the recursion by defining T_0 as the smallest number larger than α_0 that is only a product of the factors 2, 3, and 5, and for computational speed we add the heuristic (or modification to the cost function) that $T_i < 3\alpha_i$, i.e., that no band should be over-pixelized by more than three times the Nyquist frequency.

The actual calculation is then a simple exercise in dynamic programming, as described in any standard text on algorithms (e.g., Cormen et al. 1989). Our implementation is summarized in Figure 3, which has a worst-case computational complexity of $\mathcal{O}(n\alpha_n) = \mathcal{O}(N_{\rm rings}^2) = \mathcal{O}(N_{\rm pix})$, and the same worst-case memory use. Due to the low computational complexity and the fact that the optimization only needs to be performed once per grid resolution, we do not present benchmarks for this operation; its computational cost is negligibly small for our purposes.

Optimal-SymPix-Grid:
Inputs:
- $\ell_{\rm max}$ – Band-limit of field to represent
- k – Tile size

Output:
- n – number of bands
- T_i – number of tiles in each band

Auxiliary:
- α_i – minimum number of tiles for band i
- $C_{i,t}$ – the cost of the best partial solution for bands 0 to i when assuming $T_i = t$
- $P_{i,t}$ – "previous-pointers"; when assuming $T_i = t$, the solution for bands 0 to i has $T_{i-1} = P_{i,t}$

Treat unassigned $C_{i,t}$ as ∞ and unassigned P_{i,T_0} as -1
$N_{\rm rings} \leftarrow \ell_{\rm max} + 1$ rounded up to next multiple of $2k$
$n \leftarrow N_{\rm rings}/2k$
Find θ_j for each ring j as for Gauss-Legendre grid
$T \leftarrow \max(100, \ell_{\rm max}/100)$
for each $i \in \{0, \ldots, n-1\}$:
  Find minimum m that satisfies Equation (3) for θ_{ik}
  $\alpha_i \leftarrow \lceil (2m+1)/k \rceil$
$T_0 \leftarrow \min(\{2^i 3^j 5^k \mid 2^i 3^j 5^k \geq \alpha_0, i \in \mathcal{N}, j \in \mathcal{N}, k \in \mathcal{N}\})$
$C_{i,T_0} \leftarrow (T_0 - \alpha_0)^2$
for each i **from** 1 **to** $n-1$:
  for each $t_{\rm prev}$ such that $C_{i-1,t_{\rm prev}} < \infty$:
    for each $x \in \{3, 2, 1, 4/3, 5/4, 6/5\}$ such that $x t_{\rm prev} \in \mathcal{N}$:
      $t \leftarrow x t_{\rm prev}$
      if $C_{i-1,t_{\rm prev}} + (\alpha_i - t)^2 < C_{i,t}$
          and $\alpha_i \leq t \leq 3\alpha_i$
          and $(P_{i-1,t_{\rm prev}} = t_{\rm prev}$ or $x = 1)$:
        $C_{i,t} \leftarrow c + (\alpha_i - t)^2$
        $P_{i,t} \leftarrow t_{\rm prev}$
$T_{n-1} \leftarrow \mathrm{argmin}_t(C_{n,t})$
for each i **from** $n-2$ **to** 1:
  $T_i \leftarrow P_{i+1,T_{i+1}}$

Figure 3. Dynamic programming algorithm for optimizing the SymPix grid layout. In summary, the algorithm considers all possible solutions, and employs look-up tables of partial solutions for bands 0 to $i-1$ when considering band i. The condition $P_{i-1,t_{\rm prev}} = t_{\rm prev}$ ensures that at least two bands in a row have the same number of tiles, except (possibly) for the first two rows, $T_1 \neq T_0$.

3. BENCHMARKS AND COMPARISONS

Before considering specific applications, we first characterize the basic performance of the SymPix grid in terms of computational efficiency and numerical accuracy.

3.1. Geometric Efficiency

We start by quantifying the geometric efficiency of our grid, as characterized by the overall number of grid points and the pixel area uniformity. For these tests, we consider an example grid with $\ell_{\rm max} = 2000$ and $k = 4$, sufficient to discretize a spherical field with an angular resolution of 15′ FWHM. Running the algorithm summarized in Figure 3 with these input parameters yields a SymPix grid with 5.6×10^6 grid points.

In Figure 4 we compare the number of SymPix grid points per ring with the optimal number of points per ring used by the reduced GL grid (Reinecke & Seljebotn 2013). The ratio between the solid and dashed lines thus indicates the amount of longitudinal oversampling implied by the SymPix grid. Except for very close to the poles, where there are very few points in terms of absolute numbers, this ratio is never larger than 1.35.

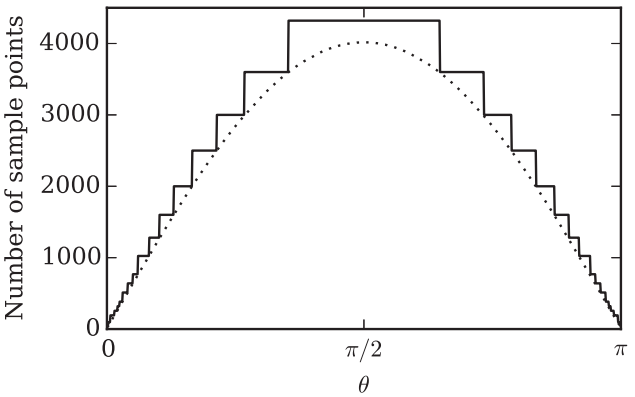


Figure 4. Number of SymPix grid points per ring as a function of latitude (solid line). The dotted line shows α_i, i.e., the same quantity for the reduced Gauss–Legendre grid (Reinecke & Seljebotn 2013).

A similar illustration is provided in Figure 5, where we plot the pixel area as a function of latitude, defining pixel borders strictly along longitudes and latitudes. The pixel area is given in units of the pixel area averaged over the full-sky, i.e., $4\pi/N_{\rm pix}$, such that a perfectly uniform pixelization, like HEALPix, corresponds to a constant value of unity. Overall, we see that the effective pixel areas vary at most by 20% relative to the average, except near the poles, where the normalized area may be as low as 0.1.

Figure 6 shows a histogram of normalized pixel areas, and we see that the vast majority of grid points have a normalized

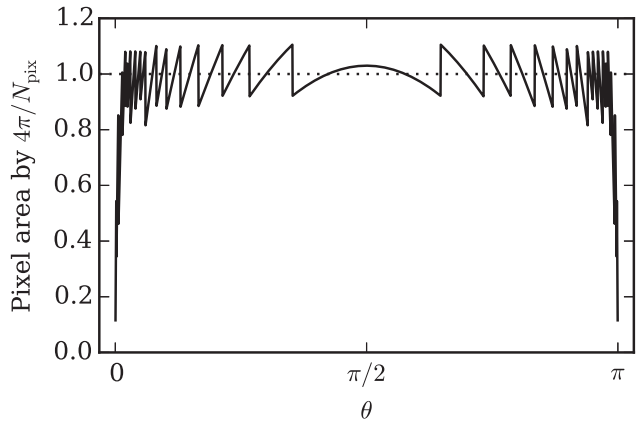


Figure 5. SymPix pixel area as a function of latitude in units of $4\pi/N_{\rm pix}$ (solid line). For the HEALPix grid, pixel areas are perfectly uniform (dotted line), while significant oversampling occurs close to the poles for the SymPix grid.

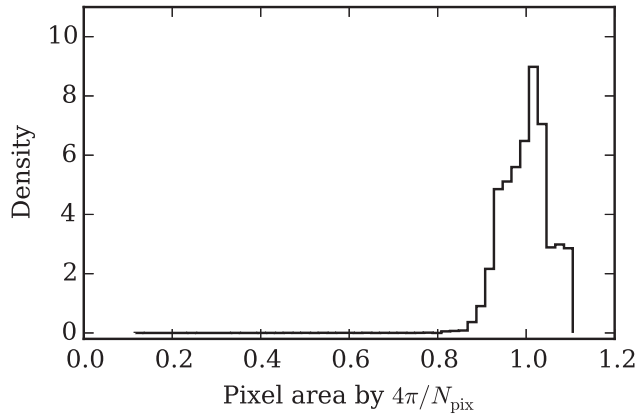


Figure 6. Histogram of normalized SymPix pixel areas. The tail extending below 0.8 corresponds to polar oversampling, and contains about 0.4% of the total number of pixels for this particular grid setup.

area between 0.9 and 1.1. The tail below 0.8 corresponds to the over-pixelized polar caps, and these contain only 0.4% of the total number of grid points for this particular example. Overall, the SymPix grid implies an oversampling of about 11% compared to the reduced GL grid, which is acceptable for our purposes.

3.2. Accuracy of Spherical Harmonic Quadrature

Next, we compare the numerical accuracy of spherical harmonics transforms as implemented on the SymPix, HEALPix, and reduced GL grids. This test is carried out through the following experiment.

1. We draw a fiducial signal $\boldsymbol{a} = \{a_{\ell m}\}$ in a spherical harmonic domain, band-limited by some $\ell_{\rm max}$. The spherical harmonics coefficients are drawn such that they correspond to an random isotropic and Gaussian field with a power spectrum (coefficient variance) $C_\ell \equiv 1/(\ell(\ell+1))$ for $\ell > 0$ and $C_0 \equiv 1$, i.e., the same overall properties as signals of interest for cosmic microwave background (CMB) analysis.
2. We project this signal onto the respective grid sample points by spherical harmonic synthesis.
3. We convert the real-space signal back to harmonic space through spherical harmonic analysis, including multipoles up to $\ell_{\rm max}$, to recover $\widehat{\boldsymbol{a}}$.
4. We repeat this procedure $N_{\rm sim}$ times, and summarize the results in terms of relative round-trip errors, $e^{(i)}_{\ell m} \equiv (\widehat{a}^{(i)}_{\ell m} - a^{(i)}_{\ell m})/\sqrt{C_\ell}$.

Before presenting the results, we note that no fundamental band-limit and/or resolution parameter $N_{\rm side}$ exist for HEALPix for a given angular resolution. For instance, changing the band-limit $\ell_{\rm max}$ will add/reduce aliasing for *all* scales. A quantitative head-to-head comparison at a given resolution is therefore difficult, as additional parameter tuning can affect the results. With this caveat in mind, in Table 1 we present results for three different band-limits, $\ell_{\rm max} = \{2.0, 2.5, 3.0\}N_{\rm side}$, with $N_{\rm side} = 256$, quoting both the maximum and mean errors as evaluated over all error coefficients $e^{(i)}_{\ell m}$. Each case includes $N_{\rm sim} = 100$ simulations, and the SymPix tile size is fixed at $k = 8$.

Starting with the highest-bandwidth case, $\ell_{\rm max} = 3N_{\rm side}$, we first note that the regular GL grid is the only grid that achieves overall machine precision, with a mean error of $\mathcal{O}(10^{-14})$ and a maximum error of $\mathcal{O}(10^{-12})$. For comparison, the corresponding mean and maximum SymPix errors are $\mathcal{O}(10^{-6})$ and $\mathcal{O}(10^{-2})$, respectively, while HEALPix achieves $\mathcal{O}(10^{-1})$ and $\mathcal{O}(1)$ for this high-bandwidth case. Reducing the band-limit to $\ell_{\rm max} = 2N_{\rm side}$ improves the latter by about two orders of magnitude. As already noted by Górski et al. (2005), the large difference between the average and maximum error is largely driven by the $m = 0$ modes, which integrate poorly on iso-latitude rings; both types of errors can, however, be reduced by iterative quadrature.

The statistics listed in Table 1 provide only a very coarse comparison point, because the round-trip errors are highly scale-dependent. In Figure 7 we therefore plot the error as a function of multipole, ℓ, choosing the SymPix and HEALPix band-limits such that the corresponding grids roughly match a HEALPix $N_{\rm side} = 256$ grid in terms of the total number of sample points. For SymPix, this corresponds to $\ell_{\rm max} = 735$, and for the GL grid it is $\ell_{\rm max} = 628$.

Starting with the GL grid (blue lines), we see that the error reaches machine precision up to the bandwidth limit; at higher multipoles no information is carried by the grid. In contrast, the SymPix grid reaches machine precision up to $\ell \approx 0.5\ell_{\rm max}$, while the error increases more smoothly at higher multipoles. However, even though the high-ℓ error increase is smooth, it is still exponential, and the mean and maximum statistics listed in Table 1 are therefore strongly dominated by the small-scale errors. Thus, by virtue of deriving its main geometric grid layout from the GL grid, we see that the numerical performance of the SymPix grid is excellent on large and intermediate angular scales, and the cost of its superior symmetry properties primarily comes in the form of sub-optimal small-scale residuals. For comparison, the HEALPix errors are roughly constant at $\mathcal{O}(10^{-4})$–$\mathcal{O}(10^{-2})$, and vary only weakly with angular scale. Note that in all cases the errors can be reduced by iteration techniques, essentially using least squares minimization to find the spherical harmonic signal with the least power that projects exactly to the map, and employing the result of spherical harmonic analysis as a preconditioner.

The large errors seen for the GL grid above $\ell_{\rm max}$ are due to undersampling, or equivalently, aliasing. In Figure 8 we study this effect directly by varying the spherical harmonics

Table 1
Comparison of Different Grids in Terms of the Number of Pixels and the Accuracy of Spherical Harmonic Analysis

$\ell_{\max}$	Grid	Parameter	$N_{\rm pix}$	$N_{\rm pix}/N_{\rm pix}^{\rm HEALPix}$	Max. Error	Mean Error	CPU Time for SHT (ms)
511	HEALPix	$N_{\rm side} = 256$	786 432	1.00	$2.1 \cdot 10^{-2}$	$2.9 \cdot 10^{-5}$	160
	SymPix	$\ell_{\max} = 511$	390 656	0.50	$7.8 \cdot 10^{-3}$	$8.1 \cdot 10^{-7}$	67
	Gauss–Legendre	$\ell_{\max} = 511$	524 288	0.67	$7.5 \cdot 10^{-13}$	$2.8 \cdot 10^{-14}$	66
639	HEALPix	$N_{\rm side} = 256$	786 432	1.00	$2.2 \cdot 10^{-1}$	$1.3 \cdot 10^{-3}$	219
	SymPix	$\ell_{\max} = 639$	591 232	0.75	$7.2 \cdot 10^{-3}$	$1.1 \cdot 10^{-6}$	118
	Gauss–Legendre	$\ell_{\max} = 639$	819 200	1.04	$1.2 \cdot 10^{-12}$	$3.2 \cdot 10^{-14}$	118
767	HEALPix	$N_{\rm side} = 256$	786 432	1.00	$1.6 \cdot 10^{0}$	$6.8 \cdot 10^{-2}$	287
	SymPix	$\ell_{\max} = 767$	838 656	1.07	$4.0 \cdot 10^{-2}$	$4.8 \cdot 10^{-6}$	188
	Gauss–Legendre	$\ell_{\max} = 767$	1 179 648	1.50	$1.0 \cdot 10^{-12}$	$3.8 \cdot 10^{-14}$	188

Note. The HEALPix resolution is kept constant at $N_{\rm side} = 256$, while the spherical harmonic band-limit varies over $\ell_{\max} = \{2.0, 2.5, 3.0\} N_{\rm side}$. The SymPix and Gauss–Legendre band-limits are identical to the spherical harmonic band-limit.

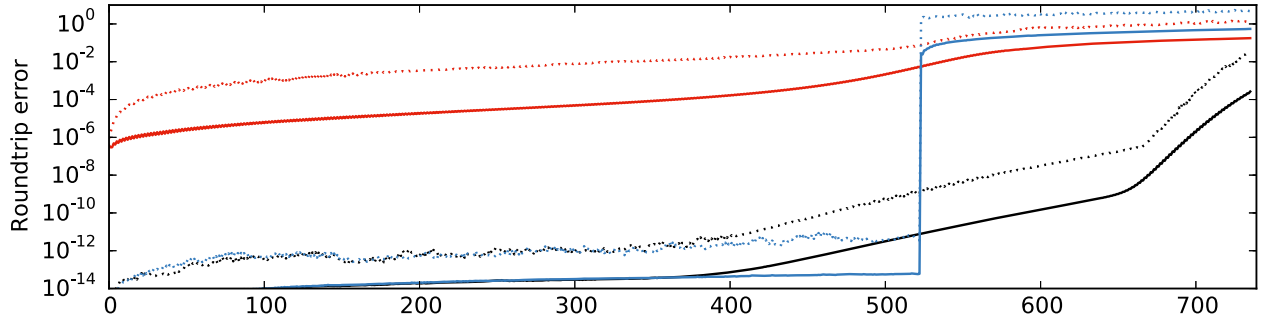


Figure 7. Spherical harmonic round-trip error as a function of multipole, summarized in terms of maximum (dotted lines) and mean (solid lines) errors, averaged over both harmonic quantum number m and $N_{\rm sim} = 100$ simulations. Black lines show results for a SymPix grid with $\ell_{\max} = 735$ and tile size 8; red lines show results for a HEALPix grid with $N_{\rm side} = 256$ and $\ell_{\max} = 735$; and blue lines show results for a regular Gauss–Legendre grid with $\ell_{\max} = 628$. All grids have roughly the same number of grid points, $N_{\rm pix} \approx 780,000$.

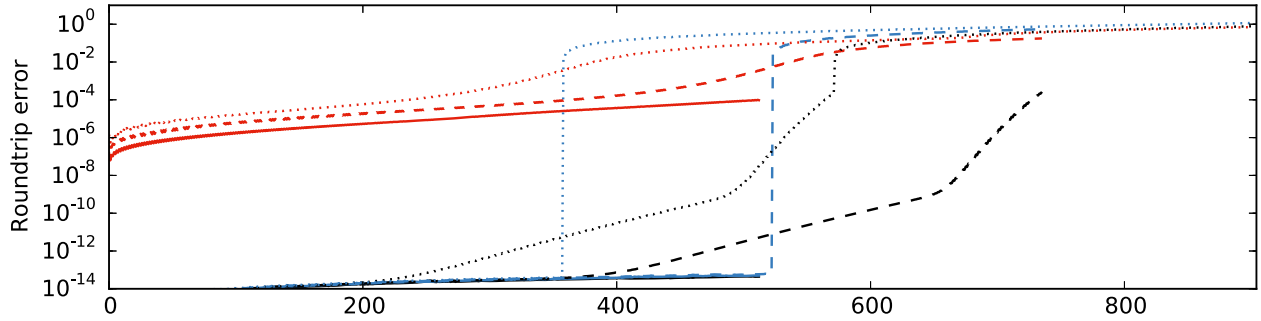


Figure 8. Error induced by undersampling (aliasing) as a function of multipole in terms of average errors, averaged over both harmonic quantum number m and $N_{\rm sim} = 100$ simulations. The experimental setup is the same as in Figure 7, but the spherical harmonic bandwidth limit varies between $\ell_{\max} = 512$ (solid), $\ell_{\max} = 735$ (dashed), and $\ell_{\max} = 900$ (dotted).

bandwidth limit between $\ell_{\max}^{\rm SH} = 512$, 735 and 900; note, however, that the actual grid resolution parameters are kept fixed at the above values, and the higher resolutions enforced here therefore no longer match the respective grid properties. Considering first the GL grid with a SHT band-limit of $\ell_{\max} = 512$, we see, as expected, that the errors reach machine precision at all scales. However, for the higher band-limits, $\ell_{\max} = 735$ and 900, both of which are higher than the grid resolution of $\ell_{\max}^{\rm grid} = 628$, the errors saturate at a multipole *below* the grid resolution. To be specific, the critical multipole is $2\ell_{\max}^{\rm grid} - \ell_{\max}^{\rm SH}$, corresponding to the well-known aliasing limit from standard Fourier theory. However, at lower multipoles no aliasing is observed for the GL grid, which implies that it is fully robust with respect to undersampling, given a known band-limit.

In comparison, the corresponding HEALPix errors are non-local, in the sense that increasing the spherical harmonics band-limit increases the errors at *all* angular scales: the dotted line ($\ell_{\max} = 900$) lies consistently higher than the dashed line ($\ell_{\max} = 735$), which in turn lies consistently higher than the solid line ($\ell_{\max} = 512$). The HEALPix grid is thus not robust against undersampling, and it is very important to choose a grid resolution appropriate for the bandwidth of the signal under consideration, which in several applications may imply oversampling the signal.

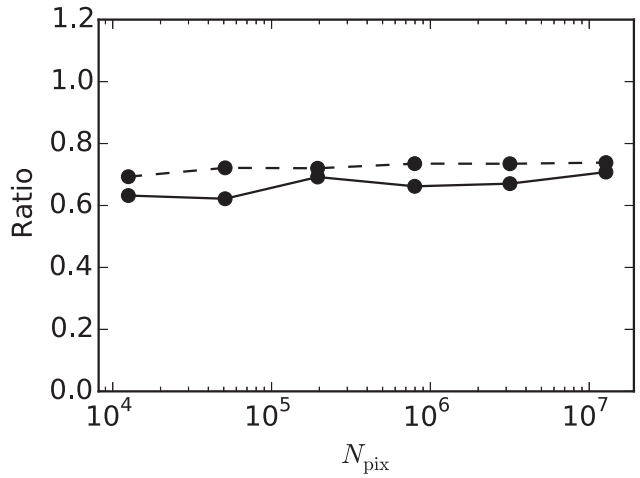


Figure 9. Comparison between spherical harmonic transforms cost as performed with SymPix and HEALPix as a function of $N_{\rm pix}$, plotted in terms of their ratios (black solid line). The dashed line shows the ratio between the number of grid point rings.

The SymPix grid performance lies, as expected, between those of GL and HEALPix. On large angular scales, it achieves numerical precision, while on small scales the aliasing increases exponentially with multipole, and eventually reaches similar levels as HEALPix.

3.3. Computational Speed of SHTs

Before ending this section, we compare the performance of the SymPix, HEALPix, and GL grids in terms of computational speed. The rightmost column in Table 1 lists the CPU time for each of the cases considered above in units of wall-clock milliseconds, while Figure 9 presents a head-to-head comparison of the SymPix and HEALPix grid performance as a function of $N_{\rm pix}$. All benchmarks were performed using `libsharp` on a single Intel Core i7 Q840 at 1.87 GHz (SSE2); for full details including CPU times in absolute numbers, we refer the interested reader to Reinecke & Seljebotn (2013).

Overall, SymPix perform similarly to the GL grid, and both execute about 30% faster than HEALPix. This latter difference may be explained by the fact that the HEALPix grid points form a zig-zag pattern in which every other ring is longitudinally shifted by half a pixel width. This implies a grid point organization that comprises about 30% more rings than GL and SymPix grids, which exhibit more regular longitudinal pixel organizations. This is relevant, because the computational complexity of SHTs scales as

$$C_{\rm SHT} = \mathcal{O}(N_{\rm ring}\ell_{\rm max}^2) + \mathcal{O}\left(N_{\rm pix}\log\frac{N_{\rm pix}}{N_{\rm ring}}\right) = \mathcal{O}(\ell_{\rm max}^3) + \mathcal{O}(\ell_{\rm max}^2\log\ell_{\rm max}). \quad (6)$$

The first term represents the cost of computing the associated Legendre polynomials for each ring, and dominates the second term, which accounts for evaluating FFTs along each ring. Thus, the number of grid points per ring is not critical for the overall speed of SHTs, while the total number of rings is.

In addition, by construction, SymPix grids have rings with pixel numbers that are only products of 2, 3, and/or 5, which ensures efficient FFTs. In contrast, many HEALPix rings have pixel numbers that includes large primes, and therefore the Bluestein algorithm must be employed for these. This effect is more important for lower-resolution grids, for which the cost of FFTs is relatively higher.

4. APPLICATIONS

We now turn our attention to practical applications, and in particular to the construction of efficient preconditioners. Before doing that, however, we consider a simpler application, namely real-space convolution, in order to build up intuition regarding the relevant operations. We emphasize that the purpose of this preliminary discussion is not to provide a real-world alternative to SHTs, or the methods presented by Elsner & Wandelt (2011) and Sutter et al. (2012) for such convolutions, but simply to quantify the computational efficiency of the SymPix grid on a simple and intuitive application.

4.1. Spherical Convolution

The convolution of a spherical image f with a kernel b is given by the spherical surface integral

$$g(\hat{n}) = \int_{4\pi} b(\hat{n}, \hat{m}) f(\hat{m}) d\Omega_{\hat{m}}. \quad (7)$$

In our case we assume an azimuthally symmetric kernel, and $b(\hat{n}, \hat{m})$ therefore depends only on the distance between $\hat{n}$ and $\hat{m}$, such that

$$g(\hat{n}) = \int_{4\pi} b(\hat{n}\cdot\hat{m}) f(\hat{m}) d\Omega_{\hat{m}}. \quad (8)$$

This integral is most commonly performed in a spherical harmonic domain, turning full-sky convolution into coefficient-wise multiplication with a corresponding transfer function, b_ℓ, which is given by the Legendre transform of $b(\hat{n}\cdot\hat{m})$. These computations are dominated by the SHTs, and therefore have a computational scaling of $\mathcal{O}(N_{\rm pix}^{3/2}) = \mathcal{O}(\ell_{\rm max}^3)$.

If b is spatially narrow compared to the required pixelization, as is usually the case, one could instead consider the pixel-domain convolution by evaluating

$$g(\hat{n}_i) = \sum_{j=1}^{N_{\rm pix}} b(\hat{n}_i\cdot\hat{n}_j) f(\hat{n}_j), \quad (9)$$

where the convolution kernel reads

$$b(x) = \sum_{\ell=0}^{\ell_{\rm max}} \frac{2\ell+1}{4\pi} b_\ell P_\ell(x). \quad (10)$$

One would then make the approximation that $b(\hat{n}_i\cdot\hat{n}_j) = 0$ whenever sample points i and j are more than k sample point distances apart, as discussed in Section 1.

For HEALPix, almost all sample point distances are different, and b must therefore be evaluated $\mathcal{O}(N_{\rm pix}\,k^2)$ times. The computational complexity of pixel-domain convolution on the HEALPix grid therefore scales as $\mathcal{O}(N_{\rm pix}\,k^2\,\ell_{\rm max}) = \mathcal{O}(k^2\,\ell_{\rm max}^3)$, which is clearly inferior to the harmonic approach both in terms of speed and accuracy. With SymPix, however, the large number of symmetries allows us to reduce the computational complexity to $\mathcal{O}(k^2\,N_{\rm pix} + \sqrt{N_{\rm pix}}\,\ell_{\rm max}) = \mathcal{O}(k^2\,N_{\rm pix})$: one simply needs to choose a tile size k such that only sample point pairs within a

Table 2
CPU Time and Theoretical Speed-up for Evaluating $b(\hat{n}\cdot\hat{m})$

$\ell_{\max}$	CPU Time (s)	Speed-up (factor)
3000	9.8	732
1500	3.6	335
750	1.4	149
375	0.74	70
188	0.50	26
100	0.31	14

Note. We have approximated $b(\hat{n}\cdot\hat{m}) = 0$ whenever $\hat{n}$ and $\hat{m}$ are not in neighboring tiles. The third column shows the number of non-zero $b(\hat{n}\cdot\hat{m})$, which scales as $O(k^2 N_{\rm pix})$, divided by the number of elements we had to compute when making use of the SymPix symmetries, which scales as $O(k^2\sqrt{N_{\rm pix}})$. In this example we have chosen $k = 8$.

tile and between neighboring tiles must be considered. Then for, each band of k rings, $b(\hat{n}\cdot\hat{m})$ only needs to be evaluated for the first few tiles of the band, as other distances within the same band will be identical within the remainder of the band.

The speed-ups for evaluating all necessary $b(\hat{n}\cdot\hat{m})$, when approximating $b(\hat{n}\cdot\hat{m}) = 0$ whenever $\hat{n}$ and $\hat{m}$ are not in neighboring tiles, are given in Table 2. In addition to scaling better than the $\mathcal{O}(N_{\rm pix}^{3/2})$ SHTs, this approach should also be easier to parallelize and implement efficiently on a GPU.

Note that yet another method for spherical convolution with a symmetric kernel has been implemented in the ARKCoS code (Elsner & Wandelt 2011; Sutter et al. 2012), with a computational scaling of $\mathcal{O}(k\,\ell_{\max}^2\,\log\ell_{\max}) = \mathcal{O}(k\,N_{\rm pix}\log N_{\rm pix})$. Whether a SymPix-based convolution would improve relative to their work for relevant resolution parameters and accuracy requirements remains to be explored.

4.2. Preconditioner Construction for Linear Systems

Finally, we are in the position to discuss the application of the SymPix grid to our main usecase, namely for solving linear systems involving rotationally invariant operators in a pixel domain, either through multi-grid methods or constructing efficient preconditioners. The simplest example of such a system is

$$\boldsymbol{Y}\boldsymbol{B}\boldsymbol{Y}^T\boldsymbol{x} = \boldsymbol{b}, \tag{11}$$

where $\boldsymbol{Y}$, as usual, is the matrix corresponding to spherical harmonic synthesis and $\boldsymbol{B}$ is a diagonal matrix in a spherical harmonic domain, $B_{\ell m,\ell' m'} = b_\ell\,\delta_{\ell,\ell'}\,\delta_{mm'}$. The product $\boldsymbol{Y}\boldsymbol{B}\boldsymbol{Y}^T$ is a pixel-domain operator with strong spatial couplings within the correlation length implied by b. Of course, this particular system could have been trivially solved by converting to a spherical harmonic domain, which would diagonalize the coefficient matrix. However, if there are more terms in the operator, this is no longer possible, and iterative solvers like Conjugate Gradients or multi-level algorithms are needed. In these cases SymPix is useful to construct preconditioners or smoothers.

Our own main interest lies in drawing constrained Gaussian realizations of the CMB sky by using a multi-level solver (Seljebotn et al. 2014). This maybe performed by solving the following linear system (Eriksen et al. 2004; Jewell et al. 2004; Wandelt et al. 2004),

$$\boldsymbol{Y}_{\rm I}(\boldsymbol{D} + \boldsymbol{B}\boldsymbol{Y}_{\rm obs}\boldsymbol{N}^{-1}\boldsymbol{Y}_{\rm obs}^T\boldsymbol{B})\boldsymbol{Y}_{\rm I}^T\boldsymbol{x} = \boldsymbol{r}, \tag{12}$$

where $\boldsymbol{D}$ and $\boldsymbol{B}$ are diagonal matrices in a spherical harmonic domain, characterized by transfer functions d_ℓ and b_ℓ, $\boldsymbol{N}^{-1}$ is a diagonal (inverse noise covariance) matrix in a pixel domain, pixelized on some external grid θ_i, and $\boldsymbol{r}$ is a stochastic term that depends on the data set in question.

Two different spherical grids are involved in a system. First, the outermost spherical harmonics transform, $\boldsymbol{Y}_{\rm I}$, denotes synthesis to a grid of our own choosing. We will use a SymPix grid of resolution $\ell_{\max}$ for this operator in the following. The inner transform, $\boldsymbol{Y}_{\rm obs}$, is determined by some external experiment, and is thus not flexible. Here we will assume that this operator is defined on a full-sky HEALPix grid of $N_{\rm side} = 2048$, typical for the CMB maps published by the *Planck* experiment (Planck Collaboration 2015).

Of course, from the viewpoint of the overall linear system, the details of any individual operator are irrelevant, and the only crucial point is that the combined operator remains the same. In order to speed up the calculations through the use of symmetries, we therefore substitute the innermost HEALPix-based noise covariance matrix product with a corresponding SymPix-based product,

$$\boldsymbol{Y}_{\rm obs}\boldsymbol{N}^{-1}\boldsymbol{Y}_{\rm obs}^T = \boldsymbol{Y}_2\boldsymbol{N}_2^{-1}\boldsymbol{Y}_2^T, \tag{13}$$

where $\boldsymbol{Y}_2$ denotes an auxiliary SymPix grid; note that this does not need to be the same as $\boldsymbol{Y}_{\rm I}$, but its resolution can be adjusted to trade numerical precision for computational speed. As shown by Seljebotn et al. (2014), Equation (13) holds true if $\boldsymbol{N}_2$ is constructed from

$$\boldsymbol{\theta}_2 = \boldsymbol{W}_2\boldsymbol{Y}_2\boldsymbol{Y}_{\rm obs}^T\boldsymbol{\theta}, \tag{14}$$

in the same way as $\boldsymbol{N}$ is constructed from θ. In this latter expression, $\boldsymbol{W}_2$ is a diagonal matrix containing the quadrature weights used in the spherical harmonic analysis of the target grid, while $\boldsymbol{Y}_{\rm obs}^T$ lacks the ring weights one normally uses in spherical harmonic analysis. Note that this operation is in fact the opposite procedure compared to naive resampling, which would be written as $\boldsymbol{Y}_2\boldsymbol{Y}_{\rm obs}^T\boldsymbol{W}_{\rm obs}$ in our notation. For full details, we refer the interested reader to Seljebotn et al. (2014).

The precision of Equation (13) depends on the relative band-limits of $\boldsymbol{Y}_{\rm I}$, $\boldsymbol{Y}_2$, and $\boldsymbol{Y}_{\rm obs}$. For instance, choosing $\ell_{\max}$ for $\boldsymbol{Y}_2$ and $\boldsymbol{Y}_{\rm obs}$ to be twice that of $\boldsymbol{Y}_{\rm I}$ yields a numerical precision of $\mathcal{O}(10^{-10})$. Increasing these to four times that of $\boldsymbol{Y}_{\rm I}$ results in an accuracy of $\mathcal{O}(10^{-14})$, whereas reducing it to only one, such that $\boldsymbol{Y}_{\rm I} = \boldsymbol{Y}_2$, gives an accuracy of $\mathcal{O}(10^{-2})$. Even the latter may be acceptable for preconditioning purposes.

In order to derive an approximation to the full coefficient matrix defined by Equation (12), we first rewrite the system as

$$\widehat{\boldsymbol{D}} + \widehat{\boldsymbol{B}}^T\boldsymbol{N}_2^{-1}\widehat{\boldsymbol{B}}\boldsymbol{x} = \boldsymbol{r}, \tag{15}$$

where

$$\widehat{\boldsymbol{D}} = \boldsymbol{Y}_{\rm I}\boldsymbol{D}\boldsymbol{Y}_{\rm I}^T \quad \text{and} \quad \widehat{\boldsymbol{B}} = \boldsymbol{Y}_2\boldsymbol{B}\boldsymbol{Y}_{\rm I}^T. \tag{16}$$

We now introduce the approximation that $\widehat{D}_{ij} = 0$ and $\widehat{B}_{ij} = 0$ whenever two sample points i and j are not in the same or neighboring tiles, as per the SymPix organization. The non-zero elements (i.e., the "local" part) of $\widehat{\boldsymbol{D}}$ and $\widehat{\boldsymbol{B}}$ are evaluated by Equation (10), at a cost of $\mathcal{O}(\ell_{\max})$ operations per matrix

Table 3
CPU Time for Constructing Preconditioner

$\ell_{\max}$	Naive (CPU min)	SymPix (CPU min)	Speed-up
Evalution of $\widehat{\boldsymbol{B}}^T \boldsymbol{N}^{-1} \widehat{\boldsymbol{B}}$			
3000	727	5.4	130
1500	509	1.4	360
750	340	0.37	920
375	230	0.11	2 100
188	452	0.035	13 000
100	363	0.027	13 000
Sum	2 621	7.3	360
Evalution of $\widehat{\boldsymbol{D}}$			
3000	85	3.3	26
1500	15	0.83	18
750	2.4	0.22	11
375	0.36	0.07	5
188	0.05	0.02	3
100	0.01	0.01	1
Sum	103	4.5	23

Note. The top section lists the CPU time for preconditioner calculations that depend only on data geometry (mask, beam, noise characterization), while the bottom section lists the corresponding CPU time for calculations that depend on d_ℓ, which in CMB applications typically corresponds to an angular power spectrum, C_ℓ. The second column is copied directly from Seljebotn et al. (2014), which does not employ any symmetries. The third row shows similar results using SymPix, while the fourth column shows the ratio between the two.

element. However, as discussed in Section 4.1, evaluating all required elements for a SymPix grid scales as $\mathcal{O}(k^2\ \sqrt{N_{\rm pix}})$, as opposed to $\mathcal{O}(k^2\ N_{\rm pix})$ for less symmetric grids.

These calculations constitute essential components of the pre-computation step of the multi-grid solver presented by Seljebotn et al. (2014). In that paper, all evaluations were performed without employing any symmetries, with a computational scaling of $\mathcal{O}(\ell_{\max} k^2 N_{\rm pix})$ as discussed above. Their Table 2 summarizes the resulting computational costs in units of CPU minutes. Here we repeat those calculations, adopting the exact same overall parameters, facilitating a one-to-one comparison, but we employ SymPix for intermediate calculations. The results are summarized in Table 3, in which the second column is copied directly from Seljebotn et al. (2014), and the third column shows the new SymPix results. The fourth column shows the ratio between the two.

Clearly, the net gains achieved by the SymPix grid vary with resolution. For the high-resolution levels the speed-up is driven by symmetries drastically reducing the time taken to evaluate $\widehat{\boldsymbol{B}}$. The theoretical speed-up of 732 times for evaluating $\widehat{\boldsymbol{B}}$ at $\ell_{\max} = 3000$, found in Table 2, is reduced to 130 and 26 for $\widehat{\boldsymbol{B}}^T \boldsymbol{N}^{-1} \widehat{\boldsymbol{B}}$ and $\widehat{\boldsymbol{D}}$, respectively. This is due to work that was previously unimportant now dominating the computation.

As already noted, the HEALPix grid also exhibits a handful of internal symmetries that could have been exploited in a similar manner to reduce the overall computing time. The benchmarks presented here therefore do not represent a head-to-head comparison of grids, but rather a comparison of specific implementations. To be explicit, the implementation presented by Seljebotn et al. (2014) may in theory be sped up by a factor of 24 if exploiting all HEALPix symmetries, and this factor should be compared to the results presented in the third column of Table 2. However, at lower resolutions the speed-ups seen in in Table 3 are almost entirely due to being able to use the operator resampling given in Equation (13). This degradation procedure is not as straightforward when using the HEALPix grid, as the approximation is significantly less accurate. Our previous code therefore used a resolution of $N_{\rm side} = 2048$ along columns on all the levels, leading to very long computation times.

To summarize, the SymPix grid reduces what used to be overnight jobs with our previous implementation to essentially interactive tasks.

5. CONCLUSION

We have presented SymPix, a novel spherical grid for efficiently sampling rotationally invariant operators. This grid derives many of its properties from the GL grid, ensuring overall excellent spherical harmonics transform performance. The main difference between the two grids is that SymPix sacrifices proper Nyquist sampling in the longitudinal direction in order to increase pixel symmetries, such that all grid pair distances repeat perfectly along constant-latitude rings. This decreases the computational scaling of evaluating rotationally invariant operators from $\mathcal{O}(N_{\rm pix})$ to $\mathcal{O}(\sqrt{N_{\rm pix}})$.

The intended primary application of the SymPix grid is efficient construction of preconditioners (or smoothers) for iterative linear solvers. In this paper we considered the specific example of drawing constrained Gaussian realizations using a multi-grid solver, which is an important problem in current CMB analysis. Comparing with previous state-of-the-art results (Seljebotn et al. 2014), we achieve average speed-ups of 360 and 23 for the two most important pre-computation steps when using SymPix for internal calculations.

However, we emphasize that SymPix is a special-purpose grid designed for precisely such tasks; it is not intended to provide a general-purpose spherical pixelization that is suitable for, say, mapmaking. HEALPix is clearly preferred for such purposes due to its uniform pixel areas, regular pixel window, and hierarchical pixel structure. Likewise, if machine precision spherical harmonics transforms are required, the GL grid is the obvious choice. However, for those particular applications that can benefit from efficient pixel-space sampling of linear operators, such as ours, SymPix holds a clear edge over existing alternatives.

D.S.S. and H.K.E. are supported by European Research Council grant StG2010-257080.

APPENDIX
CODE

The SymPix code has been developed as part of the Commmander project, and does not yet have its own library. For the benefit of the reader, however, we have copied the source files relevant to this paper to their own repository at http://github.com/dagss/sympix. Please consult the accompanying README file for further details. This repository will be updated if the code does eventually develop into a stand-alone package.

The SHTs are all done using `libsharp` (Reinecke & Seljebotn 2013), which, at the time of writing, is available at http://sourceforge.net/projects/libsharp/. We then construct the grid geometry in our Python code and feed it to

`libsharp`. In the future we may port our Python code to C and make it available directly in `libsharp`.

REFERENCES

Cormen, T. H., Leiserson, C. E., & Rivest, R. L. 1989, Introduction to Algorithms (Cambridge, MA: MIT Press)

Doroshkevich, A. G., Naselsky, P. D., Verkhodanov, O. V., et al. 2005, IJMPD, 14, 275

Elsner, F., & Wandelt, B. D. 2011, A&A, 532, A35

Eriksen, H. K., O'Dwyer, I. J., Jewell, J., et al. 2004, ApJS, 155, 227

Górski, K. M., Hivon, E., Banday, A. J., et al. 2005, ApJ, 622, 2

Jewell, J., Levin, S., & Anderson, C. H. 2004, ApJ, 609, 1

Planck Collaboration 2015, A&A, submitted, (arXiv:1502.01582)

Prézeau, G., & Reinecke, M. 2010, ApJS, 190, 267

Reinecke, M. 2011, A&A, 526, A108

Reinecke, M., & Seljebotn, D. S. 2013, A&A, 554, A112

Seljebotn, D. S., Mardal, K.-A., Jewell, J. B., Eriksen, H. K., & Bull, P. 2014, ApJS, 210, 24

Sutter, P., Wandelt, B. D., & Elsner, F. 2012, in Proc. Big Bang, Big Data, Big Computers (Big3), 2012

Wandelt, B. D., Larson, D. L., & Lakshminarayanan, A. 2004, PhRvD, 70, 08351

Paper VI (draft)

Planck 2017 results. II. Low Frequency Instrument data processing

For each Planck release one reaches new levels of sensitivity in the data analysis. In this release, Commander had a role in the core time-ordered data (TOD) processing of LFI. It turned out that in order to increase the precision of the estimate of the gain function, an estimate of the sky was needed. To make such an estimate of the sky, one needs data maps; to get data maps of the sky, one needs a gain function; and so this turned into an iterative scheme were results were sent back and forth between the Commander group in Oslo and the LFI TOD group, as sort of a poor mans iterative system solver. This underlines the need for Commander to directly work with TOD data in the future.

DRAFT!

Note: This paper has not been peer-reviewed as of this writing. It will be submitted to A&A.

Paper VII (draft)

Planck 2017 results. IV. Diffuse component separation

This paper presents the third Planck component seperation effort. The release contains results made with Commander 2 code, which employes the multi-resolution modelling and preconditioner of Paper II. Thus Commander can work directly with the native resolution of all of the Planck channels, and produce exact, joint estimates of the low-frequency foregrounds, high-frequency foregrounds, and the CMB.

DRAFT!

This paper has not been peer-reviewed as of this writing. It will be submitted to A&A.

Paper VIII

CMB likelihood approximation for banded probability distributions

I contributed some ideas in the early phases of this project. In particular I contributed to an early attempt at using splines to regularize the Blackwell-Rao estimator. This was a stepping stone to the simpler solution presented in this paper.

The Astrophysical Journal, 777:150 (9pp), 2013 November 10 doi:10.1088/0004-637X/777/2/150

COSMIC MICROWAVE BACKGROUND LIKELIHOOD APPROXIMATION FOR BANDED PROBABILITY DISTRIBUTIONS

E. Gjerløw[1], K. Mikkelsen[1], H. K. Eriksen[1], K. M. Górski[2,3], G. Huey[2], J. B. Jewell[2], S. K. Næss[1,4], G. Rocha[2,5], D. S. Seljebotn[1], and I. K. Wehus[2]
[1] Institute of Theoretical Astrophysics, University of Oslo, P.O. Box 1029 Blindern, N-0315 Oslo, Norway; eirik.gjerlow@astro.uio.no
[2] Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Drive, Pasadena, CA 91109, USA
[3] Warsaw University Observatory, Aleje Ujazdowskie 4, 00-478 Warszawa, Poland
[4] Department of Astrophysics, University of Oxford, Keble Road, Oxford OX1 3RH, UK
[5] California Institute of Technology, Pasadena, CA, USA
Received 2013 March 30; accepted 2013 September 17; published 2013 October 23

ABSTRACT

We investigate sets of random variables that can be arranged sequentially such that a given variable only depends conditionally on its immediate predecessor. For such sets, we show that the full joint probability distribution may be expressed exclusively in terms of uni- and bivariate marginals. Under the assumption that the cosmic microwave background (CMB) power spectrum likelihood only exhibits correlations within a banded multipole range, $\Delta\ell_C$, we apply this expression to two outstanding problems in CMB likelihood analysis. First, we derive a statistically well-defined hybrid likelihood estimator, merging two independent (e.g., low- and high-ℓ) likelihoods into a single expression that properly accounts for correlations between the two. Applying this expression to the *Wilkinson Microwave Anisotropy Probe* (*WMAP*) likelihood, we verify that the effect of correlations on cosmological parameters in the transition region is negligible in terms of cosmological parameters for *WMAP*; the largest relative shift seen for any parameter is 0.06σ. However, because this may not hold for other experimental setups (e.g., for different instrumental noise properties or analysis masks), but must rather be verified on a case-by-case basis, we recommend our new hybridization scheme for future experiments for statistical self-consistency reasons. Second, we use the same expression to improve the convergence rate of the Blackwell–Rao likelihood estimator, reducing the required number of Monte Carlo samples by several orders of magnitude, and thereby extend it to high-ℓ applications.

Key words: cosmic background radiation – cosmology: observations – methods: data analysis – methods: numerical – methods: statistical

Online-only material: color figures

1. INTRODUCTION

The cosmic microwave background (CMB) radiation is one of the most pristine sources of information about the early universe available to us. Since its discovery in 1964 (Penzias & Wilson 1965), the amount of information available to us about the CMB has increased at a rapid pace through series of ground-based, sub-orbital and satellite experiments. The recently released *Planck* temperature sky maps (Planck I 2013) is just the latest example of how the present challenge in the field of cosmology is one of overabundance rather than shortage of data.

To extract cosmological parameters from these ever growing data sets requires increasingly sophisticated and efficient algorithms, both due to larger data volumes and to more stringent requirements to statistical precision. For example, the *COBE*-DMR sky maps published 20 yr ago (Smoot et al. 1992) comprised $\mathcal{O}(10^4)$ pixels, and could be analyzed using exact brute-force likelihood techniques (e.g., Górski 1994), with a computational scaling of $\mathcal{O}(N_{\rm pix}^3)$. The *Wilkinson Microwave Anisotropy Probe* (*WMAP*) sky maps published 10 yr ago comprised $\mathcal{O}(10^7)$ pixels (Bennett et al. 2003a), at which point faster and approximate methods had to be used for parameter estimation (Hivon et al. 2002; Verde et al. 2003). However, for *WMAP* the error budget was still dominated by cosmic variance on large angular scales and instrumental noise on small angular scales, and confusion with Galactic and extra-Galactic emission was minimal, allowing for very simple component separation methods (Bennett et al. 2003b; Hinshaw et al. 2003). For *Planck*, the total number of data points in nine frequency bands is $\mathcal{O}(3\times10^8)$, and instrumental noise never dominates the uncertainties at any angular scales, as small-scale astrophysical confusion becomes important at multipoles $\ell\gtrsim1500$ (Planck XII 2013). As a result, an unprecedented study of all important sources of uncertainty, including instrumental, systematic and astrophysical, was required for *Planck* to reach its ambitious goals (Planck XV 2013).

With the advent of these massive mega-pixel data sets, a number different analysis strategies have been developed to robustly extract cosmological parameters with acceptable computational cost. As of today, the preferred option for full-sky high-resolution experiments such as *Planck* and *WMAP* is to divide the analysis into two separate components according to large and small angular scales, and merge the two at the likelihood level. On large angular scales, they use a Gibbs sampling based (Jewell et al. 2004; Wandelt et al. 2004; Eriksen et al. 2004) Blackwell–Rao (BR) estimator (Chu et al. 2005) that takes into account the full non-Gaussian structure of the true CMB likelihood, while on small angular scales, they use faster approaches (e.g., Hivon et al. 2002; Rocha et al. 2011; Planck XV 2013) coupled to an analytic multivariate Gaussian (and/or log-normal) likelihood approximation. The computational cost of this hybrid approach is dominated by spherical harmonics transforms, and therefore scales as $\mathcal{O}(N_{\rm pix}^{3/2})$, which is acceptable even for large data sets. However, there is an unsolved problem

associated with this hybrid approach, and that is how to merge the two likelihood components into a single all-scale expression; correlations between the smallest scales in the large-scale likelihood and the largest scales in the small-scale likelihood should in principle be accounted for. As of today, no fully satisfactory solution to this exists in the CMB literature, though various approaches were explored during the Planck analysis.

Having a computational scaling of $\mathcal{O}(N_{\rm pix}^{3/2})$, the Gibbs sampling approach could in principle be employed for all angular scales, thus eliminating the need for any hybrid approximation. Unfortunately, in practice this method is in its current implementation limited to low angular scales for two reasons: First, joint CMB analysis and component separation is currently implemented in terms of pixel-based fits of physical foreground models, requiring all frequency bands to have the same angular resolution, dictated by the coarsest resolution in a given data set. Second, although the computational scaling for the Gibbs sampler is acceptable, the prefactor is high. The 2013 *Planck* likelihood employed 100,000 Gibbs samples in order to achieve robust BR convergence, and each of those samples required ∼2000 Conjugate Gradient iterations (and twice as many spherical harmonic transforms) to converge, for a total cost of 500,000 CPU hours. Naively scaling this to full *Planck* resolution suggest a final cost of $\mathcal{O}(10^8)$ CPU hours (only taking into account the additional computational cost per sample for high-resolution Gibbs sampling, not the additional number of samples needed for the BR estimator to converge at higher multipoles).

The main result of the present paper is a statistically well motivated block factorization of the CMB power spectrum likelihood that is applicable to several of these problems. Specifically, we show that for sets of random variables that can be arranged sequentially in such a way that all correlations have a finite range within the sequence, the full joint probability distribution may be written in terms of lower-dimensional marginals. The archetypal example of such a distribution is a multivariate Gaussian with a strictly banded covariance matrix, and we therefore call the general (non-Gaussian but conditionally limited) case also "banded." With this statistical identity ready at hand, we first suggest a statistically well-motivated likelihood hybridization scheme that takes properly into account correlations between the low- and high-ℓ regimes, and, second, we show how the convergence rate of the BR estimator can be improved by factorizing the full high-dimensional multivariate posterior into a set of lower-dimensional distributions, each of which converges much faster than the full distribution. This approach differs from the direct Gaussianization technique proposed by Rudjord et al. (2009) in that the underlying probabilistic structure (e.g., shapes of marginal and N-point correlations) is conserved; in principle, the only modification to the full likelihood enforced by our new approach is that assumed negligible correlations are explicitly set to zero.

2. FACTORIZING THE CMB LIKELIHOOD

2.1. Factorization of Banded Probability Distributions

We begin with a general joint probability density $P(\{\theta\}) = P(\theta_1, \theta_2, \theta_3, \ldots, \theta_n)$ for a set of random variables, θ_k, with $k = 1, 2, 3, \ldots, n$. We choose one specific sequential ordering of these variables (out of all the possible orderings), and use the definition of a conditional to write the joint distribution as a product of univariate conditionals,

$$\begin{aligned} P(\{\theta\}) = &\quad P(\theta_1, \theta_2, \theta_3, \ldots, \theta_n) \\ = &\quad P(\theta_1|\theta_2, \theta_3, \ldots, \theta_n) \\ &\cdot P(\theta_2|\theta_3, \ldots \theta_n) \cdots \\ &\cdot P(\theta_{n-1}|\theta_n) \cdot P(\theta_n). \end{aligned}$$

We then assume that our variables only have a conditional probability dependence on their immediate neighbors in the sequence, i.e., that the probability distribution is *tri-diagonally* banded,

$$\begin{aligned} P(\{\theta\}) \approx &\; P(\theta_1|\theta_2) \cdot P(\theta_2|\theta_3) \cdots P(\theta_{n-1}|\theta_n) \cdot P(\theta_n) \\ = &\; \frac{P(\theta_1, \theta_2)}{P(\theta_2)} \cdot \frac{P(\theta_2, \theta_3)}{P(\theta_3)} \cdots \frac{P(\theta_{n-1}, \theta_n)}{P(\theta_n)} \cdot P(\theta_n) \\ = &\; \frac{\prod_{k=1}^{n-1} P(\theta_k, \theta_{k+1})}{\prod_{k=2}^{n-1} P(\theta_k)}. \end{aligned} \tag{1}$$

Thus, this simple derivation shows that a strictly (tri-diagonally) banded probability distribution may be factorized recursively into a product of uni- and bivariate marginals.

Before applying this expression to CMB likelihood approximation, we note that even if the joint probability distribution do not have correlations exclusively between neighboring variables, it may still be possible to factorize it, provided at least some correlations may be ignored. For instance, suppose we can ignore all but the nearest *two* neighbors; in that case, the joint distribution will factorize into a product of uni-, bi- and trivariate marginals.

2.2. Block Factorization of the CMB Likelihood

In its most basic representation, a CMB data set, $\mathbf{d}$, may be modeled as

$$\mathbf{d} = \mathbf{s} + \mathbf{n}, \tag{2}$$

where $\mathbf{s}$ is the true sky signal and $\mathbf{n}$ represents instrumental noise. Both the signal and noise are usually assumed to be zero-mean Gaussian variables with covariances $\mathbf{S}$ and $\mathbf{N}$, respectively.

The noise covariance matrix is typically given by external knowledge about the instrumental noise characteristics and the scanning strategy of a given experiment. The signal covariance matrix, on the other hand, is generally unknown, and must be estimated from the data. However, given the fact that we only have one observable sky available, it is impossible to estimate the $N_{\rm pix}^2$ elements in $\mathbf{S}$ from the $N_{\rm pix}$ elements in $\mathbf{d}$ without imposing strong priors on its structure. The most commonly accepted prior is simply that the CMB sky is isotropic and homogeneous (e.g., Planck XXIII 2013). It is therefore convenient to expand $\mathbf{s}$ in spherical harmonics, such that

$$\mathbf{s}(\hat{\mathbf{n}}) = \sum_{\ell,m} a_{\ell m} Y_{\ell m}(\hat{\mathbf{n}}), \tag{3}$$

where $\hat{\mathbf{n}}$ is a unit vector pointing to a given position on the sky, $Y_{\ell m}$ are the spherical harmonics, and $a_{\ell m}$ are the corresponding spherical harmonics coefficients. Then the signal covariance matrix may be written as

$$S_{\ell m, \ell' m'} = \langle a_{\ell m} a^*_{\ell' m'} \rangle \equiv C_\ell \delta_{\ell \ell'} \delta_{m m'}, \tag{4}$$

where C_ℓ is known as the angular power spectrum.

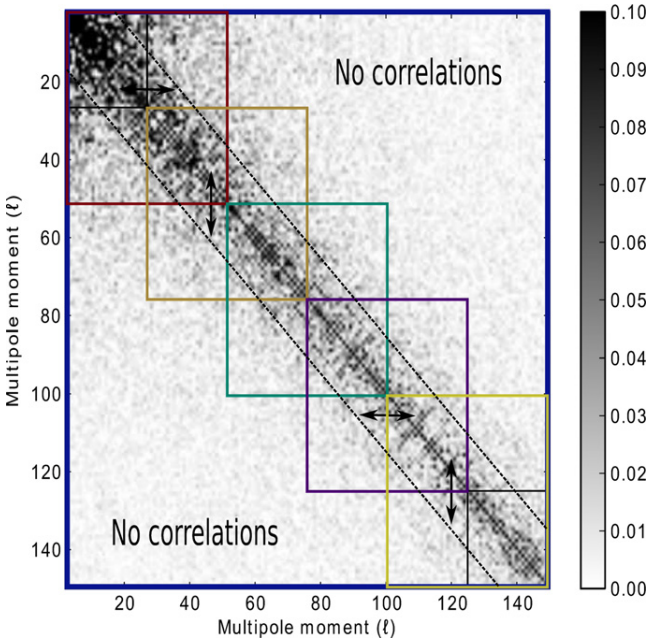


Figure 1. Angular power spectrum correlation matrix, $M_{\ell\ell'}$, for the official *Planck* low-ℓ CMB data set, estimated by Monte Carlo sampling. Note that any two-point correlations are contained within a band of $\Delta\ell_C \sim 15$, suggesting that the CMB likelihood may be approximated as a banded probability distribution. To factorize the CMB likelihood into lower-dimensional elements, we partition the full multipole range into a set of disjoint blocks such that all non-zero covariance elements are embedded within a tri-diagonal block structure, indicated here by colored squares.

(A color version of this figure is available in the online journal.)

The main goal of most CMB experiments is precisely to measure the CMB power spectrum, and the most straightforward way to do so is by maximum-likelihood estimation. Since we have assumed that both signal and noise are Gaussian distributed, the CMB power spectrum likelihood simply reads

$$\mathcal{L}(C_\ell) \equiv P(\mathbf{d}|C_\ell) \propto \frac{e^{-\frac{1}{2}\mathbf{d}^t(\mathbf{S}(C_\ell)+\mathbf{N})^{-1}\mathbf{d}}}{\sqrt{|\mathbf{S}(C_\ell)+\mathbf{N}|}}, \tag{5}$$

where $\mathbf{S} = \mathbf{S}(C_\ell)$ is the covariance matrix given in Equation (4) expressed in pixel domain. Note that C_ℓ denotes the set of all power spectrum coefficients, and the likelihood therefore spans an $\ell_{\rm max}$-dimensional space.

As already mentioned, brute-force evaluation of Equation (5) scales computationally as $\mathcal{O}(N_{\rm pix}^3)$, and is therefore feasible only for very low angular resolutions. Much of the CMB analysis literature therefore revolves around finding computationally tractable approximations to this expression.

In order to build up some intuition about the correlation structure of $\mathcal{L}(C_\ell)$, it is useful to plot the correlation matrix

$$M_{\ell\ell'} \equiv \frac{\langle (C_\ell - \langle C_\ell\rangle)(C_{\ell'} - \langle C_{\ell'}\rangle\rangle}{\sqrt{\langle (C_\ell - \langle C_\ell\rangle)^2\rangle \langle (C_{\ell'} - \langle C_{\ell'}\rangle)^2\rangle}}. \tag{6}$$

Figure 1 shows this matrix for the official *Planck* low-ℓ CMB data, as evaluated from 200,000 Monte Carlo samples generated with a CMB Gibbs sampler (Jewell et al. 2004; Wandelt et al. 2004; Eriksen et al. 2004). In this case, there are significant correlations between all elements at $\ell \lesssim 20$, while at $\ell \gtrsim 50$ any correlations are well contained inside a band of $\Delta\ell_C = 15$; any correlations beyond $\Delta\ell_C \gtrsim 30$ are well below 1%. Higher-order correlations are significantly smaller than these two-point correlations.

For typical sky cuts and instrumental noise characteristics, the basic CMB likelihood can therefore be approximated as a banded probability distribution with a bandwidth of $\ell \lesssim 15$, and can therefore in principle be factorized by Equation (1). However, as currently written this expression only applies to a strictly tri-diagonal covariance matrix. To circumvent this problem, we therefore introduce an auxiliary block structure that embeds all non-negligible elements within a larger tri-diagonal structure, as illustrated by the colored blocks in Figure 1. That is, we define a set of multipole blocks such that $\theta_1 = \{C_{\ell_{\rm min}}, \ldots, C_{\ell_1}\}$, $\theta_2 = \{C_{\ell_1+1}, \ldots, C_{\ell_2}\}, \ldots, \theta_n = \{C_{\ell_{n-1}+1}, \ldots, C_{\ell_{\rm max}}\}$. Thus, each univariate marginal in Equation (1) is replaced with a multivariate distribution of dimension $\ell_i - \ell_{i-1}$, and each bivariate marginal is replaced with a multivariate distribution of dimension $\ell_i - \ell_{i-2}$. This block-wise factorization constitutes the main result of this paper, and in the following sections we will apply this to two concrete problems in CMB likelihood estimation.

3. ACCURATE HYBRID CMB LIKELIHOOD ESTIMATION

As already mentioned, both *Planck* and *WMAP* have adopted so-called "hybrid" likelihood approximations, combining a Gibbs sampling based BR estimator at large angular scales with a Gaussian (and/or log-normal) pseudo cross-spectrum approximation at small angular scales. These two components are merged into a single expression at the log-likelihood level. The *Planck* likelihood simply adds the two log-likelihoods (Planck XV 2013), adopting a so-called "sharp transition" between the low- and high-ℓ regimes, schematically illustrated in the left panel of Figure 2. This is the simplest possible approach, and assumes that any correlations across the transition multipole are negligible. The *WMAP* likelihood makes a different choice, by including the off-diagonal terms between the low- and high-ℓ blocks in the (Gaussian plus log-normal) high-ℓ likelihood, as illustrated in the middle panel of Figure 2.

In this section, we introduce a new and statistically better motivated approach than either of two employed by *Planck* and *WMAP*, taking advantage of the block factorization derived in Equation (1). The first step in our approach is to partition the full multipole range between $\ell_{\rm min}$ and $\ell_{\rm min}$ into three disjoint regions, $L = \{\ell_{\rm min}, \ldots, \ell_{\rm low}\}$, $T = \{\ell_{\rm low}+1, \ldots, \ell_{\rm high}-1\}$ and $H = \{\ell_{\rm high}, \ldots, \ell_{\rm max}\}$, corresponding to a low-ℓ region, a transition region and a high-ℓ region, respectively. The width of the transition region is chosen to be at least as wide as the effective bandwidth of the C_ℓ covariance matrix (see Figure 1). With this partitioning, we now specialize Equation (1) to the case with $n = 3$ regions:

$$\log\mathcal{L}(C_\ell) = \log\mathcal{L}(L,T) + \log\mathcal{L}(T,H) - \log\mathcal{L}(T). \tag{7}$$

Note that this approximation is exact under the assumption of vanishing correlations between the low- and high-ℓ regions, which can be ensured simply by letting the transition region be sufficiently wide. This estimator is schematically illustrated in the right panel of Figure 2.

Equation (7) has a simple intuitive interpretation: The log-likelihood is simply the sum of a low- and a high-ℓ contribution, defined such that they overlap over a sufficiently wide multipole range that all non-negligible correlations are included. However, because the diagonal block in the transition region

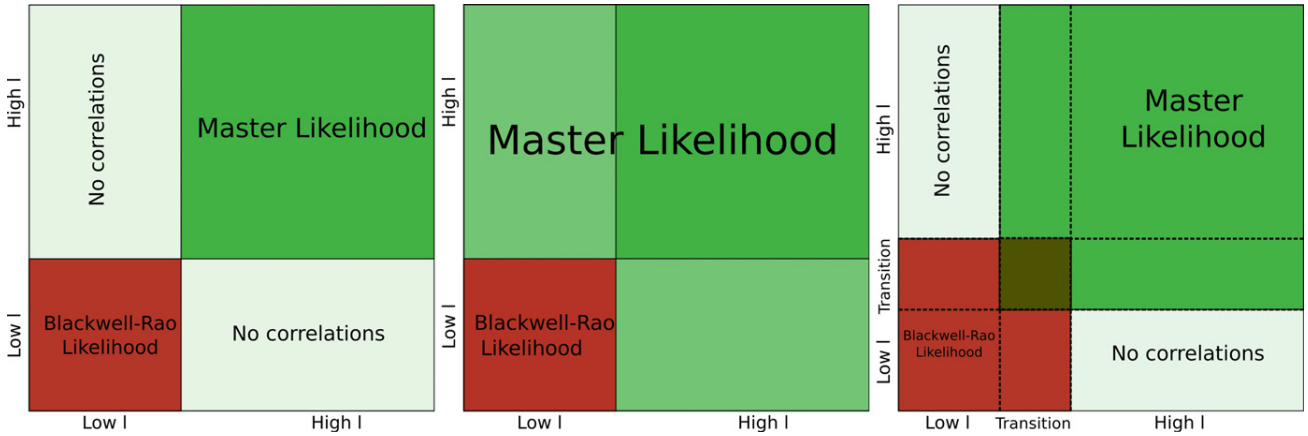


Figure 2. Schematic overview of the three hybridization schemes discussed in the text. The left panel illustrates a sharp transition between the low- (Blackwell–Rao) and high-ℓ (MASTER) likelihood, as currently adopted by *Planck*. The middle panel illustrates the *WMAP* approach, which includes the off-diagonal elements between the low- and high-ℓ regions in the high-ℓ likelihood estimator. The right panel illustrates the new estimator proposed in this paper, in which correlations are accounted for through an transition region that is sufficiently wide to include all non-negligible correlations between the low- and high-ℓ regions. To avoid double-counting of the diagonal elements, the total log-likelihood is corrected by the log-likelihood including elements within the transition region only.

(A color version of this figure is available in the online journal.)

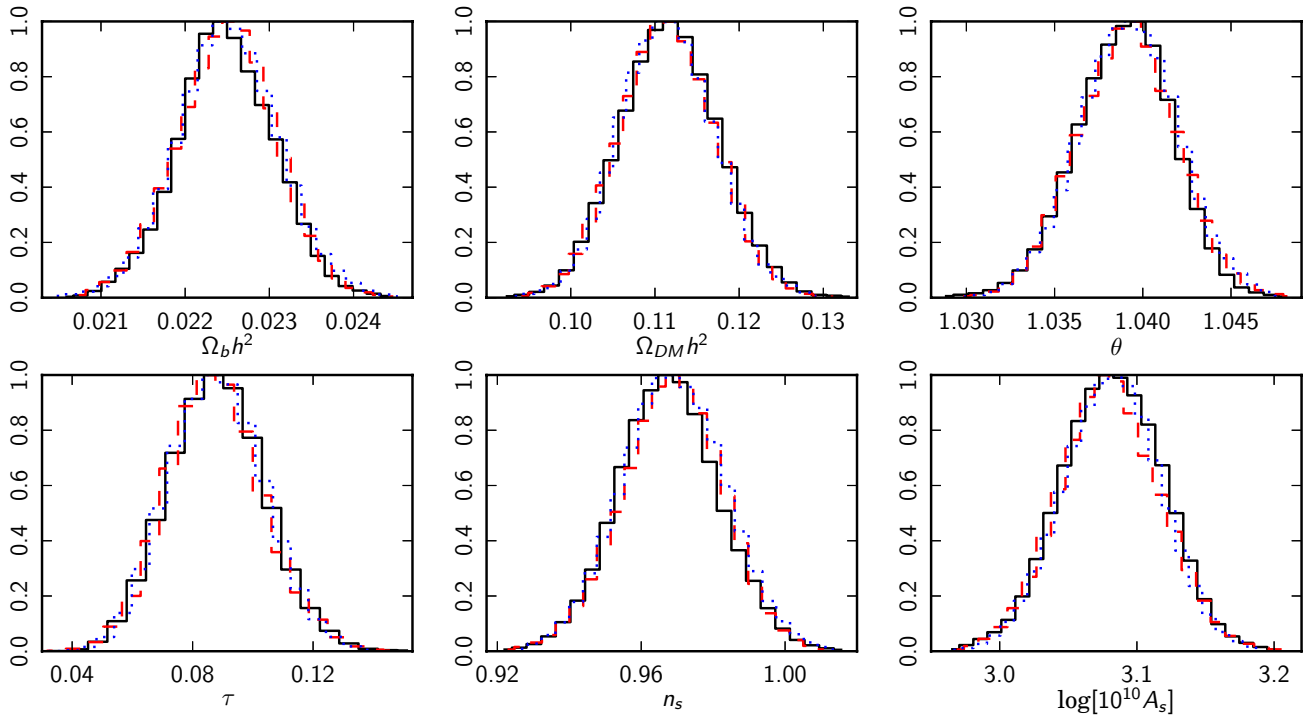


Figure 3. Comparison of best-fit parameters derived by CosmoMC from *WMAP* using likelihood approximations based on the new hybrid estimator presented in this paper (solid black line); the *WMAP* approach including off-diagonal elements in the inverse covariance matrix (dashed red line); and a sharp transition between the low- and high-ℓ regions (dotted blue line).

(A color version of this figure is available in the online journal.)

is included twice, both by the low- and the high-ℓ likelihood, one must subtract the corresponding marginal for the transition region once to avoid double-counting (this is also an immediate consequence of Equation (1), under the assumption that $p(L|T, H) = p(L|T) = p(L, T)/P(T)$, i.e., the low-$\ell$ region is conditionally independent of the high-ℓ region given the transition region). Note that any estimator for the transition likelihood may be used for the correction term, typically by extracting the relevant range from either the low- or the high-ℓ likelihoods.

To assess the importance of the specific strategy adopted for hybridization, we modify the (7 yr) *WMAP* likelihood to include each of the three solutions, and derive constraints on the standard ΛCDM model using *WMAP* data only. The transition multipole is set to $\ell_{\rm trans} = 32$ for the sharp transition case, whereas the transition region is defined as $\ell = \{21, \ldots, 32\}$ for the new hybrid scheme. The *WMAP* BR estimator is used both for the low-ℓ and the transition regions in the latter case. We adopt $\Omega_b h^2$, $\Omega_m h^2$, θ, τ, n_s, and $\log(10^{10} A_s)$ as our primary parameters, and adopt CosmoMC (Lewis & Bridle 2002) as our MCMC engine. The resulting one-dimensional marginals are shown in Figure 3 for all three cases, and posterior mean summary statistics are given in Table 1.

With a largest relative difference between any two cases of 0.06σ, these results demonstrate that the standard six-parameter

Table 1
Summary of Cosmological Parameters Derived with Three Different Hybridization Schemes

	Default *WMAP*	Sharp Transition		Transition Region	
	Constraint	Constraint	Deviation (σ)	Constraint	Deviation (σ)
$\Omega_b h^2$	0.0225 ± 0.0006	0.0225 ± 0.0006	0.02	0.0225 ± 0.0006	0.02
$\Omega_m h^2$	0.111 ± 0.005	0.111 ± 0.005	0.01	0.112 ± 0.006	0.05
θ	1.039 ± 0.003	1.039 ± 0.003	0.04	1.039 ± 0.003	0.05
τ	0.088 ± 0.015	0.088 ± 0.015	0.04	0.088 ± 0.015	0.05
n_s	0.969 ± 0.013	0.969 ± 0.014	0.03	0.968 ± 0.014	0.06
$\log[10^{10} A_s]$	3.08 ± 0.04	3.08 ± 0.03	0.03	3.08 ± 0.04	0.05

Notes. The three hybridization schemes are as follows. The original *WMAP* approach including off-diagonal elements in the inverse covariance matrix (second column), a sharp transformation at $\ell_{\rm trans} = 32$ (third column), and the new approach implementing a transition region between $\ell = 21$ and 32 (fifth column). The confidence intervals are 1 σ, and the best-fit points are the marginalized means of the parameters. The fourth and sixth columns show the relative shifts with respect to the *WMAP* approach measured in units of σ.

ΛCDM model is highly robust with respect to assumptions about the correlations across the transition regime. Similar conclusions were found when performing an identical analysis for the recently released *Planck* likelihood (Planck XV 2013), and this motivated the choice of a sharp transition for that particular implementation. For future experiments and analyses we nevertheless recommend the hybrid approach presented here, for two main reasons. First, our expression provides a statistically well motivated solution whose validity may be monitored directly through the C_ℓ covariance matrix; without the same level of statistical rigor, detailed simulations are more critical for the other two approaches, and these should in principle be repeated both when the data set or the parametric model is changed. Second, this expression is implementationally trivial once both low- and high-ℓ likelihoods are available, and there is therefore no practical reason for not including these correlations, even if their impact may be small.

4. FASTER BLACKWELL–RAO CONVERGENCE

4.1. Review of the Blackwell–Rao Estimator

As mentioned in Section 1, both the *Planck* and *WMAP* low-ℓ likelihoods (Planck XV 2013; Hinshaw et al. 2013) employs a specific BR estimator to produce an accurate likelihood approximation that accounts for all correlations and non-Gaussian structures (Chu et al. 2005). The main advantages of this estimator are (1) computational speed, (2) implementational simplicity, and (3) support for seamless marginalization over systematic effects and component separation errors through Gibbs sampling (Jewell et al. 2004; Wandelt et al. 2004; Eriksen et al. 2004).

This estimator may be explained intuitively as follows: Suppose it is possible to construct an experiment that provides a perfect full-sky noiseless image of the CMB sky, $\mathbf{d} = \mathbf{s}$. For that experiment, the only source of uncertainty on C_ℓ is cosmic variance, and the exact CMB likelihood in Equation (5) reduces to an inverse Gamma distribution,

$$\mathcal{L}_0(C_\ell) \propto \frac{e^{-\frac{1}{2}\mathbf{s}^t \mathbf{S}(C_\ell)^{-1}\mathbf{s}}}{\sqrt{|\mathbf{S}(C_\ell)|}} \propto \prod_\ell \sigma_\ell^{-\frac{2\ell-1}{2}} \frac{e^{\frac{2\ell+1}{2}\frac{\sigma_\ell}{C_\ell}}}{C_\ell^{\frac{2\ell+1}{2}}}. \quad (8)$$

Here we have defined $\sigma_\ell \equiv (1/(2\ell+1)) \sum_{\ell=-m}^{m} |a_{\ell m}|^2$ to be the realization specific power spectrum of $\mathbf{s}$.

However, for any real experiment there are additional sources of uncertainty beyond cosmic variance, for instance from instrumental noise and foreground contamination, and $P(\mathbf{s}|\mathbf{d})$ is no longer a delta function. In order to account for this additional uncertainty, one must weight the ideal likelihood in Equation (8) with respect to $P(\mathbf{s}|\mathbf{d})$,

$$\mathcal{L}_{\rm BR}(C_\ell) = \int d\mathbf{s}\, \mathcal{L}_0(C_\ell)\, P(\mathbf{s}|\mathbf{d}). \quad (9)$$

At first glance, this integral appears difficult to evaluate, as it involves millions of degrees of freedom. However, this is precisely where the CMB Gibbs sampler enters the picture. As explained in detail by Jewell et al. (2004), Wandelt et al. (2004), and Eriksen et al. (2004), the output from this algorithm is a set of samples drawn directly from $P(\mathbf{s}|\mathbf{d})$, accounting for both instrumental noise and foreground errors. Thus, the integral can be simply evaluated by Monte Carlo integration as a sum over these samples,

$$\mathcal{L}_{\rm BR}(C_\ell) \approx \sum_{i=1}^{N_{\rm samp}} \prod_{\ell=\ell_{\rm min}}^{\ell_{\rm max}} {\sigma_\ell^i}^{\frac{2\ell-1}{2}} \frac{e^{\frac{2\ell+1}{2}\frac{\sigma_\ell^i}{C_\ell}}}{C_\ell^{\frac{2\ell+1}{2}}}. \quad (10)$$

This is the CMB power spectrum BR estimator, which is guaranteed to converge to the true likelihood in the limit of $N_{\rm samp} \to \infty$.

4.2. Lifting the "Curse of Dimensionality" by Block Factorization

While the BR estimator is guaranteed to converge to the correct answer, it is not obvious how fast it does so, as measured in terms of number of samples required for convergence, $N_{\rm samp}$. Further, since the computational cost of a single Gibbs sample is typically on the order of several CPU hours (Eriksen et al. 2004), depending on the angular resolution and/or signal-to-noise ratio of the data set under consideration, it is important to understand this scaling before attempting a full-scale analysis. Indeed, Chu et al. (2005) showed that $N_{\rm samp}$ scales exponentially with $\ell_{\rm max}$, effectively limiting its operational range to $\ell_{\rm max} \approx 50$–70. The main goal of the present section is to improve on this limit, and extend the BR estimator to high ℓ's.

To understand the origin of the exponential scaling, we show in Figure 4 a simple two-dimensional Gaussian distribution mapped by a Monte Carlo sampler. The top and left panels show the respective one-dimensional marginals. The BR estimator establishes a smooth approximation to these distribution by assigning a kernel of finite width to each individual Monte Carlo sample (illustrated by blue contours/Gaussians) before taking the average over all samples. Suppose now that the width of the one-dimensional kernel is 10% of the width of the marginal

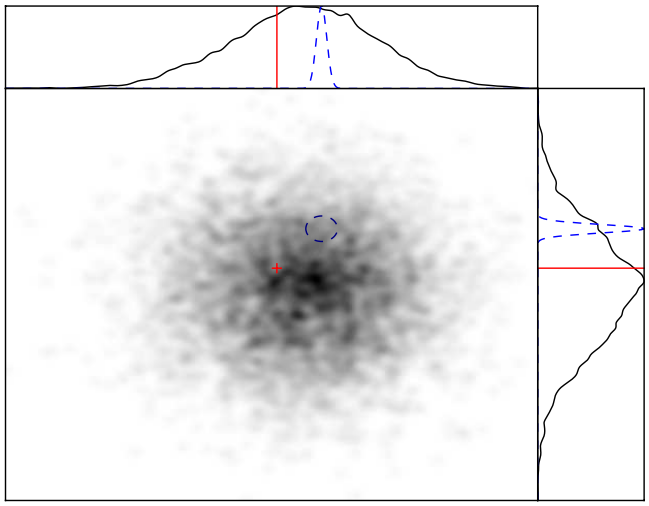


Figure 4. Illustration of the "curse of dimensionality." The Blackwell–Rao estimator builds up a smooth histogram from a finite set of Monte Carlo samples by assigning a distribution (or kernel) to each sample. The number of samples required to reach convergence is proportional to the ratio between the volume of the kernel (blue) and the volume of the full distribution (black). If this ratio is $r < 1$ in one dimension (top and left panels), it is r^2 in two dimensions (central panel), and r^n in n dimensions. This implies that the number of Monte Carlo samples required to reach convergence for the CMB BR estimator scales exponentially with $\ell_{\rm max}$. The evaluation of the two-dimensional likelihood at a specific point in parameter space (red cross) will be much more sensitive to the number of samples than the corresponding evaluations in the respective marginalized parameter spaces (red lines).

(A color version of this figure is available in the online journal.)

distribution; in that case, one needs ~ 10 samples in order to cover the marginal once. In two dimensions, however, one needs $\sim 10^2$ samples to cover the full joint distribution once, since the ratio now is only 10% in each of the two directions. More generally, in n dimensions one would need $\sim 10^n$ samples. This is a variation of the well-known "curse of dimensionality," which says that the number of points required to cover an n-dimensional space scales exponentially with n.

The BR estimator given in Equation (10) converges well up to $\ell \approx 30$ with only a few thousand samples for *WMAP* (Chu et al. 2005), while for *Planck* it is found to be robust up to $\ell \approx 70$ with 100,000 samples (Planck XV 2013). To extend to even higher ℓ's by brute force would soon require a prohibitively large number of samples, as the computational cost for the Gibbs sampling step of the latter case is already half a million CPU hours.

Fortunately, the block factorization presented in Section 2 may be used to define an alternative and computationally much cheaper algorithm.

1. Partition the full $\ell_{\rm max}$-dimensional $\mathcal{L}(C_\ell)$ into a sequence of lower-dimensional blocks, r_k. Here, we take the blocks to be of the same width, which we call $\Delta\ell$.
2. Use the standard BR estimator to estimate the marginal likelihood for each block and each neighboring set of two blocks.
3. Merge these block marginals into a single all-ℓ estimator through the block factorization in Equation (1).

Thus, our new likelihood approximation can be written succinctly on the following form,

$$\mathcal{L}(C_\ell) \approx \frac{\prod_{k=1}^{n-1} \mathcal{L}_{\rm BR}(r_k, r_{k+1})}{\prod_{k=2}^{n-2} \mathcal{L}_{\rm BR}(r_k)}. \tag{11}$$

Note that all the likelihood evaluations on the right side of this expression involve a maximum of $2\Delta\ell - 1$ dimensions, as opposed to $\ell_{\rm max} - \ell_{\rm min} + 1$ for the full joint BR estimator, effectively lifting the curse of dimensionality.

4.3. Accuracy and Convergence

4.3.1. Methodology

Before the block factorized BR estimator can be used for real analysis, it is necessary to assess its accuracy and convergence properties. To this aim, we analyze two different simulations with the above machinery, adopting the convergence analysis methodology of Chu et al. (2005). We use this to perform various tests which will be reported in the "Results" section. Monte Carlo samples are produced with Commander (Eriksen et al. 2004, 2008).

The first simulation consists of a full-sky high-resolution ($N_{\rm side} = 512$, $\ell_{\rm max} = 1024$, $14'$ Gaussian beam) data set with uniform noise (65 μK rms per pixel). The main advantage of this case is that the C_ℓ likelihood (Equation (5)) factorizes in ℓ, and can be evaluated analytically,

$$\mathcal{L}_{\rm ideal}(C_\ell) \propto \prod_\ell \frac{e^{-\frac{2\ell+1}{2}\frac{\hat{\sigma}_\ell}{(C_\ell+N_\ell)}}}{(C_\ell + N_\ell)^{\frac{2\ell+1}{2}}}, \tag{12}$$

where $\hat{\sigma}_\ell$ is the angular power spectrum of the noisy sky map, and N_ℓ is the ensemble averaged noise power spectrum. The second simulation consists of a low-resolution ($N_{\rm side} = 32$, $\ell_{\rm max} = 95$, 6° FWHM Gaussian beam) data set with various sky masks imposed. White noise of 0.3 μK rms is added to each pixel, resulting in a signal-to-noise of unity at $\ell \approx 70$. The main purpose of this simulation is to study the effect of correlations between different multipoles arising from the sky cut through comparison with brute-force pixel-space likelihood evaluation. However, because of the brute-force evaluations, this case is necessarily limited to low angular resolution.

The CMB signal is drawn from a Gaussian distribution with a covariance given by the best-fit *WMAP* ΛCDM power spectrum, $C_\ell^{\rm ref}$ (Hinshaw et al. 2013). In each case, we fit a two-parameter amplitude-tilt (A–n) model on the form

$$C_\ell(A, n) = A \left(\frac{\ell}{\ell_0}\right)^n C_\ell^{\rm ref}, \tag{13}$$

where $\ell_0 = \ell_{\rm max}/2$, simply by mapping out $\mathcal{L}(A, n)$ over a two-dimensional grid. For $\ell_{\rm min} = 2$, this choice of pivot multipole ensures a low degree of correlation between A and n.

To assess both convergence and accuracy, we adopt the following measure of difference between two likelihoods, $\mathcal{L}_1$ and $\mathcal{L}_2$ (Chu et al. 2005),

$$q = \int |\mathcal{L}_1(A, n) - \mathcal{L}_2(A, n)|\, dA\, dn. \tag{14}$$

One can show that if $\mathcal{L}_1$ and $\mathcal{L}_2$ are two bivariate Gaussian distributions with the same covariance matrix, $\mathbf{\Sigma}$, but different means, $\boldsymbol{\mu}_1$ and $\boldsymbol{\mu}_2$, then

$$q = \Phi\left(\tfrac{1}{2\sqrt{2}}\sqrt{(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)\mathbf{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)}\right), \tag{15}$$

where Φ is the cumulative standard normal distribution function. From this, one finds that a 0.1σ shift in a Gaussian distribution

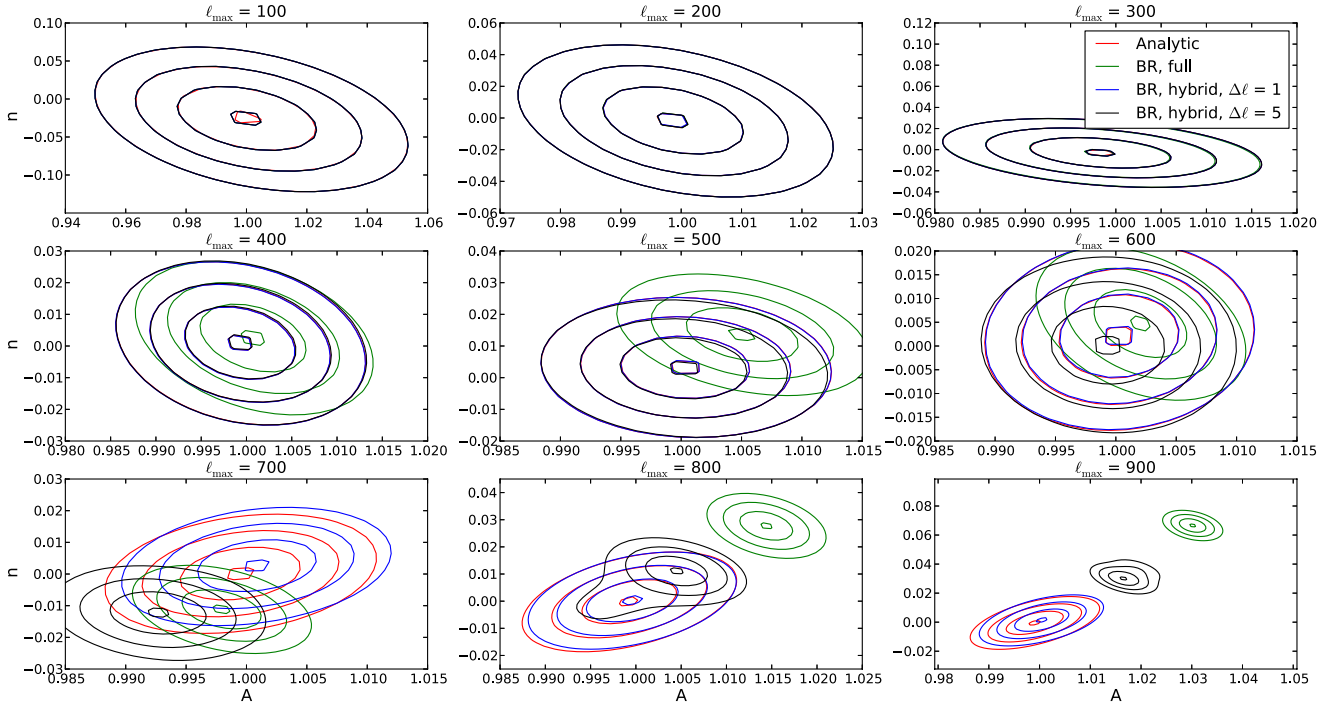


Figure 5. Comparison of four different methods of evaluating a simple amplitude-tilt likelihood for a full-sky simulation: The analytic case, the full Blackwell–Rao case, and two versions of the hybrid likelihood described in this paper—with $\Delta\ell = 1$ and 5, respectively.

corresponds to $q \sim 0.05$. In the following, we therefore define two distributions to agree if $q < 0.05$.

For the accuracy assessment, we simply compare the block factorized BR likelihood with the exact case. Convergence assessment, however, is done by drawing two disjoint sample subsets from the full set of available Monte Carlo samples, compute the BR estimator from each subset, and compare the resulting likelihoods. We then increase the number of samples in the two subsets, $N_{\rm samp}$, until q is consistently lower than 0.05 even when adding 100 additional samples; the latter criterion is imposed in order to avoid chance agreement. Finally, we repeat this calculation a certain number of times with different sample subsets (but drawn from the same full sample set), and report the median of the resulting values of $N_{\rm samp}$ as the final estimate of the number of samples required for convergence.

4.3.2. Results

Figure 5 shows $\mathcal{L}(A, n)$ evaluated from the high-resolution full-sky simulation for nine different values of $\ell_{\rm max}$ with four different likelihood expressions; analytic, standard BR, and two variations of the block-factorized BR estimator. A total of $N_{\rm samp} = 28{,}000$ samples are included in the two latter, a choice that is set to highlight the fundamental difference between the various cases. In particular, since there are no correlations between any multipoles in this case, all four approaches are in principle exact, and the only difference among the four cases are their relative convergence rates.

For $\ell_{\rm max} \leqslant 300$, we see that all four estimators agree to very high accuracy. However, from $\ell_{\rm max} \geqslant 400$ the full-range BR likelihood away starts to diverge from the others. At $\ell_{\rm max} = 900$, it is separated from the analytic result by more than 15σ. In this case, the sum in Equation (10) is strongly dominated by the one sample that happens to have the lowest power spectrum scatter about some best-fit mode, and the resulting distribution is simply an imprint of the cosmic variance kernel (Equation (8)) for that sample.

The block factorized BR estimators remain valid to higher $\ell_{\rm max}$, demonstrating how the "curse of dimensionality" is lifted by breaking the full parameter space into smaller regions that are easier to handle. In particular, the case with $\Delta\ell = 1$ agrees with the analytic case even at $\ell_{\rm max} = 900$ to $\sim 0.3\sigma$.

Next, in the top panel of Figure 6 we plot the number of samples required for convergence according to the above criterion for the high-resolution full-sky simulation described above, and in the bottom panel we show the same, but after applying the *WMAP* mask, covering 25% of the sky, in order to introduce a realistic multipole correlation structure. The upper vertical limit in these plots is set by the finite number of samples included in the analysis.

In all cases we see the same qualitative behavior. Reducing the dimensionality of the BR estimator through block factorization greatly improves the convergence rate by reducing the required number of samples by orders of magnitude at high ℓ's. For instance, for the full-sky case and with a block size of $\Delta\ell = 6$, only 10^3 samples are required in order to reach convergence up to $\ell_{\rm max} = 500$, whereas the full BR estimator would require 10^6. For the 25% *WMAP* mask, about 10^4 samples are required for $\ell_{\rm max} = 200$, while it is difficult to establish any sensible estimate for the full BR estimator in this case. (Note that the high-ℓ projection for the latter case, marked by a dashed line, is based on linear extrapolation from a few low-ℓ points, since convergence was not reached at all within the current sample set at higher multipoles. This projection is therefore associated with a very large systematic uncertainty.)

Finally, in Figure 7 we illustrate how the robustness of the split BR estimator depends on the block size chosen, for a fixed number of samples. We use a symmetric mask, shown in Figure 8, which covers a 20 deg strip of the galactic center. It is close in extent to the COBE mask. We then apply the

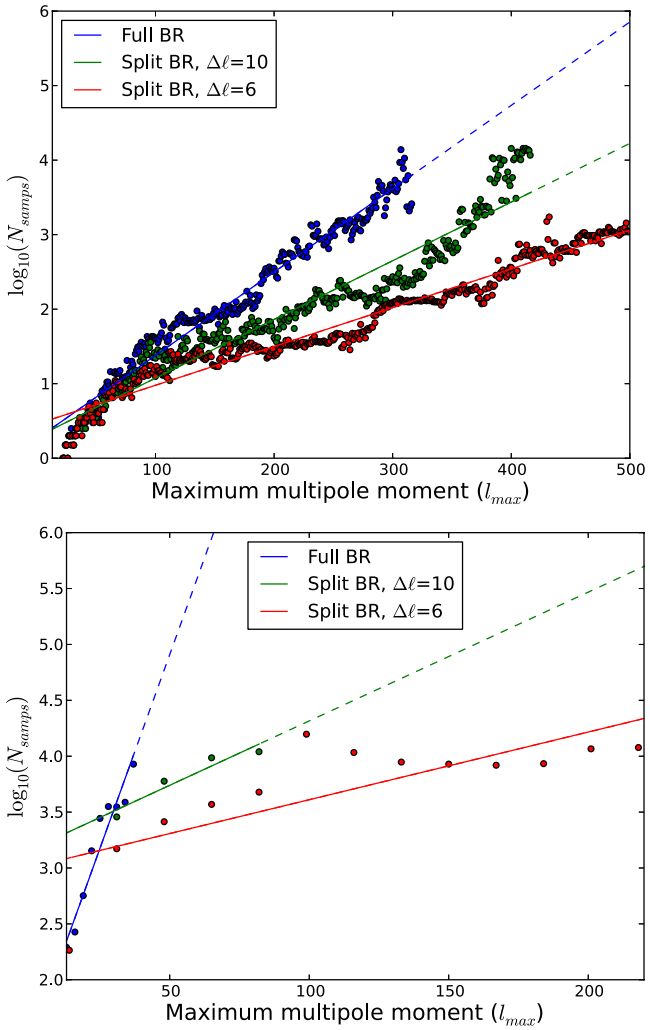


Figure 6. Convergence analysis for the split Blackwell–Rao estimator, with convergence defined in Section 4.3. The samples come from running Commander on a full-sky simulation. We show the median of the number of samples needed for convergence for a given $\ell_{\rm max}$, along with the best-fit regression line in $\log_{10}$-space. The median is computed from 10 (top) and 1024 (bottom) runs where the samples are scrambled between each run. The regression lines are dotted when they extend past the available data points. The high number of runs per data point for the bottom plot is also the reason for the more sparse sampling—each data point represented a very high computational cost, and so the number of data points were reduced.

BR estimator, using 30,000 Gibbs samples based on the low-resolution simulation. We also use the pixel-based estimator and calculate the q convergence between the BR result and the pixel-based result. Varying $\Delta\ell$, we the get a sense of how this block size affects the result. For this likelihood evaluation, we used $\ell_{\rm max} = 40$. A $\Delta\ell = 0$ on the plot means that the full BR estimator was used.

Conceptually, there should be two effects going on in this plot that both serve to reduce accuracy: When $\Delta\ell$ is too small, the between-multipole correlations set up by the mask are not modeled well enough, whereas when $\Delta\ell$ is too large, the parameter space becomes so large as to reduce accuracy. We can see this effect in play in the plot, although these data points are only indicative, not conclusive or exhaustive in any way. The specific block size dependence will typically depend on both the actual CMB signal and the morphology of the mask, and so prior testing like the methods outlined above should be performed before deciding on the optimal block size.

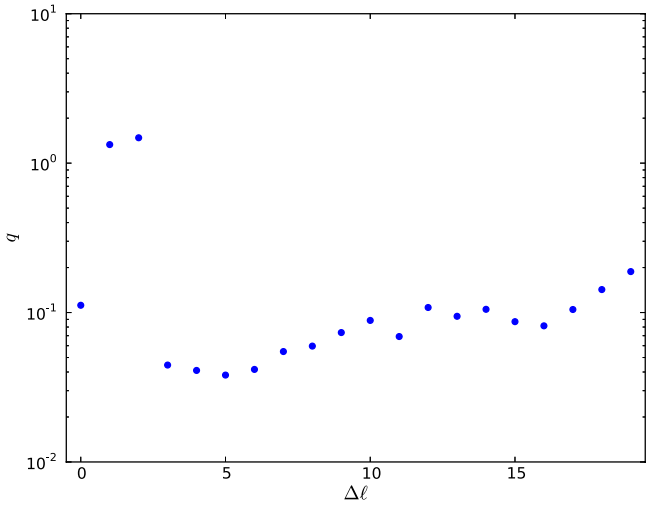


Figure 7. q-statistic (defined in the text) calculated for various modes of the Blackwell–Rao estimator compared with a pixel-based evaluation. The mask used is shown in Figure 8. A $\Delta\ell = 0$ means that the full BR estimator was used, while for $\Delta\ell > 0$ we have used the split BR estimator with a block size of $\Delta\ell$. The $\ell_{\rm max}$ used in the likelihood evaluation is 40.

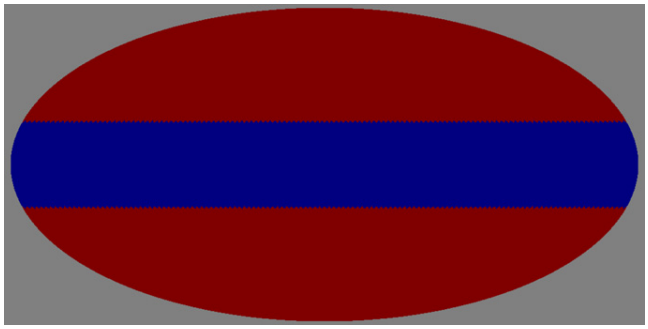


Figure 8. Mask used to test robustness in this section, covering a 20 deg strip centered on the galactic center.

We also tried applying the above method to larger masks, covering 30 deg and more, but the number of samples needed for convergence then quickly rose beyond 30,000. This shows that for masks that are significantly larger than those currently used in full-sky experiments, this likelihood estimator should not be trusted without prior testing—especially with regard to the number of Gibbs samples needed for a robust evaluation.

5. CONCLUSIONS

The main result presented in this paper is a statistically well motivated block factorization of the CMB power spectrum likelihood. Because the spherical harmonics are nearly orthogonal over the large sky coverages achieved by current CMB satellite experiments such as *Planck* and *WMAP*, any correlations between different C_ℓs are localized in multipole space. Under the assumption that these probabilistic dependencies have a strictly finite range, the full CMB likelihood may be reduced into a product of lower-dimensional marginals.

We have applied this result to two outstanding problems in CMB analysis. First, we use this expression to derive a well-motivated hybrid CMB likelihood estimator, merging an exact low-ℓ component with an approximate high-ℓ component, that accounts for correlations between the two regions. Although a detailed analysis of the *WMAP* likelihood shows that these

correlations are negligible for the *WMAP* sky cut and the six-parameter ΛCDM model, we nevertheless recommend this new estimator for future experiments and analyses, both because its implementation is trivial, and because it provides additional safety when analyzing non-standard models.

Second, we have shown how the same expression may be used to accelerate the convergence rate of the BR CMB likelihood estimator by orders of magnitude at high ℓs. This is achieved by factorizing the full parameter space into subspaces that each individually converge faster, and then merging these sub-blocks into a full-range estimator at the likelihood level using the block factorization formula.

It should be noted that these results rely directly on the assumption of vanishing long-range correlations. While this assumption holds to a very high accuracy for the basic CMB signal plus noise data model, it is in general not valid when including systematic effects in the analysis. Perhaps the two most important examples are correlated beam uncertainties and unresolved extra-Galactic point sources, each of which extend through all ℓ's (e.g., Planck XV 2013). Fortunately, these long-range degrees of freedom may be modeled in terms of a small number of power spectrum templates, each with an unknown amplitude. One can therefore marginalize over these by sampling the unknown amplitudes as nuisance parameters, similar to what was done for high-ℓ astrophysical parameters in the 2013 *Planck* likelihood (Planck XV 2013).

Finally, we note that the block factorization presented in Section 2 is a completely general statistical result that holds exactly for any banded probability distribution, and we therefore expect it to also find applications outside the CMB field.

This project was supported by the ERC Starting Grant StG2010-257080. Part of the research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with NASA. Some of the results in this paper have been derived using the HEALPix (Górski et al. 2005) software and analysis package.

REFERENCES

Bennett, C. L., Halpern, M., Hinshaw, G., et al. 2003a, ApJS, 148, 1
Bennett, C. L., Hill, R. S., Hinshaw, G., et al. 2003b, ApJS, 148, 97
Chu, M., Eriksen, H. K., Knox, L., et al. 2005, PhRvD, 71, 103002
Eriksen, H. K., Jewell, J. B., Dickinson, C., et al. 2008, ApJ, 676, 10
Eriksen, H. K., O'Dwyer, I. J., Jewell, J. B., et al. 2004, ApJS, 155, 227
Górski, K. M. 1994, ApJL, 430, L85
Górski, K. M., Hivon, E., Banday, A. J., et al. 2005, ApJ, 622, 759
Hinshaw, G., Larson, D., Komatsu, E., et al. 2013, ApJS, 208, 19
Hinshaw, G., Spergel, D. N., Verde, L., et al. 2003, ApJS, 148, 135
Hivon, E., Górski, K. M., Netterfield, C. B., et al. 2002, ApJ, 567, 2
Jewell, J., Levin, S., & Anderson, C. H. 2004, ApJ, 609, 1
Lewis, A., & Bridle, S. 2002, PhRvD, 66, 103511
Penzias, A. A., & Wilson, R. W. 1965, ApJ, 142, 419
Planck Collaboration I 2013, arXiv:1303.5062
Planck Collaboration XII 2013, arXiv:1303.5072
Planck Collaboration XV 2013, 1303.5075
Planck Collaboration XXIII 2013, arXiv:1303.5083
Rocha, G., Contaldi, C. R., Bond, J. R., & Górski, K. M. 2011, MNRAS, 414, 823
Rudjord, Ø., Groeneboom, N. E., Eriksen, H. K., et al. 2009, ApJ, 692, 1669
Smoot, G. F., Bennett, C. L., Kogut, A., et al. 1992, ApJL, 396, L1
Verde, L., Peiris, H. V., Spergel, D. N., et al. 2003, ApJS, 148, 195
Wandelt, B. D., Larson, D. L., & Lakshminarayanan, A. 2004, PhRvD, 70, 083511

Technical Report I

Efficient spherical harmonic transform codes for CPU and GPU

Chapter 3 describes novel techniques used to implement the Legendre transforms on the GPU. To my knowledge it is the world's fastest code for Legendre transforms, at 40% of the peak FLOP rate of the GPU. Achieving this was highly non-trivial. The Legendre transforms is a very interesting thing to try to make work well on the GPU; as demonstrated by the final implementation performing 45 times better than the initial naive implementation.

This code was written in order to pass a course on multi-architecture programming at the university. The idea was to develop the code further and turn it into a full paper, but that never happened, and so I simply attach the course work report unedited.

Note: This report has not been peer-reviewed and is only published as part of this thesis.

Draft version May 3, 2017
Preprint typeset using LaTeX style emulateapj v. 5/2/11

EFFICIENT SPHERICAL HARMONIC TRANSFORM CODES FOR CPU AND GPU

D. S. Seljebotn[1]

Draft version May 3, 2017

ABSTRACT

(Update May 2017: This abstract was added to an otherwise unedited report.) This report was submitted as part of doing the INF 9063 course at the University of Oslo in 2011. Chapter 2 covers the CPU implementation present in Wavemoth (Seljebotn 2011) in further detail. Chapter 3 describes a GPU implementation which to our knowledge was the fastest GPU implementation available of Legendre transforms at the time of writing. The first iteration performed at 4.57 GFLOP/s, which is incrementally increased through a series of optimizations described in this report, which git commits given for each step of the way. The most important is an algorithm for doing small matrix-vector products which is very specific to GPU hardware and minimizes communication between "warps", while still keeping all of the warps working on the same Legendre transform in order to make optimal use of cache. The final performance is at 209 GFLOP/s, or 40% of the theoretical maximum of the GPU. Conclusion: For the Legendre transform part of the SHTs, one GPU from 2011 could replace 32 CPUs from 2011. If one wishes to combined this with FFTs done on CPUs, one needs 8 CPUs to keep up with the GPU; so that, roughly speaking, 8 CPUs and 1 GPU could replace 32 CPUs.

Subject headings: Methods: numerical

1. INTRODUCTION

1.1. *The spherical harmonic transforms*

The spherical harmonic transform (SHT) is the spherical analog of the Fourier transform, and is an essential tool for data analysis and simulation on the sphere. A scalar field $f(\theta, \phi)$ on the unit sphere can be expressed as a weighted sum of the spherical harmonic basis functions $Y_{\ell m}(\theta, \phi)$,

$$f(\theta, \phi) = \sum_{\ell=0}^{\infty} \sum_{m=-\ell}^{\ell} a_{\ell m} Y_{\ell m}(\theta, \phi). \tag{1}$$

The coefficients $a_{\ell m}$ contain the spectral information of the field, with higher ℓ corresponding to higher frequencies. In calculations the the spherical harmonic expansion is truncated for $\ell > L$, and the spherical field represented by $O(L^2)$ grid samples (θ_i, ϕ_i). Computing the sum above is known as the *backward SHT* or *synthesis*, while the inverse computation,

$$a_{\ell m} = \sum_{i=1}^{N_{\text{pix}}} w_i f(\theta_i, \phi_i) Y^*_{\ell m}(\theta_i, \phi_i). \tag{2}$$

is known as the *forward SHT* or *analysis*. Here, the w_i are some predefined weights which depends on the spherical grid used.

In order to compute an SHT, the first step is nearly always a seperation of sums; similar to the seperation of a 2D Fourier transform to several 1D transforms. One makes use of the fact that

$$Y(\theta, \phi) = \widetilde{P}^m_\ell(\cos\theta) e^{\sqrt{-1} m \phi}, \tag{3}$$

where $\widetilde{P}^m_\ell$ is known as the *normalized associated Legendre function* of order m and degree ℓ. Note that the right term is the same term that appears in Fourier transforms, and that it does not depend on ℓ and θ, while the left term does not depend on ϕ. Assume that the pixels are organised in iso-latitude rings with n_i equidistant pixels within each ring i, so that pixel t within ring i has latitude θ_i and longitude

$$\phi_{t,i} = \phi_{0,i} + \frac{2\pi t}{n_i}. \tag{4}$$

We let $x_i = \cos\theta_i$. It turns out that one can compute equation (1) by first computing a *Legendre transform*,

$$q_m(x_i) = \sum_{\ell=|m|}^{L} a_{\ell m} \widetilde{P}^m_\ell(x_i), \tag{5}$$

then wrap around or zero-pad the coefficients on each ring,

$$\tau_v(x_i) = \sum_{u=-\infty}^{\infty} q_{n_i u + v}(x_i) e^{\sqrt{-1}\phi_{0,i}(n_i u + v)}. \tag{6}$$

and finally perform a Fourier transform,

$$f(\theta_i, \phi_{t,i}) = \sum_{v=0}^{n_i - 1} \tau_v(x_i) e^{2\pi v \sqrt{-1}/n_i}. \tag{7}$$

See figure 1 for an illustration. To compute equation (2), one can simply reverse the steps.

The complexity of all FFTs required is $O(L^2 \log L)$ in total, and very efficient open source computer codes are available (e.g., FFTW3). In contrast, not much work has been done on speeding up Legendre transforms. The brute-force computation scales as $O(L^3)$. Some more efficient algorithms are available, but not in wide use due to the constant prefactors involved. My primary project since May 2011 has been to implement an $O(L^2 \log^2 L)$ SHT algorithm (Seljebotn 2011). However, the speedup achieved from a better algorithm was only 2–3x on the

Electronic address: d.s.seljebotn@astro.uio.no
[1] Institute of Theoretical Astrophysics, University of Oslo, P.O. Box 1029 Blindern, N-0315 Oslo, Norway

resolutions of interest, which is in the same range as what one can gain by code optimization, while having significant disadvantages. The primary focus in this report will therefore be on optimizing brute-force Legendre transforms.

1.2. *About the code*

The code is available as part of the *Wavemoth* package. The CPU parts have been submitted as a paper (Seljebotn 2011), this version also contains the beginnings of a GPU implementation.

The codebase can be found at

`https://github.com/wavemoth/wavemoth/tree/cuda`

both for CPU and GPU; I refer to the commit `7393d75` unless another other commit hash is indicated[2].

When working on features specifically for INF 9063 (GPU support), I have continued to work within this code base. I will provide pointers to the relevant file within the source code base and ask the reader to simply ignore the rest of the code.

Both for the CPU and GPU implementations, I have relied heavily on string-based templating to generate the C or CUDA source code. The computional cores are in files with the extension `.c.in`/`.cu.in`, which is processed by the build system to pure C or CUDA code. The templating language of choice is Tempita, which basically means that any code between `{{` and `}}` is in a Python-like language. I am not sure how I could have completed the project at all if I had to write C directly.

Around the computational core in C or CUDA, I call the code from Python and use Python for conveniently writing and launching tests. For CUDA, this means that there are seperate kernels whose sole purpose is testing. The CPU C code can be used (but not tested) without of Python, while the CUDA implementation only has a Python frontend. Eventually, the CUDA implementation should also get a C frontend so that it can be called directly from C or Fortran without the Python dependency.

1.3. *Legendre transforms*

To make a long story short, one can cut computation in half by computing the even and odd parts of equation (5) seperately. We therefore change notation a bit, so that the Legendre transform, needed for spherical harmonic synthesis, becomes

$$q_{m,\omega}(x_i) = \sum_{k=0}^{K_{m,\omega}} \Lambda^{m,\omega}_{k,i} a'_{m,\omega,k} \tag{8}$$

where ω is 0 for the *even transforms* and 1 for the *odd transforms*. Essentially, $\ell = \omega + 2k$, and we use the notation $\Lambda^{m,\omega}_{k,i} = \widetilde{P}^m_{\omega+2k}(x_i)$ and $a'_{m,\omega,k} = a_{\omega+2k,m}$. Similarly, the *transpose Legendre transform*, needed for spherical harmonic analysis, becomes

$$a'_{m,\omega,k} = \sum_{i=1}^{n_i} \Lambda^{m,\omega}_{k,i} q_{m,\omega}(x_i). \tag{9}$$

[2] `https://github.com/wavemoth/wavemoth/tarball/7393d75`

Note that these equations are simply matrix-vector products, respectively,

$$\mathbf{q}_{m,\omega} = \mathbf{\Lambda}_{m,\omega} \mathbf{a}'_{m,\omega} \tag{10}$$

and

$$\mathbf{a}'_{m,\omega} = \mathbf{\Lambda}^T_{m,\omega} \mathbf{q}_{m,\omega}. \tag{11}$$

Note that while $\mathbf{a}'_{m,\omega}$ and $\mathbf{q}_{m,\omega}$ are complex vectors, the $\mathbf{\Lambda}$-matrices are real, so the matrix-vector products can be performed seperately on the real and imaginary parts. In the code, we simply let $n_{\text{vec}} = 2$ for a single transform, $n_{\text{vec}} = 4$ for two simultaneous transforms, and so on.

Examples of the $\mathbf{\Lambda}$-matrices can be seen in figure 2. For a full spherical harmonic transforms, one must perform $L + 1$ Legendre transforms for each of the even and odd cases, for a total of $2(L + 1)$ Legendre transforms. Note that the sizes of the transforms become gradually smaller as $K_{m,\omega} = \lfloor (L - m - \omega + 2)/2 \rfloor$, so that $\mathbf{\Lambda}_{0,0}$ has size $L/2 \times n_i$, while $\mathbf{\Lambda}_{L,1}$ has size $0 \times n_i$.

The elements in $\mathbf{\Lambda}$ can be computed by using the following recurrence relation in k,

$$\Lambda^{m,\omega}_{k+1,i} = (x_i^2 + \alpha^{m,\omega}_k)\beta^{m,\omega}_k \Lambda^{m,\omega}_{k,i} + \gamma^{m,\omega}_k \Lambda^{m,\omega}_{k-1,i}. \tag{12}$$

Using this recurrence relation corresponds to starting at the top of each column in the matrices in figure 2 and working downwards to higher k. The $K_{m,\omega}$ triplets of auxiliary values α, β and γ can be computed explicitly from m and $\ell = \omega + 2k$ using simple expressions containing division and square root (see Seljebotn (2011) for details). The n_i values of $x_i^2 = (\cos\theta_i)^2$ are determined by the spherical pixel grid in use, and although explicit expressions exists for some grids, they are treated as precomputed and read from memory in my code.

For only one or a few simultaneous transforms, transferring the $O(L^3)$ data volume of the $\mathbf{\Lambda}$-matrices over the memory bus (or even store them in memory) is not viable. Instead, one must use equation (12) at the same time as doing the Legendre transforms of equations (8) and (9). The FLOP-count is then $5T$ for the recurrence relation plus $2Tn_{\text{vec}}$ for the matric-vector products, where T is the number of elements in the $\mathbf{\Lambda}$-matrices in total.

There is a slight numerical problem: The upper-right corners of the $\mathbf{\Lambda}$-matrices can become extremely close to zero, much lower than the $\sim 10^{-300}$ supported by IEEE double precision floating point. This is solved by storing the exponent in a seperate variable while doing the arithmetic in this region, which involves a lot of checking and branching. My approach has been to, for each column, store the first two values larger in absolute value than $\sim 10^{-30}$ as precomputed data, and ignore the near-zero region during transforms, so that the computation time in the near-zero region does not matter. Still, this makes writing the code a bit harder as the region of computation is not rectangular; instead one must "hug" the arc on the right side of the non-zero regions. I only include the non-zero elements of the $\mathbf{\Lambda}$-matrices in FLOP count T.

1.4. *Parallelism in the SHT*

As illustrated in figure 1, the SHT is a two-step process. For the Legendre transforms, each $\mathbf{\Lambda}_{m,\omega}$ can be processed independently, with no sharing in input, output,

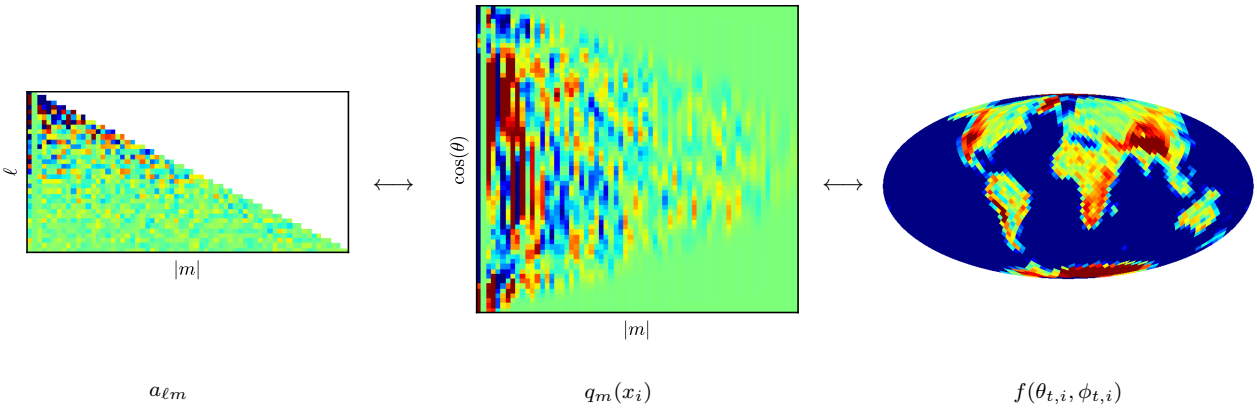


Figure 1. Separating the sums in an SHT. A Legendre transform is used on each column m to go between $a_{\ell m}$ (left) and the ring-phases $q_m(x_i)$ (center), while a Fourier transform is used on each ring to go between $q_m(x_i)$ and the spherical field $f(\theta_{t,i}, \phi_{t,i})$ (right), here plotted in the Mollweide projection.

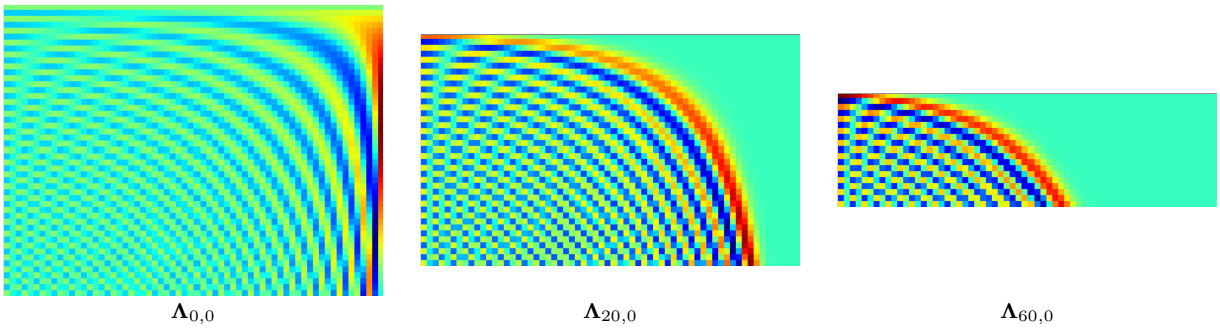


Figure 2. Examples of $\mathbf{\Lambda}$-matrices, here for $L = 64$. Througout this report and the code, k corresponds to the row index and i to the column index of these matrices.

or auxiliary data. Similarly, the Fourier transforms can be carried out independently on each iso-latitude ring. However, the data transposition involved means that all Legendre transforms must be finished before starting the Fourier transforms (or vice versa, depending on whether one is doing synthesis or analysis). At this point, the FLOP count scale as $O(L^3)$, while the number of independent tasks scale as $O(L)$. For current resolutions, this is plenty of parallelism for the CPU, but not quite enough for the GPU.

Going deeper, there is room for parallelism within each Legendre transform or Fourier transform as well. We will not discuss the Fourier transforms, as we simply use existing libraries for those. For the Legendre transforms, there is another $O(L)$ factor of parallelism by processing each column of $\mathbf{\Lambda}$ in seperate threads. This can either be done independently (non-transposed transform) or with some communication (transposed transform). This brings the number of threads of execution up to $O(L^2)$. The recurrence relation of equation (12) naturally prevents splitting up $\mathbf{\Lambda}$ too much in the vertical direction, although vertical blocking would of course be possible if one uses some additional memory for storing starting values for the reccurence relations (there are also other ways of computing the elements of $\mathbf{\Lambda}$, although they tend to be either numerically instable or much more expensive). At any rate, for data resolutions that can be imagined today, the $O(L^2)$ threads of execution in total given by horizontal division of $\mathbf{\Lambda}$ alone is more than sufficient.

Finally, what about multiple simultaneous transforms? For the purposes in astrophysics (such as analysis of the Cosmic Microwave Background), one often wishes to use the SHT to solve a linear system using an iterative solver. This is an inherently sequential process, where one input depend on the previous output, which precludes data parallelism on a large scale.

There are however multiple frequency bands, and also one may sometimes wish to solve a limited number of linear systems simultaneously, so that a smaller number (between 5 and 20) simultaneous transforms is feasible. Because the Legendre transfrom FLOP count is $5T + 2Tn_{\text{vec}}$, one does *not* wish to treat multiple inputs independently, but keep them together in order to amortize the $5T$ FLOPs used for the recurrence relation.

2. CPU IMPLEMENTATION

Only spherical harmonic synthesis was implemented for the CPU, not analysis. Double precision floating point is used throughout. A single transform is understood to mean $n_{\text{vec}} = 2$ within the Legendre transform, that is, both the real and imaginary parts are present.

Important source code files:

- `src/legendre_transform.c.in` – The computational core
- `src/wavemoth.c` – Combines the Legendre transforms with Fourier transforms (from FFTW3) to do a full SHT, including NUMA-aware parallelization/scheduling over many CPU cores

- `bench/shbench.c` – Benchmark application

The test system contains 64 hyperthreaded 2.27 GHz Intel Xeon X7560 cores and 500 GB memory, supporting a theoretical 9.2 GFLOP/s per core. For all benchmarks below I

- use $L = 4096$, which is the highest resolution in use in CMB analysis today.
- measure the performance of all Legendre transforms necesarry for a full SHT ($m = \{0, \ldots, L\}, \omega \in \{0, 1\}$)
- repeat each benchmark multiple times and take the minimum wall clock time (which is quite stable). Note that the working set is hundreds of megabytes and does not fit in cache.
- launch 64 threads on 32 of the cores (to utilize HyperThreading), locked to four physical packages so that the other half of the system is free for other use (more results are present in Seljebotn (2011)).

2.1. *Precomputation and edge cases*

As mentioned above the $\mathbf{\Lambda}$-matrices are not representable everywhere. In the CPU case, I solved this by blocking the matrices horizontally in a precomputation step, so that each block is rectangular and can be represented in IEEE double precision (meaning I restrict the top row to contain values between 10^{-200} and 10^{-30} in absolute value).

All auxiliary data ($\alpha_k^{m,\omega}$, $\beta_k^{m,\omega}$ and $\gamma_k^{m,\omega}$, x_i^2, and starting values for the recurrence relation) is saved to disk during the precomputation phase.

All the techniques below relies on blocking, but the input sizes does not divide the block sizes. This is where the use of templates becomes indispensable, which allows code such as

```
{{for xchunksize in [1, 2, 6]}}
static void transform_chunk{{xchunksize}}(...) {
   ...
}
{{endfor}}
```

Then, a wrapper routine first calls `transform_chunk6` for as long as possible, then `transform_chunk2` (if 6 does not divide the input size) and finally `transform_chunk1` (if the input size is odd).

2.2. *Single Legendre transform*

The first important point is to make sure the number of loads from cache into CPU registers is balanced with the number of floating-point operations. The second is to make sure there are enough independent floating-point operations in flight simultaneously, so that operations can be pipelined. Thus,

- the values of $\Lambda_{k,i}^{m,\omega}$ should never need to leave the CPU registers. Rather, we fuse equations (8) and equation (12) in the core loop. Also, the accumulators for the output $q_{m,\omega}(x_i)$ should not leave registers, to avoid high-latency load–increment–store cycles.
- we process for several x_i simultaneously. This a) amortizes the register loads of the auxiliary data $\alpha_k^{m,\omega}$, $\beta_k^{m,\omega}$ and $\gamma_k^{m,\omega}$, and b) ensures that there are multiple independent chains of computation going on, so that we overcome the pipeline latency of the floating point operations.

In the first iteration I allocated the 32 available double precision slots in the 16 available SSE registers as follows:

- The auxiliary data $\alpha_k^{m,\omega}$, $\beta_k^{m,\omega}$, $\gamma_k^{m,\omega}$ must be loaded for each row. The values must be duplicated across the SSE registers, so that they use a full register each.
- When processing four i (columns of $\mathbf{\Lambda}$) in "parallel" during the recurrence relation (two columns in each SSE instructions, and switching between working on each of the two columns for better pipelining), one needs two registers for each of $\Lambda_{k,i}^{m,\omega}$, $\Lambda_{k-1,i}^{m,\omega}$, and x_i^2, for a total of six registers.
- Four ($2n_{\text{vec}}$) accumulation registers are needed for $q_{m,\omega}(x_i)$.
- The input $a'_{m,\omega,i}$ can be read after the auxiliary data is discarded and does not conserve extra register space.

So in total 13 registers are booked, leaving three for computation. This allocation results in a performance of 5.69 GFLOP/s per core (62% of the theoretical maximum).

Increasing the number of i processed in each pass to six books all 16 registers, leaving no work registers and causing one register to spill to stack. This still increases performance to 6.42 GLOP/s per core (70%).

Since x_i^2 above is constant, and only needs to be loaded and discarded. By loading x_i^2 again from L1 cache each time it is needed, the register spilling is avoided, resulting in a minor improvement to 6.46 GFLOP/s per core.

2.3. *Multiple simultaneous Legendre transforms*

For multiple transforms (n_{vec} in the range $4 \ldots 20$), I wish to amortize the use of the recurrence relation, since $\Lambda_{k,i}^{m,\omega}$ does not depend on the input. This leads to the following algorithm:

1. Repack the input 2D array $\mathbf{a}'$ to blocks of size $B_j \times B_k$. The block size should be tuned, but the choice of $B_j = 4$ and $B_i = 6$ achieved good performance.
2. Repeat until done:
   (a) Use recurrence relation (12) to compute a $B_k \times B_i$ block of $\mathbf{\Lambda}$. We simply choose B_k to be such that the data size is 4 KB, which is large enough to amortize any branch overhead (and avoids a TLB miss, although whether that matters is unknown).
   (b) Fully compute a $B_j \times B_i$ block of the output $\mathbf{q}$, by doing a matrix multiplication of the entire $\mathbf{a}'$ with the computed block of $\mathbf{\Lambda}$. Because of the repacking in step 1, $\mathbf{a}'$ can be contiguously accessed.

Goto & Geijn (2008) was an invaluable resource in learning about blocking techniques to properly amortize loads.

The resulting routine performed at 5.6 GFLOP/s per core (60%) for ten simultaneous transforms ($n_{\text{vec}} = 20$).

2.4. *Results*

Combining the Legendre transforms with FFTs performed by FFTW3, the results are as follows: Using 32 cores on four physical CPUs on the hardware mentioned above, the Wavemoth code is able to do brute-force spherical harmonic synthesis at resolution $L = 4096$ in 1.17 seconds for a single transform, or 0.7 seconds per transform for 10 concurrent transforms. This is respectively 2.2x and 1.6x faster than the current state-of-the-art implementation, libpsht (Reinecke 2011). More detailed results and plots are available in Seljebotn (2011).

3. GPU IMPLEMENTATION

For the GPU I choose to implement the transpose Legendre transform, needed for spherical harmonic analysis, which is more difficult and interesting than the non-transpose Legendre transform of the previous section. That is, I want to compute

$$a'_{m,\omega,k} = \sum_{i=1}^{n_i} \Lambda_{k,i}^{m,\omega} q_{m,\omega}(x_i). \tag{13}$$

The fundamental problem to be solved is that the direction of the sum is along i, the columns of $\mathbf{\Lambda}$, while the sequential recurrence relations used to obtain $\Lambda_{k,i}^{m,\omega}$ works along k, the rows of $\mathbf{\Lambda}$.

Important files:

- `wavemoth/cuda/legendre_transform.cu.in` – The (heavily templated) CUDA coda.
- `wavemoth/cuda/*.py` – Python code that is responsible for instantiating the CUDA code, compiling it, and calling it. This is mostly boilerplate thanks to PyCUDA, which accepts string input, compiles it, and dynamically loads it.
- `wavemoth/cuda/test/test_cuda.py` – Test case for the sum-reduction framework (see below)
- `examples/gpu.py` – Benchmarks the code and validates the result for a single (m, ω), repeating the same computation on an arbitrary number of thread blocks. This is where I put loops over different compile-time parameters and benchmark them against one another.
- `examples/gpusht.py` – Benchmarks the code and validates the result for all the (m, ω) needed for a full SHT, using one thread block per (m, ω) (for a total of 8192 thread blocks for $L = 4096$). Also implements pipelining in order to hide data transfer times for multiple subsequent transforms.

I follow what appears to be common practice in CUDA development: Hard-code as much as possible compile-time. For instance, while the number of threads in a thread-block is in principle a dynamic quantity in CUDA, I also pass it in compile-time, so that it can be taken into account both during template code generation and by the compiler.

I feel it is hard to oversell the convenience of using Python for CUDA development. For instance, I wrote a Python context manager which parses the CUDA profile logs, so that the following code is sufficient to profile a kernel and display some statistics (including occupancy etc.):

```
with cuda_profile() as prof:
    for rep in range(3):
        plan.execute_transpose_legendre(q, a)

print prof.format('all_transpose_legendre_transforms',
                  nflops=plan.get_flops(),
                  nwarps=2)
```

3.1. *Hardware*

As the purpose of the project is to have a working code for scientific purposes, I have developed the code on an NVIDIA Tesla M2050 card (the node was rented by the hour from Amazon EC2). Important numbers:

- Theoretical compute performance of 515 GFLOP/s for double precision
- Maximum number of 32-bit registers per thread is 64, which then allows for 16 warps (the register file is 128 KB in total). Using less registers one can fit a maximum of 48 warps on each streaming multiprocessor (SM).
- 48 KB shared memory and 16 KB L1 cache on-chip (can also be configured as 16+48, respectively, but all the benchmarks below use 48KB shared memory)
- 2.6 GB memory with ECC turned on (performance-wise, ECC did not matter much)

3.2. *Transpose Legendre transform*

As hinted above, it seems obvious to use one thread-block per (m, ω), which creates 8192 blocks for our highest resolution of interest ($L = 4096$). Each block then shares the auxiliary data $\alpha_k^{m,\omega}$, $\beta_k^{m,\omega}$, $\gamma_k^{m,\omega}$, and must handle sum-reduction along i while employing the recurrence along k.

Benchmarks are for $n_{\text{vec}} = 2$. The auxiliary values for the recurrence relations are computed directly as they are needed. The FLOPs necesarry to compute these are *not* included in the statistics below (but are negligible in all but the first commit).

The following is stored as precomputed data: The x_i^2, the starting values for $\Lambda_{k,i}^{m,\omega}$ (the first two values larger than 10^{-30} in each column), as well as an array $i_{\text{stop},k}^{m,\omega}$ of 16-bit integers which gives the length of each row in the $\mathbf{\Lambda}$-matrices, in order to describe which part is to be treated as zero and where to start the recurrence relations. The total size of the precomputed is around 550 MB for $L = 4096$.

The full input to the necesarry set of transpose Legendre transforms weighs in at 512 MB, while the output is 128 MB, per transform.

3.3. *First attempts*

Commit `9936fe` (4.57 GFLOP/s): A simple baseline code. The auxiliary values are computed in each thread, and the row-wise sum-reduction is done only in the thread with local ID 0. Not only is the performance poor, but the approach limits the number of columns that can be handled to low resolutions (because one thread is used per column, and each row written to shared memory for reduction).

Commit `ec20cb1` (9.91 GFLOP/s): The auxiliaries are first precomputed in parallel and stored to shared memory for each k, and then reused in each thread. The first blocking appears here, as the code iterates between computing auxiliaries for B_k rows, and performing computations for B_k rows.

Commit `30ff749` (27.18 GFLOP/s): A simple step further is to make the sum-reduction multi-threaded. First, each row is processed in full, each thread processing $B_i = \lceil n_i / n_{\text{thread}} \rceil$ columns. The input is loaded again from global memory for each row. Then, the n_{thread} contributions to each output $a'_{m,\omega,k}$ is reduced to n_{warp} contributions by doing a $(\log_2 32)$-step tree-reduction within each warp. These contributions are saved to shared memory. At the end of B_k blocks, the values stored in shared memory are then summed across warp index.

3.4. *Proper blocking and an algorithm for parallel tree-reduction*

It is now clear that loading the memory from global memory for each row will not work well; one must amortize the load of the input $\mathbf{q}$ from global memory over many rows. The solution to this problem is blocking, where blocks of size $B_k \times B_i n_{\text{thread}}$ are fully processed before moving on. The B_k parameter affects how much shared memory is used for auxiliary data, while B_i affects register pressure in each thread. Good choices for these parameters were $B_k = 64$ (even $B_k = 48$ was noticeably worse) and $B_i = 4$.

While working on blocking, it also became clear that the $(\log_2 32)$-step tree-reduction within each warp is a serious bottle-neck, as it leaves some threads idle. An alternative algorithm to tree-reduction is to first have all 32 threads in a warp compute 32 rows of values to add, then have the same threads sum across each row. The problem with this approach is that even for the minimal block-size, $32 \cdot 32 \cdot 8 = 8192$, which would only allow five warps in total per SM $\Rightarrow$ not viable.

These considerations leads to the following hybrid algorithm for the sum-reduction. I have only implemented it for $n_{\text{vec}} = 2$, so that the current GPU code is somewhat less generic than the CPU code. (Apologies up front for the poor exposition, I hope to come back to this in a more visual form in the presentation. A simple prototype Python implementation can be found in `examples/parallel_tree_reduction.py`).

- First, like before, each thread sums up its B_i values per row while computing...
- ...which leaves us with 64 values per warp, $n_{\text{vec}} = 2$ per thread, which we want to sum to only 2 values. We start with the "leaf" portion of normal tree-reduction, with each thread sending 32 values to its neighbour thread for accumulation, resulting in 32 values. We see these as an array of shape (2, 1, 16): First axis corresponds to output vector, second axis to the row (k), and the third are 16 values which remains to be summed.
- Store the (2, 1, 16) array to shared memory, and proceed with the next row to produce another (2, 1, 16) array.
- Now we have 32 + 32 values, which can be added together with full thread utilization to produce a (2, 2, 8) result, spanning two rows (k).
- The scheme continues in this fashion. For each row, a new (2, 1, 16) block is produced, which is combined with any existing blocks of the same size and reduced as far as possible without loosing parallelism. In the end one is left with a single (2, 16, 1) block – the fully reduced result (for the warp) for 16 rows.
- The final (2, 16, 1) buffers are finally added together across warps. It turned out that two warps per thread block is optimal (less would use too much memory and hurt occupancy, more would mean more synchronization), so this reduction is at this point simply done in the first warp. Note that it is not until this step that `__syncthreads()` must be called.

In theory, the algorithm requires 6 buffers of 32 double precision numbers, or 1536 bytes, per warp. In practice, it was simpler to implement when using 4 buffers of 64 numbers, or 2048 bytes.

Implementing this scheme was done simply by writing the algorithm in the natural top-down recursive way *in the template generation stage*. The resulting generated code is similar to a loop having been manually unrolled 16 times, but has slightly different code for each row in order to facilitate the reduction. There is a seperate kernel whose purpose is to unit-test this template generation (simply passing in a simpler block of code to be executed for each row).

The result of implementing all of this can be found in **commit `ea462c24`**, which yields **191 GFLOP/s** over the Legendre transforms needed for an SHT ($L = 4096$). A compiler oddity: Before adding `__forceinline__` to the computational core routine, the performance was only 115 GFLOP/s, even if the routine was only called from a simple wrapper kernel. This is similar to the drop in berformance that is experienced when adding a reference to `printf` in a kernel.

The occupancy at this point is 14 warps in 7 blocks, which is limited by the amount of shared memory used – 16 warps in 8 blocks would use 53 KB at this point. Since the recurrence auxiliaries are shared within a block, increasing the number of warps per block to four increases occupancy to 16 warps in 4 blocks. However, performance decreases to 150 GFLOP/s, even if the inter-warp sum-reduction is commented out – it is clearly very important to use many independent blocks. Fixing the buffer allocation strategy in the sum-reduction scheme would allow 16 warps in 8 blocks, which would then use 45 KB of shared memory.

Another failed experiment was trying to get rid of bank conflicts in the parallel reduction code. In theory, there

should be a conflict, but getting rid of this conflict (in theory, and unless I did something wrong) degraded performance to 178 GFLOP/s.

Commit `7393d75` (209 GFLOP/s): It turned out (despite my initial intuition) that it was rather simple to decrease the scratch space for the parallel tree reduction to 1526 bytes per warp, which allowed 16 warps at once (using 45 KB).

I did not have time to do any micro-optimization and have just focused on getting the basic algorithm right; there may well be some potential for tweaking at the instruction level.

3.5. *Dealing with the near-zero regions*

The process of avoiding the near-zero regions seen in figure 2 is a lot easier to express in CUDA than using SSE intrinsics. The approach taken is:

- Mark unstarted columns using a special NaN value
- For each warp-block ($16 \times 32B_i$), check whether any columns in the warp are unstarted, and if so, take a slower code path which for each row checks whether the recurrence relations for each columns should be started, continued, or ignored.

As the blocks along the edge are few compared to all the inner blocks, it does not matter if these are relatively slow (on the order of 40–80 GFLOP/s).

3.6. *Fourier transforms on GPU?*

The next piece for a full SHT is the Fourier transforms, which can be executed independently for each ring. For the pixelization in use in CMB analysis (HEALPix), and resolution $L = 4096$, there are 8191 independent 1D FFTs to be performed, starting with 4 pixels in the ring at the north pole, then 8 pixels, then 12, and so on until one reaches 8192 pixels in a band around equator. In summary, a) many of the FFTs involve large primes, and as they are circular, require the use of the Bluestein algorithm, b) most of the FFTs are of different length.

The NVIDIA cuFFT library does support prime sizes using the Bluestein algorithm, *however*, it does not support plans with different n for each FFT. I experimented with cuFFT (see `src/wavemoth_cuda.c`), but was unable to reach acceptable performance for even 2000 kernel launches. Indeed, profiling reveals that each FFT seems to use a single block, and, according to documentation, the hardware only supports four concurrent kernel launches, so it seems impossible to utilizate the GPU with different-sized FFTs.

Another option is *pyfft*, which, despite the name, is a CUDA FFT library (forked from code originally developed by Apple). As it is open source it should be possible to construct a single kernel for doing all the FFTs. However, as it does not support the Bluestein algorithm but only 2^k transform sizes, this needs some work, and I leave it for the time being.

3.7. *Pipelining & double-buffering*

I implemented basic pipelining in `examples/gpusht.py` in order to hide data transfer; see the log in the appendix for the relevant numbers. As the number of transforms increases, the host-to-host compute rate gets close to the on-device compute rate, showing that hiding the data transfer is effective. At 20 transforms, the difference in time between host-to-host transform and on-device transform is 0.008 seconds per transform, or 0.8%. In the synchronous case, the time for the data transfer is 0.13 seconds, or 13% of the compute time. Thus the pipelining is clearly very efficient in hiding data transfer time.

3.8. *Estimates for a full SHT*

The logical conclusion of section 3.6 is that, at least in the first iteration, the FFTs should be performed on the CPU using, e.g., FFTW3. Unfortunately I did not have time to implement this yet, but I hope to do so in the future.

On the 2.27 GHz Intel Xeon system used in section 2, preparing the input for and performing the FFTs according to equations (6) and (7) takes 360 ms using 8 cores, or 189 ms using 16 cores. So it seems clear that 8 CPU cores working together with the GPU card will be able to do a full SHT in 1.15 seconds (assuming pipelining multiple transforms).

The conclusion is that 8 CPU cores and a Tesla M2050 can perform as well as 32 CPU cores.

APPENDIX

LOG FROM GPUSHT.PY

```
== Multi-buffered run with 20 transforms at nside=2048
Wall-time taken to set up instruction streams ("Python overhead"): 9.173083e-02
Wall-time taken to end of execution: 21.057755 total, 1.052888 per transform
Host-to-host compute rate: 207.159003 GFLOP/s

== Profiled run at nside=2048
Transfer in:   8.901e-02 +/- 2e-05 sec = 6.03 GFLOP/s, occupancy 0.00 (0 warps)
Compute:       1.045e+00 +/- 9e-05 sec = 208.65 GFLOP/s, occupancy 0.33 (16 warps in 8 blocks)
Transfer out:  4.111e-02 +/- 5e-07 sec = 6.53 GFLOP/s, occupancy 0.00 (0 warps)

== Accuracy table (m, odd, relative error)
0 0 3.73785964985e-11
0 1 4.11059904524e-11
1 0 1.31435966764e-11
1 1 1.05841910886e-11
2048 0 2.22537218451e-13
2048 1 9.93556480046e-14
```

```
4096 0 1.56068615471e-15
4096 1 0
```

Note: The inaccuracies are due to a different numerical scheme being used, and is a fundamental numerical issue with the recurrence relations (also, I have not checked yet which is more correct!). Using the less accuracte `rsqrt` function during auxiliary computation does lower the accuracy slightly, but the effect on performance is very slight. At any rate, they show that the code is "bug-free", as the presence of bugs will tend to give much higher errors.

REFERENCES

Goto, K., & Geijn, R. 2008 ACM Trans. Math. Softw., 34, 3
Reinecke, M. 2011 Astronomy & Astrophysics, 526, A108 `http://arxiv.org/abs/1010.2084`
Seljebotn, D. S. 2011 The Astrophysical Journal Supplemental Series (submitted) `http://arxiv.org/abs/1110.4874`
Tygert, M. 2010 Journal of Computational Physics, 229, 18 `http://arxiv.org/abs/0910.5435`