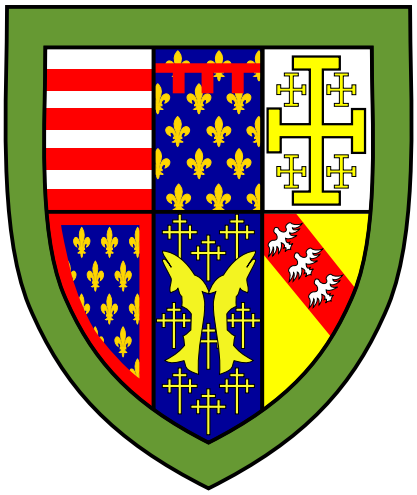
Advanced Bayesian Data Analysis for Astrophysics, Cosmology and Beyond

Adam Ormondroyd

Supervisors: Prof. M.P. Hobson,
Dr. W.J. Handley, Prof. A.N. Lasenby

Cavendish Astrophysics

November 2022



Abstract

In cosmology, there are many cases where it is useful to reconstruct one–dimensional functions, such as the primordial matter power spectrum or the evolution of the dark energy equation of state parameter with redshift. The most general approach would be to fit a free–form function, where the nature of the function is itself determined by the data, not just its parameters. One such approach to this is to use a *linf*: a linear spline between a variable number of nodes. This problem requires exploring a highly degenerate and multi–modal posterior distribution: nested sampling is a natural tool to investigate this.

For the first year of my PhD, I have focussed on learning how to use existing code: the cosmological Boltzmann code `CAMB`, the sampling and modelling framework `Cobaya`, and the nested sampler `PolyChord`.

I begin by exploring toy examples with `PolyChord`, then move to reconstructing the primordial matter power spectrum. This exposes an issue with `PolyChord` where it fails to identify clusters of sampling live points at higher resolution and therefore misses a posterior mode present at lower resolution. Dr. Handley and I add an interface to `PolyChord` to enable the user to provide a custom clustering algorithm, and plan to investigate alternatives to `PolyChord`'s native K–nearest–neighbours method.

The distribution version of `Cobaya` only allows the user to provide their own $\mathcal{P}_\mathcal{R}(k)$ function for the primodial matter power spectrum. To investigate the dark matter equation of state, I add an interface to `Cobaya` to allow $w(a)$ to be provided in a similar way. This interface is almost ready, except for the need for `Cobaya` to automatically understand that the equation of state parameters can be sampled together with the other `CAMB` parameters.

Contents

1 Introduction

This report summarises the first year of my PhD. I have focussed primarily on familiarising myself with existing code, in particular the cosmology code `CAMB` (Code for Anisotropies in the Microwave Background) [**CAMB˙1**, **CAMB˙2**], the sampling and modelling framework `Cobaya` (**Co**de for **bay**esian **a**nalysis) [**Cobaya**], and the nested sampler `PolyChord` [**PolyChord˙1**, **PolyChord˙2**]. The theme has been to use a *linf*, a linear spline between either a fixed or variable number of nodes, to reconstruct a function from data.

I began by exploring toy examples with `PolyChord`, then proceeded to apply this to cosmology. I reconstructed the primordial matter power spectrum, using the interface which `Cobaya` provides. This revealed an issue with `PolyChord`, where the native K–nearest–neighbours clustering fails to identify separate clusters of parameters at higher resolutions. I have worked with Dr. Handley to add an interface to `PolyChord` where the user can supply their own Python clustering, so that we will be able to experiment with different algorithms.

The primordial matter power spectrum was practice for reconstructing the dark energy equation of state parameter w in the same way, which is more difficult as `Cobaya` does not provide an interface for this. I have added this myself, which is working and almost finished, but I cannot make any conclusions about dark energy until the issues with `PolyChord` have been resolved.

2 Cosmological Context

2.1 Cosmological tensions

The expansion of the universe is characterised by the Hubble parameter $H(t) = \dot{a}/a$, where a is the cosmological scale factor, and its value today, the famous Hubble constant $H_0 \coloneqq H(\text{today})$. The "Hubble tension" is the descrepency between measured values of H_0; the Planck collaboration measures a value of $H_0 = 67.4\ \mathrm{km\,s^{-1}\,Mpc^{-1}}$ from the early universe, while the SH0ES team finds $H_0 = 73.2\ \mathrm{km\,s^{-1}\,Mpc^{-1}}$ using distance ladders. If this "Hubble tension" is not due to systematic error, then the ΛCDM cosmological model may be somehow incomplete [**buyers-guide**].

Another cosmological tension is the "S_8" tension. Weak galaxy lensing surveys consistently favour a lower amplitude for matter fluctuation spectrum than expected by ΛCDM at low redshifts, $z \gtrsim 0.5-1$. The matter fluctuation spectrum is quantified by the parameter $S_8 \coloneqq \sigma_8\sqrt{\Omega_\mathrm{m}/0.3}$. σ_8 is the root–mean–square matter fluctuation over a sphere of radius $8h^{-1}\ \mathrm{Mpc}^{-1}$, where h is the Hubble constant's dimensionless cousin ($H_0 = 100h\ \mathrm{km\,s^{-1}\,Mpc^{-1}}$) [**s8-drag**]. These measurements can also be reconciled with Planck ΛCDM cosmology, for example by modifying the matter power spectrum on non-linear scales [**Amon-2022**].

2.2 Dark energy parameterisations

A useful general approach in cosmology is reconstructing one–dimensional functions, prime examples are the primordial matter power spectrum $\mathcal{P}_\mathcal{R}(k)$, and the equation of state parameters of various ideal fluids $w(a)$, such as dark matter or dark energy. Historically, this has been achieved by assuming a particular functional form, in the case of the power spectrum it is usually written as a power law $\log \mathcal{P}_\mathcal{R} \propto \log k$. In lieu of a theoretical understanding of dark energy, a number of dark energy equation of state parameterisations exist, for example [**Vazquez˙2012**]:

$$w(a) = \begin{cases} = w_0 & \text{(cosmological constant)} \\ = w_0 + (1-a)w_a & \text{(CPL)} \\ = w_0 + (1-a)aw_a & \text{(JBP)} \\ = w_\mathrm{p} + (w_0 - w_\mathrm{p})\frac{a^{1/\tau}[1-(a/a_\mathrm{t})^{1/\tau}]}{1-a_\mathrm{t}^{-1/\tau}} & \text{(FNT),} \end{cases} \tag{1}$$

to name a few. $w = -1$ is the famous cosmological constant. The Chevallier–Polarski–Linder parameterisation (CPL) is the first–order Taylor expansion around $a = 1$, so $w_a = -\frac{dw}{da}|_{a=1}$[**CPL˙1**, **CPL˙2**]. This approximately captures the low–redshift behaviour of light, with slow–roll scalar fields with minimal coupling, without the complexity of various potential shapes and initial conditions [**planck13xvi**]. This parameterisation was used in the Planck 2015 and 2018 analyses; 2015 also considers a generic expansion in the scale factor:

$$w(a) = w_0 + \sum_{i=1}^{N} (1-a)^i w_i, \tag{2}$$

which is considered up to $N = 3$. They find the w_i are weakly constrained, and that other parameters are stable with respect to N, so conclude that a linear parameterisation is sufficient [**planck15xiv**].

The Jassal–Bagla–Padmanabhan (JBP) allows dark energy to have the same equation of state at the present epoch and at high redshifts (small a), with rapid variation at low redshift [**JBP**]. Unlike CPL and JBP, the Felive–Nesseris–Tsujikawa (FNT) patameterisation depends on four parameters instead of two: a_t is the scale factor at the transition epoch, τ characterises the width of the transition, w_p is the equation of state parameter in the asymptotic past. It allows for fast transitions in w with an extremum at $a_\mathrm{t}/2^\tau$ [**FNT**].

Assuming a particular functional form has limitations, ideally the data should be able to demand structure where it needs it, and simplicity where it doesn't — a "free–form" reconstruction. One such reconstruction is a linear spline between nodes in (a, w) space, where the number of nodes is itself a parameter to be determined. This is not totally free–form, but freer than most, as the data is free to choose the level of complexity. This type of spline has been used elsewhere to reconstruct the reionisation history $x_\mathrm{e}(z)$ [**Millea˙2018**, **Heimersheim˙2022**].

The linear spline is an appropriate choice as an appeal to simplicity. If we perform a polynomial spline between all nodes, then the requirement of continuous first and second derivatives can lead to nodes having tight error bars while lying nowhere near the original function. For example, suppose a three-node reconstruction of $f(x)$ is well constrained at low and medium x, but less so at large x. As the positions of the left and intermediate node will be well constrained, they will influence the position of the right node, attributing it a lower uncertainty than appropriate. This is not desired, the choice of polynomial has directly influenced the position and uncertainty of a node, which is at odds with a free–form approach. Therefore, a linear spline is a better description [**Vazquez˙2012˙pk**]. Dark energy exists as a postulate to explain the late–time acceleration of the universe's expansion, therefore the most minimal assumptions about its dynamical behaviour must be made and tight derivative constraints would be directly at odds with this.

An alternative and currently popular approach might be to use a Gaussian Process regression. This approach makes the assumption that both the data and the function being reconstructed are Gaussian, and their distribution can be fully described by a mean function and a covariance kernel. The choice of mean function is important, and is neglected in a lot of the literature, which typically uses zero as the mean function [**GP**]. This choice again contradicts our desire for simplicity, and faces a similar condition to a polynomial spline in that the derivative must also conform to a Gaussian distribution.

Fitting a linear spline with a variable number of nodes is a self–consistent approach which uses Bayesian methods to constrain both the number of parameters and their values. The posterior is therefore multi–modal and degenerate, as extra nodes can always be inserted without changing the function, and more than one mode may exist for a given number of nodes, such as those seen in section **??**. Nested sampling is a natural tool to investigate such a problem, and I will focus on the `PolyChord` implementation.

2.3 Related work in the group

If dark matter and dark energy indeed exist, it would seem natural that a "dark radiation" would also exist to mediate the interactions between such particles [**Ackerman˙2009**]. A transision in the behaviour of this radiation has the potential to answer the Hubble tension [**Aloni˙2022**]. Theoretical work by Dr. Barker, another member of the group, provides a gravitational mechanism for this [**Barker˙2020**]. This work explores a one-parameter extension to ΛCDM, an effective dark radiation which tracks the dominant equation of state parameter of the current epoch:

$$w_\mathrm{eff} = \begin{cases} w_\mathrm{r,eff} = 1/3 \\ w_\mathrm{m,eff} = (1 - 1/\sqrt{3})/2 \approx 0.211 \\ w_\Lambda,\mathrm{eff} = 1/\sqrt{3} \approx 0.577. \end{cases} \tag{3}$$

Within gravity, the Lagrangian may contain invariants of the field strengths up to second order. The Einstein field equations arise from varying from the Einstein-Hilbert action:

$$L_{\mathrm{T}} = \frac{1}{2\kappa} R - \frac{\Lambda}{\kappa} + L_{\mathrm{M}}, \tag{4}$$

where the gravitational portion $L_{\mathrm{G}} = L_{\mathrm{T}} - L_{\mathrm{M}}$ is powered by the Riemann curvature, $R \equiv R^{\mu\nu}{}_{\mu\nu}$. The matter lagrangian L_{M} provides the standard model of particle physics when below the electroweak transition temperature. Two physical scales are expressed, identified by the Einstein constant $\kappa = 8\pi G$ and the cosmological constant Λ.

Barker [**barker-thesis**] notes that, had the standard model predated general relativity, the theoretical minimalism of the Einstein–Hilbert action might have been called into question as it lacks quadratic terms. For example, consider the electromagnetic lagrangian density:

$$\mathcal{L}_{\mathrm{EM}} = -\frac{1}{4} F_{\mu\nu} F^{\mu\nu} + j_\mu A^\mu. \tag{5}$$

Barker uses Poincaré gauge theory (PGT), which gauges rotational, translation, and torsional fields to find an alternative to **??** constructed entirely from quadratic invariants of the Riemann-Cartan curvature $\mathcal{R}^{ijkl}$ and the torsion $\mathcal{T}^{ijk}$:

$$\begin{aligned} L_{\mathrm{G}} = &- \frac{4}{4\kappa} \mathcal{T}_i \mathcal{T}^i - \frac{\hat{\alpha}_6}{6} [\Lambda \mathcal{T}_{ijk} (\mathcal{T}^{ijk} - 2\mathcal{T}^{jik}) \\ &+ \mathcal{R}_{ij} (\mathcal{R}^{[ij]} - 12\mathcal{R}^{ij}) - 2\mathcal{R}_{ijkl} (\mathcal{R}^{ijkl} - 4\mathcal{R}^{ikjl} - 5\mathcal{R}^{klij})] + 2\hat{\alpha}_5 \mathcal{R}_{[ij]} \mathcal{R}^{[ij]}. \end{aligned} \tag{6}$$

The Einstein–Hilbert term has been replaced by the the square of the torsion contraction $\mathcal{T}_i \equiv \mathcal{T}^j_{ij}$, which self-couples with strength κ. The cosmological constant Λ is also a torsion self-coupling, rather than parameterising a mysterious energy density. The values of the dimensionless self-coupling constants $\hat{\alpha}_5$ and $\hat{\alpha}_6$ are not determined, but with the conditions

$$\Lambda > 0, \quad \hat{\alpha}_6 < 0, \quad (\hat{\alpha}_5 + 2\hat{\alpha}_6)(\hat{\alpha}_5 - \hat{\alpha}_6) > 0, \tag{7}$$

the linearisation of **??** on a matter–free, flat and torsionless background is power–counting renormalisable, unitary, and free from ghosts and tachyons. Ghosts are unphysical states in a gauge theory, which would be necessary if the local fields exceed the number of physical degrees of freedom. An example of a ghost exists in electromagnetism: one uses a four-vector potential $A_\mu(x)$, while the photon only has two polarisations [**Faddeev:2009**]. Tachyons are particles that travel faster than light [**tachyon**].

Equation **??** produces the same cosmology as **??**, but also allows a deviation from the vacuum expectation value which masquerades as the dark radiation described in equation **??** — a possible solution to the Hubble tension.

3 Computational Context

3.1 Bayes' theorem

Inductive logic, also known as plausible reasoning, is the problem faced by most scientists. Given that a certain set of effects have been observed, what model would have best predicted these? This is as opposed to the deductive logic usually employed in pure mathematics, and is typically a greater challenge. [**skilling**]

Starting from Cox's rules of probability one arrives at Bayes' theorem [**Sivia2006**]:

$$\Pr(B|A) = \frac{\Pr(A|B)\Pr(A)}{\Pr(B)}, \tag{8}$$

for events A and B. This can be framed as *"updating the prior"*:

$$\text{Posterior} = \frac{\text{Likelihood} \times \text{Prior}}{\text{Evidence}}. \tag{9}$$

The prior represents the knowledge of the parameters of a model, before undertaking an experiment. The likelihood is the probability of observing the results of that experiment, given a set of parameters. The

evidence, also known as the global likelihood, is the numerator of **??** marginalised over all possible parameters, and therefore normalises the expression. The posterior is then the updated knowledge of the parameters following the experiment.

Evaluating the evidence represents the key challenge of Bayesian analysis, and one must turn to numerical methods to evaluate the high-dimensional integral.

3.2 Numerical integration for evidence calculations

Numerical integration techniques in general could fill several reports by itself, so I will restrict discussion to techniques used in cosmological evidence calculations.

A Monte Carlo algorithm is a randomised algorithm, whose result will differ from the true result with known probability. Unlike deterministic approaches such as trapezoid rule, Monte Carlo algorithms are non-determministic. Monte Carlo integration uses the approximation

$$\int_\Omega f(x)dx \approx V\frac{1}{N}\sum_{i=1}^{N} f(x_i). \tag{10}$$

V is the volume of the integration domain Ω. Monte Carlo integration methods then differ by how the samples x_i are generated. The most vanilla Monte Carlo integration simply draws samples at random from x. This algorithm is at risk of missing small regions of Ω containing large contributions to the integral, which is the case for most high-dimensional integrals. This can be improved using importance sampling: drawing samples from a probability distribution which weighted towards the regions of high f. Markov chain Monte Carlo (MCMC) starts with a random point, and a Markov chain is constructed from it. There are many algorithms for constructing the chain, Metropolis Hastings, Gibbs sampling, slice sampling, to name a few.

As the number of dimensions increases, MCMC can too suffer from the "curse of dimensionality" it was designed to avoid. Shortening the Markov steps can help, but this leads to highly autocorrelated samples, so more samples are needed, and would therefore need to run for longer. MCMC is also not trivially parallelisable, aside from running multiple chains in parallel, which can be used to assess convergence. In cosmological applications, where the greatest computational cost is the generation of a new point, a different approach is needed: nested sampling.

4 Nested Sampling

4.1 The idea

Nested sampling is an algorithm developed by John Skilling for calculating the evidence Z, the probability of data given a particular model [**skilling**, **Ashton˙2022**]. Posterior samples are an optional by–product. This is typically in the context of Bayes theorem:

$$\text{Posterior} = P = \Pr(\boldsymbol{\theta}|D,\mathcal{M}) = \frac{\Pr(D|\mathcal{M}(\boldsymbol{\theta}))\times\Pr(\boldsymbol{\theta})}{\Pr(D|\mathcal{M})} = \frac{\text{Likelihood}\times\text{Prior}}{\text{Evidence}} = \frac{L(\boldsymbol{\theta})\pi(\boldsymbol{\theta})}{Z}, \tag{11}$$

for a model $\mathcal{M}$ and its parameters $\boldsymbol{\theta}$. The evidence Z is used to compare models $\mathcal{M}_i$ and $\mathcal{M}_j$ through the Bayes factor and posterior odds ratio:

$$\mathcal{P}_{i,j} = \ln\frac{\Pr(D|\mathcal{M}_j)}{\Pr(D|\mathcal{M}_i)}\mathcal{B}_{i,j} = \mathcal{B}_{i,j} + \ln\frac{\pi_{\mathcal{M}_i}}{\pi_{\mathcal{M}_j}}, \quad \mathcal{B}_{i,j} \coloneqq \ln\frac{Z_i}{Z_j}, \tag{12}$$

where Z_i is the evidence of model $\mathcal{M}_i$. The posterior odds is perhaps a more complete assessment of the relative performance of two models, but the Bayes factor is more common in the literature [**Sonke**]. The models I consider have equal priors, so $\mathcal{P}_{i,j} = \mathcal{B}_{i,j}$ [**Sonke**].

The Bayesian evidence is the normalisation of the numerator in Bayes' theorem, so in theory can be calculated by numerical integration. This becomes impractical to solve using typical numerical integration

techniques, such as rastering over $\boldsymbol{\theta}$, once $\boldsymbol{\theta}$ has many dimensions. Instead, define an element of prior mass, $dX = \pi(\boldsymbol{\theta})d\boldsymbol{\theta}$. X can be accumulated from dX in any order, so by defining:

$$X(\lambda) = \int_{L(\boldsymbol{\theta})>\lambda} \pi(\boldsymbol{\theta})d\boldsymbol{\theta}, \tag{13}$$

i.e. X is the cumulative prior mass for likelihood greater than λ. The evidence then becomes a one-dimensional integral:

$$Z = \int_0^1 L(X)dX, \tag{14}$$

where $L(X(\lambda)) = \lambda$. This transformation requires sorting the prior mass elements by likelihood. Given the set of m points:

$$0 < X_m < \cdots < X_2 < X_1 < 1, \quad L_i = L(X_i), \tag{15}$$

a sensible numerical recipe would estimate Z as a weighted sum:

$$Z \approx \sum_{i=1}^{m} w_i Z_i. \tag{16}$$

Because $L(X)$ is monotonically decreasing, it must be larger than any value at larger X, so $w_i = X_i - X_{i+1}$ gives a lower bound. As it must also be smaller than any value at smaller X, $X_{i-1} - X_i$ provides an upper bound:

$$\sum_{i=1}^{m}(X_i - X_{i+1})L_i \leq Z \leq \sum_{i=1}^{m}(X_{i-1} - X_i)L_i + X_m L_{\max}. \tag{17}$$

$L_{\max}$ is the maximum likelihood value found as $X \to 0$. A tiny volume containing huge likelihood could always exist, and as such $L_{\max}$ cannot be determined by any kind of sampling.

Z is dominated by the region of bulk posterior mass. This typically occupies a fraction e^{-H} of the prior, where:

$$H = \text{information} = \int \log\left(dP/dX\right) dP. \tag{18}$$

H can be the order of thousands in practical problems. By considering the likelihood to consist of C principal components, the posterior mass is expected to be broadly distributed over $-H \pm \sqrt{C}$ in $\log X$. Each principal component significantly restricts the range permitted by the prior, so H should usually exceed C, which suggests that finding the posterior domain is more difficult than navigating within it. Therefore, to cover such a range, sampling should be linear in $\log X$ rather than X, so instead set:

$$X_1 = t_1, X_2 = t_1 t_2, \ldots, X_m = t_1 t_2 t_3 \ldots t_m, 0 < t_i < 1. \tag{19}$$

It is not usually possible to set precise values of t, but setting them statistically is enough. At each step, we need to find a new X_i from the prior, subject to $X_i = t_i X_{i-1} < X_{i-1}$. The knowledge of this new point is specified by drawing t_i from Uniform$(0, 1)$. This could be obtained by:

1. sampling X_i uniformly from the range $[0, X_{i-1}]$ then using the original likelihood sorting to find its corresponding $\boldsymbol{\theta}_i$, or

2. sample $\boldsymbol{\theta}$ directly subject to $L(\boldsymbol{\theta}) > L_{i-1}$ in proportion to the prior.

These methods are equivalent, but (b) does not use the prior mass, so the sorting step is not needed.

4.2 Nested sampling algorithms

Several variants around Skilling's original nested sampling algorithm exist. In general, the posterior will be multi-modal, and is dealt with in different ways in different algorithm.

- `MultiNest`, developed by Prof. Hobson and Dr. Faroz, constructs a series of overlapping ellipsoids around the live points to approximate the isolikelihood contour. Multi–modal posteriors will cause the ellipsoids to separate, at which point the modes can be identified and evolved independently.
- `PolyChord` uses slice sampling to navigate the isolikelihood contour. It applies a clustering algorithm to the live points to identify posterior modes, so that each can be "whitened" independently. The modes then evolve semi-independently. `PolyChord` is discussed further in sections **??** and **??**.
- Dynamic nested sampling differs from traditional nested sampling by using a dynamic number of live points. This can yield greater number of posterior samples for the same computational cost of traditional nested sampling. `dyPolyChord` and `dynesty` implement this by beginning with a traditional run with a small number of live points. The algorithm then calculates the importance of each sample, then finds the first (j) and last (k) live points with greater than a fraction (default 0.9 in `dyPolyChord`) of the greatest importance. Nested sampling is then repeated with more live points, starting at likelihood L_{j-1} and finishing with the first sample taken with likelihood greater than L_{k+1}. This process is repeated until a termination condition is reached. The initial number of live points must be large enough to discover all modes of the posterior, as any modes with a maximum likelihood greater than L_{k+1} will not be explored properly on the subsequent runs. Therefore, it may be unsuitable for the applications discussed here, which are highly multi–modal. `PolyChord` also implements dynamic nested sampling, but puts responsibility on the user, who must provide a function $n_{\rm live}(L)$. Similarly, this function would be calculated using an initial low–resolution run. The low–$n_{\rm live}$ parts of the run will contribute larger Poisson error to the volume estimate, and therefore the evidence calculation. In practice, normal nested sampling provides plenty of posterior samples, so aside from very specific applications, dynamic nested sampling is unnecessary.

4.3 PolyChord

`PolyChord` is a nested sampling algorithm developed by my supervisors Dr. Handley, Prof. Hobson and Prof. Lasenby [**PolyChord˙1**, **PolyChord˙2**]. It is distinguished from historical implementations in how it performs the final part of the iterative step, drawing a new point from the prior subject to $L(\boldsymbol{\theta}) > L_{i-1}$. Skilling orignally envisaged a Markov-Chain procedure, taking steps according to a proposal distribution until an independent sample is produced. These intermediate "phantom" points have the potential to yield more information.

`PolyChord` uses slice sampling to sample within the isolikelihood contours. To slice sample, first choose a probability level "slice" P_0 within $[0, P_{\rm max}]$, then sample uniformally within the parameter space $P(\boldsymbol{\theta}) > P_0$. `PolyChord` adds features atop Aitken and Akman's slice-based nested sampling. It uses information present in both the live and phantom points to deal with correlated posteriors, and a clustering algorithm to identify separate posterior modes and evolve them semi-independently, calculating local evidences for each of them. It also has the option to implement fast-slow parameters, which is effective when sampling cosmologies.

The `PolyChord` algorithm is written in FORTRAN, and parallelised using `OpenMPI`. `PolyChordLite`[1] uses C++ to provide a Python interface, which I have used exclusively for my cosmological work. It is optimised for the case where runtime is dominated by the cost of generating new live points. This is frequently encountered in cosmology, due to the high dimensionality and costly likelihood evaluation.

4.4 Slice sampling

For each iteration i of nested sampling, a point within the iso–likelihood contour L_i is generated. This sampling is performed in the unit hypercube, with coordinate $\boldsymbol{x}$. One of the live points is chosen at random

[1] https://github.com/PolyChord/PolyChordLite

as a start point for a new chain, with coordinate $\boldsymbol{x}_0$. A step is then taken in the direction $\hat{\boldsymbol{n}}_0$ with distribution $P(\hat{\boldsymbol{n}})$ with width w, to generate the point $\boldsymbol{x}_1$. This is repeated n_{repeats} times.

`PolyChord` uses the covariance matrix Σ of the set of live and phantom points to infer the size and shape of the iso-likelihood contour, and determine an optimal $P(\hat{\boldsymbol{n}})$ and w. Uniformly transformed points remain so after an affine transformation:

$$\mathrm{L}^{-1}\boldsymbol{x} = \boldsymbol{y}, \qquad \mathrm{L}\mathrm{L}^{\intercal} = \Sigma, \tag{20}$$

where L is the Cholesky decomposition of the covariance matrix. The new space containing $\boldsymbol{y}$ is termed the "sampling space". This transformation gives the contour size $\mathcal{O}(1)$ size in every direction, "whitening" it, so $w = 1$ is a suitable initial step size. $P(\hat{\boldsymbol{n}})$ is inspired by CosmoMC, which chooses a randomly oriented orthonormal basis, uses these directions in a random order, and generates a new one when this is exhausted [**CosmoMC**].

4.5 Clustering

The following step in nested sampling is clustering. A multi–modal distribution cannot be explored properly without grouping live points by their separate iso–likelihood contours, and exploring each mode independently. These modes are identified by the clustering of live points. Performing this at every iteration would add a large overhead to the calculation; in practice checking every $\mathcal{O}(n_{\text{live}})$ iterations is enough, as the live points will have compressed only by a factor of e [**PolyChord˙1**]. The algorithm must be capable of identifying the number of clusters, as this is unknown. `PolyChord` uses a recursive variant of the K–nearest–neighbours, which satisfies this property. In principle, any such clustering algorithm should be sufficient, as the conditions of nested sampling mean there is no noise as each of the live points will be in exactly one of the iso–likelihood contours. When a live point dies, the new live point may appear in any of the surviving clusters, weighted by the prior volume of each cluster. A cluster dies when it contains zero live points.

4.6 Displaying results

I have used the Python packages `anesthetic` for reading nested sampling chains into Python and producing posterior plots, and `fgivenx` for producing the functional posterior plots [**anesthetic**, **fgivenx**]. The functional posterior plots produce a plot of some function $f(x, \boldsymbol{\theta})$ marginalised over the posterior samples of $\boldsymbol{\theta}$. Examples of this are shown in figures **??**, **??**, **??** and **??**. This provides more insight than simply plotting f for the "best fit" $\boldsymbol{\theta}$, as there might exist different modes of solutions that would otherwise be missed.

4.7 Interpreting results from different datasets

As mentioned in section **??**, two models may be compared through either their Bayes factor or posterior odds ratio. However, one may also wish to compare two datasets A and B, to assess whether they are in tension. This is usually achieved through their evidence ratio statistic R [**Marshall˙2006**]:

$$R = \frac{Z_{AB}}{Z_A Z_B}, \tag{21}$$

where Z_A and Z_B are the evidences calculated using datasets A and B, and Z_{AB} combines the datasets. $R \gg 1$ is interpreted as the two datasets are consistent, while $R \ll 1$ implies the datasets are consistent. For example, this is how the tension between the Dark Energy Survey (DES) and Planck measurements of Ω_m and σ_8 (see discussion of S_8 tension in **??**) is assessed [**suspiciousness**].

The Kullback–Leiber (KL) divergence quantifies the information gain, i.e. the compression of the posterior relative to the prior:

$$\mathcal{D}_D = \int P(\boldsymbol{\theta}|D) \log \frac{P(\boldsymbol{\theta}|D)}{\pi(\boldsymbol{\theta})} d\boldsymbol{\theta}. \tag{22}$$

$\log\left[P(\boldsymbol{\theta}|D)\right]/\pi(\boldsymbol{\theta})$ is the Shannon information [**shannon**] provided by the posterior relative to the prior. The KL divergence may be used to define the information ratio I:

$$\log I = \mathcal{D}_A + \mathcal{D}_B - \mathcal{D}_{AB}. \tag{23}$$

The ratio of the evidence ratio and the information ratio quantifies the mismatch between the datasets, termed *suspiciousness* S [**suspiciousness**]:

$$\log S = \log R - \log I \tag{24}$$

Suspiciousness is unaffected by changing the prior widths, since I and R change similarly under prior transormations, provided the posterior is not significantly altered. Suspiciousness is free from the prior–dependent properties of R, which has been misused in previous works. However, a true Bayesian should consider this kind of prior dependence a feature of R, rather than a bug. In fact, we should consider that if there are *any* reasonable priors which result in $R \ll 1$, then the datasets are in tension.

5 Toy Examples

To develop my understanding of `PolyChord`, I repeated the work by S. Hee et al. [**Sonke**] This work uses linear splines as empirical models whose parameters are found using `PolyChord`, beginning with two toy datasets. One is $\sin(x/2\pi)$ plus Gaussian noise, the other a "line" wave, consisting of straight lines passing though $(1/4, 1)$ and $(3/4, -1)$ with period 1. I added a third "flat" case, replacing the line or sine with zero. For each case, I considered both errors in just the y–coordinate, and in both x and y. A *linf* can represent the line and flat cases exactly with $N = 1$ and $N = 4$ respectively.

To create each dataset $\{x_i, y_i\}$, 50 x–coordinates were drawn uniformly from $[0, 1]$, to which I applied one of the three functions to give the y–coordinate, and finally added homoskedastic Gaussian noise with $\sigma = 0.05$ in either just the y or both directions. Any points taken out of $0 \leq x < 1$ by the noise were deleted. My code is also vectorised to handle heteroskedastic errors.

Two types of linear spline are used. The first is a "vanilla" linear spline which consists of N nodes, labelled 0 through $N-1$. The nodes must be sorted in increasing x–coordinate, the end nodes (0 and $N-1$) have fixed x–coordinate, but are otherwise free to move. The second is an "adaptive" linear spline. N is instead a parameter, which controls the number of nodes to use out of a maximum $N_{\max}$. For example, if $N = 5$ and $N_{\max} = 8$, nodes 0, 1, 2, 3, and 4 are treated as normal, but node 4 is connected directly to node 7, ignoring 5 and 6. If $N = 1$, then the spline is constant at the height of the $N - 1^{\text{th}}$ node. This is an example of a nested model; vanilla splines with N ranging from 1 to $N_{\max}$ are nested with each other. In practice, `PolyChord` sees N as a continuous parameter which is then rounded down. Note that the N used in this and S. Hee's work is different, here it refers to the total number of nodes, as opposed to the number of internal nodes.

These two splines allow two different approaches to the problem. In the vanilla case, a separate `PolyChord` run is performed with N from 1 to 9, creating samples of the 9 posteriors. I used the Python package `fgivenx` to marginalise the linear splines to produce contour plots, both separately for each N, and then combined them weighted by relative evidence to produce an overall plot.

The adaptive case instead requires one `PolyChord` run, whose posterior samples can be used to produce a similar marginalised spline, equivalent to combining the vanilla cases. While the adaptive posterior can be split by N values to recreate the vanilla samples, these may not be of high quality. Nested samplers rapidly converge to peaks in the distribution, so will spend less time sampling regions of improbable N. This is clearest for the "line" case, since the true solution is $N = 2$, so `PolyChord` spends more time sampling in that region than $N = 1$. This is intended — these regions are sampled sufficiently well to identify that they are less probable, and further computing time can be focussed on finding the most probable models.

`PolyChord` requires a (log) likelihood function of a single parameter vector θ, so the x– and y–coordinates of the nodes would need to be arranged in some way. Keeping the parameters in node order was the most straightforward, i.e

$$\boldsymbol{\theta} = [y_0, x_1, y_1, x_2, y_2, ..., x_{(N-2)}, y_{(N-2)}, y_{(N-1)}]$$

for the vanilla case, then

$$\boldsymbol{\theta}_{\text{adaptive}} = [N, y_0, x_1, y_1, x_2, y_2, ..., x_{(N_{\max}-2)}, y_{(N_{\max}-2)}, y_{(N_{\max}-1)}]$$

for adaptive. This convention requires some slicing gymnastics, for example the `PolyChord` provided sorted–uniform and uniform priors need to be interleaved, and the NumPy linear spline `interp` takes the x– and y–coordinates as two separate arrays. Therefore, it was appropriate to package the splines, priors and likelihoods into a separate pip-installable repository, so once I got the slicing right I would never have to think about it again.

For the prior, I took the nodes to be sorted–uniform in x in $[0,1]$ and uniform in y in $[-1.5,1.5]$. The amplitude of the y prior was simply chosen to be greater than the amplitude of any of the data points, without being so large as to waste computation time.

This just leaves calculating the likelihood. This is straightforward for the y-errors case:

$$L\left(\theta\right)=\prod_{i=1}^{M}\frac{1}{\sqrt{2\pi\sigma_{y_i}^2}}\exp\left[-\frac{\left(y_i-\mathrm{linf}\left(x_i,\theta\right)\right)^2}{2\sigma_{y_i}^2}\right], \tag{25}$$

with M data points. Taking logs then reduces the product into a sum, which can be easily evaluated. Including the abscissa errors makes the evaluation more involved as one must marginalise over X_i, the possible "true" value of the abscissa given the datum x_i:

$$L\left(\theta\right)=\prod_{j=1}^{M}\int_{X_-}^{X_+}dX_j\frac{\exp\left[-\frac{\left(x_j-X_j\right)^2}{2\sigma_{x_j}^2}-\frac{\left(y_j-\mathrm{linf}(X_j)\right)^2}{2\sigma_{y_j}^2}\right]}{2\pi\sigma_{x_j}\sigma_{y_j}\left(X_+-X_-\right)}, \tag{26}$$

where X_- and X_+ are the limits of x, 0 and 1 here. I split this integral by summing each straight section of the *linf*, which can be each be reparameterised as straight lines in the usual way:

$$\begin{aligned}f_k(x)&=m_kx+c_k,\qquad m_k=\frac{\theta_{y_{k+1}}-\theta_{y_k}}{\theta_{x_{k+1}}-\theta_{x_k}},\qquad c_k=\theta_{y_k}-m_k\theta_{x_k}\\ \implies L\left(\theta\right)&=\prod_{j=1}^{M}\sum_{k=0}^{N-2}\int_{\theta_{x_k}}^{\theta_{x_{k+1}}}dX_j\frac{\exp\left[-\frac{\left(x_j-X_j\right)^2}{2\sigma_{x_j}^2}-\frac{\left(y_j-(m_kX_j+ck)\right)^2}{2\sigma_{y_j}^2}\right]}{2\pi\sigma_{x_j}\sigma_{y_j}\left(X_+-X_-\right)},\end{aligned} \tag{27}$$

where θ_{x_k} and θ_{y_k} are the x and y coordinates of the k^{th} of N nodes in the *linf*. By completing the square, and scaling the integration variable, the integrals can be rewritten as the difference of error functions:

$$\begin{aligned}L&=\prod_{j=1}^{M}\frac{1}{2\sqrt{2\pi}\left(X_+-X_-\right)}\sum_{k=0}^{N-2}\frac{\exp\left[-\frac{\left(m_kx_j+c_k-yj\right)^2}{2\left(m_k^2\sigma_{x_j}^2+\sigma_{y_j}^2\right)}\right]}{\sqrt{m_k^2\sigma_{x_j}^2+\sigma_{y_j}^2}}\left[\mathrm{erf}\left(t_{jk}(\theta_{x_{k+1}})\right)-\mathrm{erf}\left(t_{jk}(\theta_{x_k})\right)\right],\\ t_{jk}(X)&=\sqrt{\frac{m_k^2\sigma_{x_j}^2+\sigma_{y_j}^2}{2\sigma_{y_j}^2\sigma_{y_j}^2}}\left[X-\frac{x_j\sigma_{y_j}^2+(y_j-c_k)m_k\sigma_{x_j}^2}{m_k^2\sigma_{x_j}^2+\sigma_{y_j}^2}\right],\qquad\mathrm{erf}(t)=\frac{2}{\sqrt{\pi}}\int_0^t e^{-x^2}dx.\end{aligned} \tag{28}$$

Care must be taken when taking logs, the most appropriate approach is to use the "logsumexp" trick to ensure computational accuracy:

$$\begin{aligned}\mathcal{L}=\ln L&=\sum_{j=1}^{M}-\ln\left(2\sqrt{2\pi\left(X_+-X_-\right)}\right)\\ &+\sum_{j=1}^{M}\left[\mathrm{logsumexp}_k\left\{-\frac{(m_kx_j+c_k-yj)^2}{2\left(m_k^2\sigma_{x_j}^2+\sigma_{y_j}^2\right)},\left(\frac{m_k^2\sigma_{x_j}^2+\sigma_{y_j}^2}{\sigma_{y_j}^2\sigma_{y_j}^2}\right)^{-\frac{1}{2}}\left[\mathrm{erf}\left(t_{jk}(\theta_{x_{k+1}})\right)-\mathrm{erf}\left(t_{jk}(\theta_{x_k})\right)\right]\right\}\right].\end{aligned} \tag{29}$$

Since the elements of the first sum are all the same, it can just be multiplied by the size of the dataset. The logsumexp function is defined by:

$$\mathrm{logsumexp}_j\left(x_j,b_j\right)=\ln\sum_j b_je^{x_j}=x_{\max}+\ln\sum_j b_j\exp\left(x_j-x_{\max}\right), \tag{30}$$

Similarly to how multiplication becomes addition in log-scale, additions become logsumexp, which is frequently encountered when dealing with log probabilities. Typical cosmological log Z values are $\mathcal{O}(-1000)$, so their exponentials are much smaller than e.g. the smallest representable number in IEEE double precision, $\approx 4.95\times10^{-324}$, so just combining the log, sum and exp functions in the appropriate way will lead to overflow errors [**IEEE**]. Assuming the x_i to be of similar magnitude, by writing logsumexp in the way shown in equation **??** pulling a factor of $e^{x_{\max}}$ from the sum will make the exponentials representable, and therefore maintain better precision [**logsumexp**].

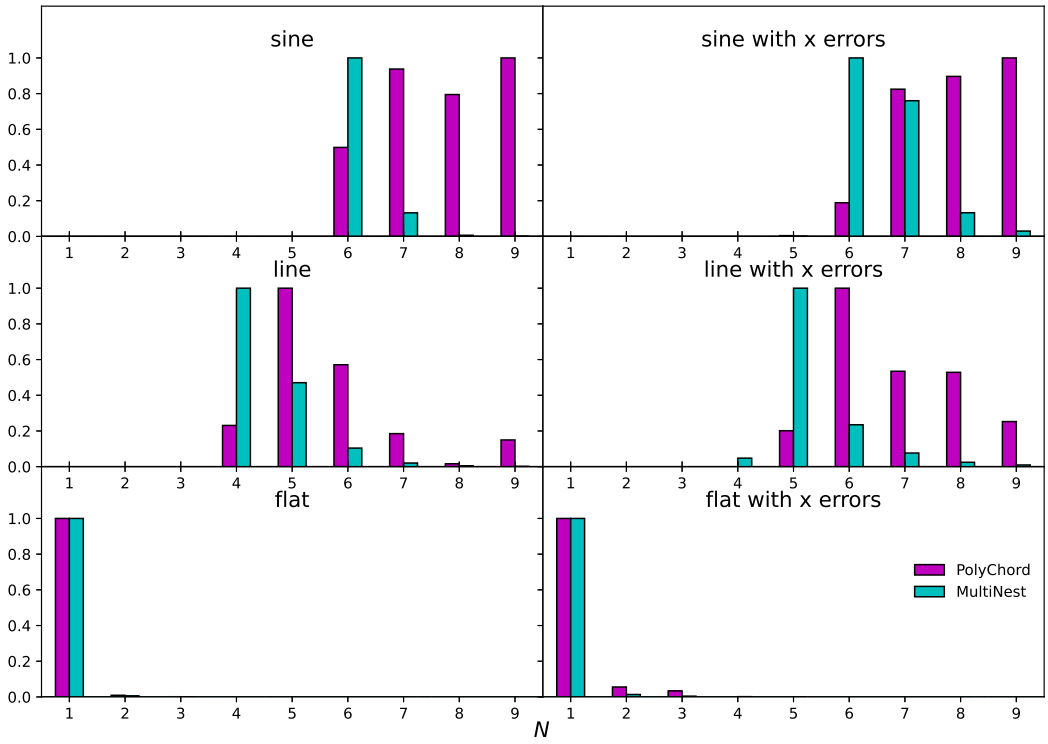


Figure 1: Histogram of posterior samples for adaptive *linf*, for sine, line and flat cases, using either `PolyChord` or `MultiNest`. Left: y errors only. Right: both x and y errors.

The posterior distribution of N for `AdaptiveLinf` are shown in figure **??**, and their respective functional posterior plots in figure **??**. As expected, the flat case favours $N = 1$. However, the line case doesn't favour $N = 4$, instead preferring five or six nodes without and with x errors respectively. Substituting the nested sampler `MultiNest` recovers $N = 4$ at least in the y errors only case; the posterior of N is also shown in figure **??**. However, `MultiNest` also favours 6 nodes for the sine wave, when one would expect the maximum number of nodes to be favoured. I am investigating this further.

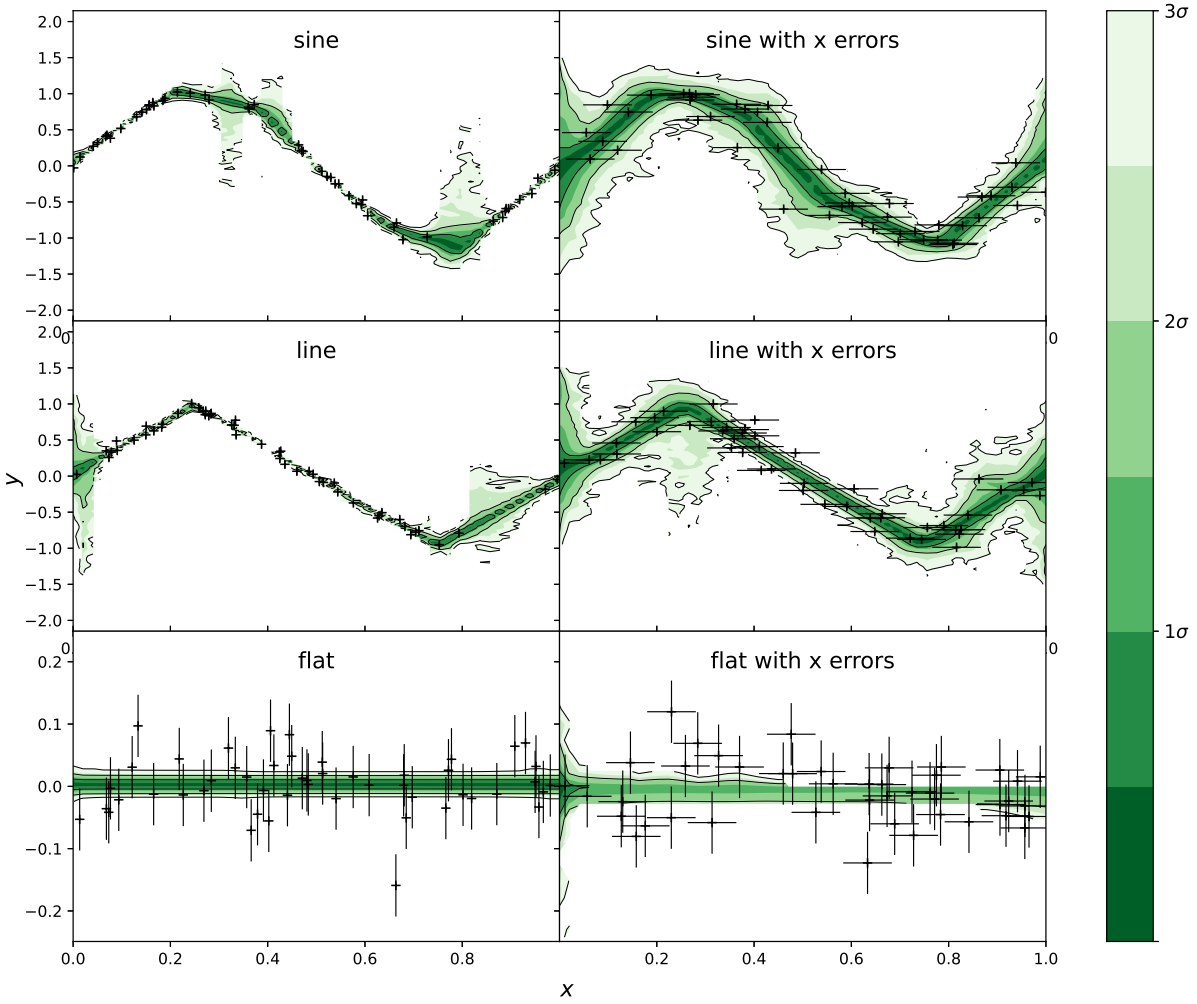


Figure 2: Functional posterior contour plots of adaptive *linf* posterior samples for sine, line and flat cases using `PolyChord`. Left: y errors only. Right: both x and y errors. Note the different scale for the line case.

6 Cosmological Applications

6.1 Cosmological codes

6.1.1 Cobaya

`Cobaya` (**Co**de for **bay**esian **a**nalysis) is a sampling and modelling framework, intended for use with cosmology [**Cobaya**]. It is written in Python, and provides interfaces to work with popular cosmological models `CAMB` and CLASS, the Planck likelihoods, and can sample using either a built-in MCMC algorithm, or the nested sampler `PolyChord`. `Cobaya` takes its input from a `.yaml` file, which specifies which models and likelihoods to use, and the priors for each of the parameters.

6.1.2 CAMB

`CAMB` (Code for Anisotropies in the Microwave Background) is the Cosmological Boltzmann code I have used exclusively so far [**CAMB˙1**, **CAMB˙2**]. It is one of two Boltzmann codes to which `Cobaya` can interface, the other is `CLASS` (Cosmic Linear Anisotropy Solving System, often labelled "`classy`" to distinguish it from the class keyword). These codes calculate the theoretical CMB power spectrum for a given set of cosmological parameters, such as the Hubble constant H_0, primordial matter power spectrum $\mathcal{P}_\mathcal{R}(k)$, and the dark energy equation of state parameter w. I chose `CAMB` over `CLASS` as `CAMB` was used in the works I have been replicating, and currently `CAMB` is faster than `CLASS`, though this is subject to change as each of the codes continues to be updated.

`CAMB` is written in object-oriented FORTRAN90 and interfaced via Python, which allows `CAMB` to be used interactively with the speed of compiled code. It can be parallelised using `OpenMP` (Open Multi-Processing), with almost linear scaling up to 16 processes [**camb˙mp**].

6.2 Primordial matter power spectrum

To check that I am using `Cobaya` and `CAMB` correctly, I repeated the work in [**Handley2019**], where the primordial matter power spectrum is reconstructed using a linear spline. This is why I created my *linf* package, as I can reuse it here.

The primordial matter power spectrum has the form:

$$\mathcal{P}_\mathcal{R}(k) = A_s \left(\frac{k}{k_\text{pivot}}\right)^{n_s-1}, \tag{31}$$

where n_s is the scalar spectral index, and A_s is the comoving curvature power at $k = k_\text{pivot}$ Then, taking logs of both sides, and converting to $\log_{10} k$ to match the literature and the original work [**Handley2019**], it can be seen that it is linear in log-log space, and therefore suitable for a linear spline reconstruction:

$$\ln \mathcal{P}_\mathcal{R} = \ln A_s + (n_s - 1) \ln \frac{k}{k_\text{pivot}} = \ln A_s + (n_s - 1) \log_{10} \frac{k}{k_\text{pivot}} \ln 10. \tag{32}$$

$\mathcal{P}_\mathcal{R}(k)$ is of $\mathcal{O}\left(10^{-10}\right)$, but it is good practice in computation to work with parameters of $\mathcal{O}(1)$. Therefore, $\ln\left(10^{10}\mathcal{P}_i\right)$ was used as the y–parameter of the *linf*. My code uses this scaling implicitly, i.e. `lnPi` refers to $\ln\left(10^{10}\mathcal{P}_i\right)$.

$$\mathcal{P}_\mathcal{R}\left(\left\{\log_{10} k_i, \ln\left(10^{10}\mathcal{P}_i\right)\right\}\right) = 10^{-10} \exp\left[\text{linf}\left(\left\{\log_{10} k_i, \ln\left(10^{10}\mathcal{P}_i\right)\right\}\right)\right]. \tag{33}$$

Unlike the toy example where I was working with `PolyChord` directly, `Cobaya` understands each of the model parameters separately. Also, what `PolyChord` calls a "prior" is the inverse cumulative distribution function of the prior, while `Cobaya` understands priors more literally. Applying this inverse to points uniformly distributed in the unit hypercube $[0,1]^D$ returns points distributed according the the prior. Therefore, the interleaved uniform/sorted-uniform prior in my `linf` package cannot be used here. Instead, each of the $\log_{10} k_i$ parameters is given its own uniform prior, with the sorted constraint implemented in a separate prior, e.g:

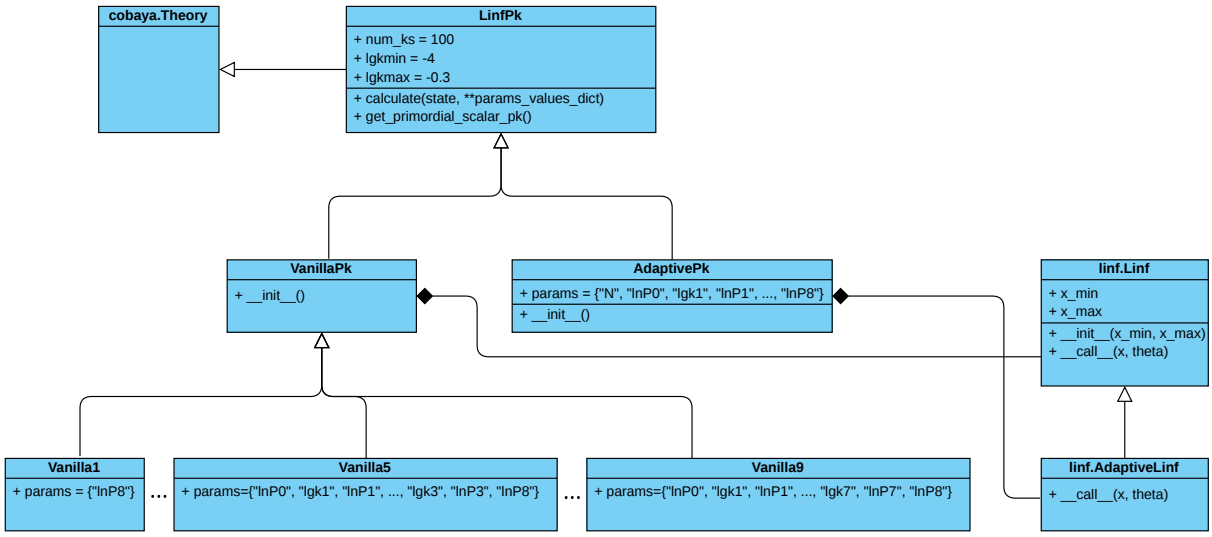


Figure 3: Class diagram for the Theory subclasses used to reconstruct the primordial matter power spectrum with a linear spline. `LinfPk` defines the `calculate` and `get_primordial_scalar_pk` methods, then the constructors for `VanillaPk` and `AdaptivePk` specifies which type of `linf` to use. `AdaptivePk` also specifies its `params` class attribute, while for the nine vanilla cases this is specified by the children of `VanillaPk`.

```
lambda lgk1, lgk2, lgk3, lgk4, lgk5, lgk6, lgk7: np.log(lgk1 < lgk2 < lgk3 < lgk4
  < lgk5 < lgk6 < lgk7)
```

for nine total nodes, where lg is shorthand for $\log_{10}$. This, along with including the parameters in the parameter block, made for laborious typing (even with Python's compact comparison operator chaining), so I automated writing the yaml files with a Python script.

The likelihoods used in the toy examples of section **??** are similarly also not suitable for use by `Cobaya`, plus the highly non–trivial task of turning the CMB measurements from Planck into a likelihood function for dark energy. Fortunately `Cobaya` interfaces with official clik code, which calculates these likelihoods. [**planck_likelihoods_1**, **planck_likelihoods_2**]. I used the following likelihoods: lensing, low–l EE and TT, high–l $TTTEEE$ and Sunyaev–Zeldovich (SZ). `Cobaya` takes care of the installation of the codes, which just need to be mentioned in the likelihood block of the input yaml file.

`Cobaya` has a built-in option for the user to provide their own $\mathcal{P}_\mathcal{R}(k)$. This is done by specifying `external_priomoridal_pk: True` in the `CAMB` block, and providing the path to a class which will provide $\mathcal{P}_\mathcal{R}(k)$. This class must inherit from `cobaya.Theory`, needs a `calculate` method which adds the k and $\mathcal{P}_\mathcal{R}(k)$ values to the state dictionary, and a `get_primordial_scalar_pk` method which returns this part of the state. The class variable `params` specifies the parameters required for the calculation. `CAMB` then internally uses a cubic spline between these values.

I expressed my ten different `Theory`s (nine vanilla plus adaptive) using inheritance. The `LinfPk` inherits from `Theory`, and defines the `calculate` and `get_primordial_scalar_pk` methods. `calculate` depends on `self.linf`, which is specified in its child classes `VanillaPk` and `AdaptivePk`. `AdaptivePk` is complete as it specifies its own `params`, while `Vanilla[1-9]` specify them for `VanillaPk` for the nine vanilla cases. A class diagram is shown in figure **??**. The functional posteriors from the vanilla cases are shown in figure **??**, using 100 `PolyChord` live points. Their respective evidences are shown in figure **??**, along with the corresponding values for the adaptive case, split by N. The two sets are clearly different, in particular the most unlikely cases of $N = 1$ and $N = 9$. For the adaptive case, these each have an effective sample size of approximately one. This is because PolyChord has successfully identified them as low evidence modes, and has saved computation time by not sampling them in more detail.

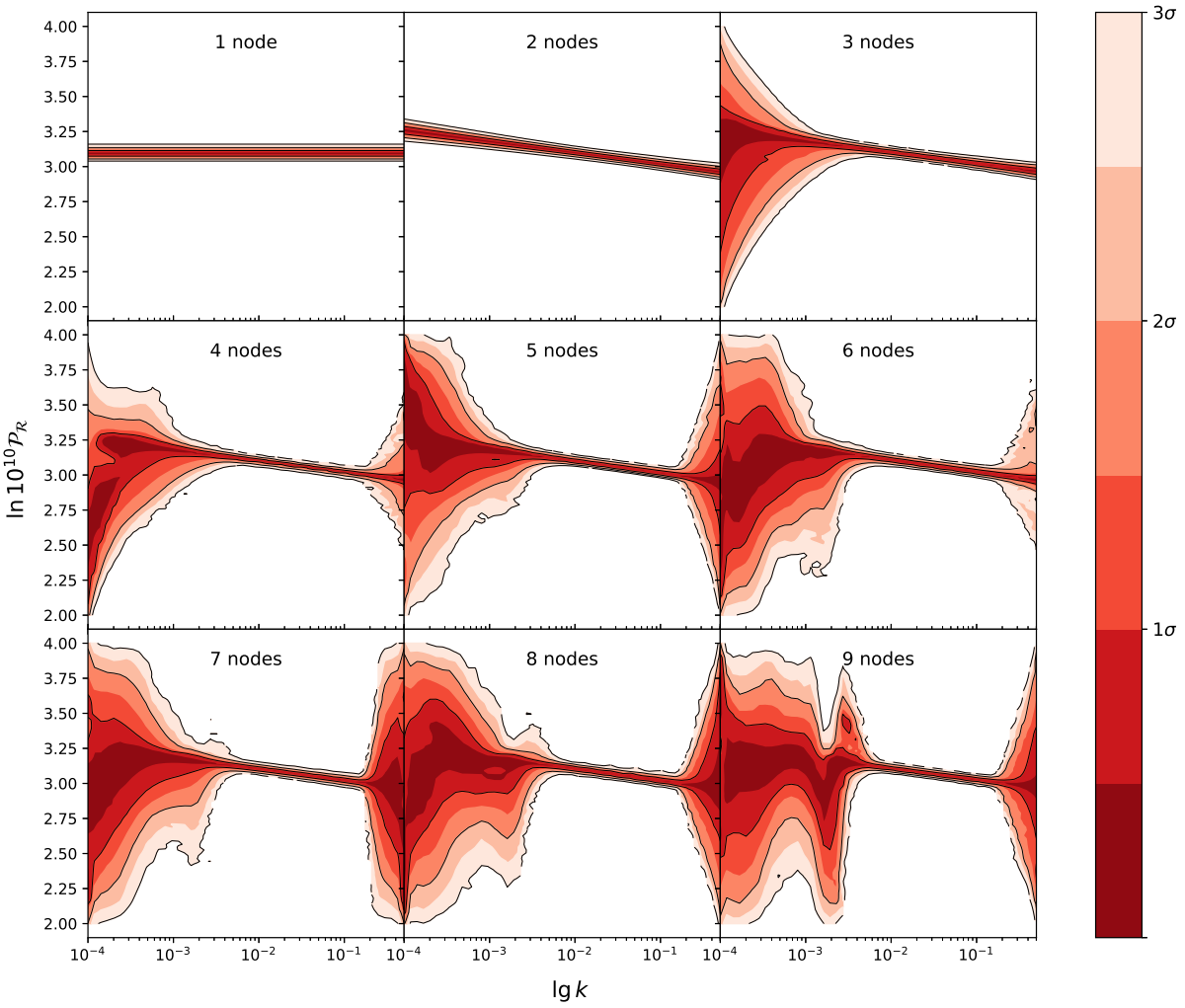


Figure 4: Functional posterior contour plots for the *linf* reconstruction of the primodial matter power spectrum, with fixed numbers of nodes from 1 to 9. 100 live points was used for each of these runs.

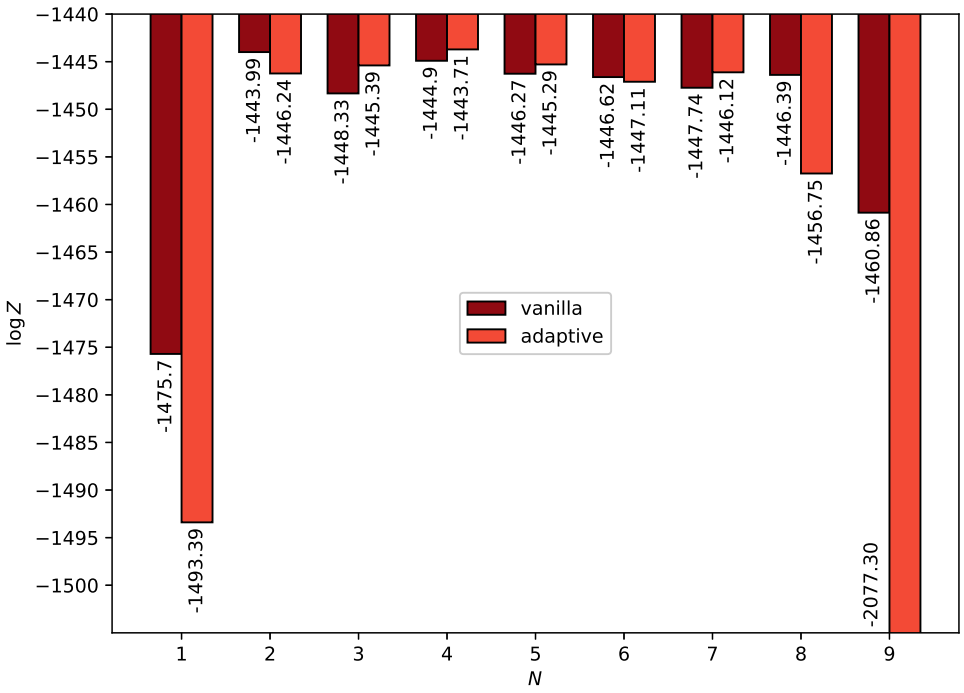


Figure 5: Evidence values for the primordial matter power spectrum for each of the vanilla reconstructions shown in figure **??**, and the adaptive reconstructions, split by N value to calculate $\log Z$ for each.

6.3 Dark energy

All the exercise with toy sine models and the primordial matter power spectrum was building up to investigate the dark energy equation of state. The equation of state of a perfect fluid is characterised the ratio of its pressure to its density: $w = p/\rho$. The goal is to perform a similar spline "*linf*" reconstruction to $\mathcal{P}_\mathcal{R}(k)$ with $w(a)$.

Since dark energy exists as an explanation for the cosmological constant, i.e. a fluid with $w = -1$, this needs to be reflected in the prior and `AdaptiveLinf`. Therefore, I modified my `AdaptiveLinf` so that for $N = 0$ (or more accurately, $0 \leq N < 1$) it is constant at -1. This is more appropriate than the cosmological constant being only the $w_{N_{\max}-1} = 1$, $1 \leq N < 2$ element of the entire prior volume.

The primordial matter power spectrum is unique in `Cobaya` in having the `external_primordial_pk` option. The primordial power spectrum is separate from the rest of the parameters passed to `CAMB`, which are either used to compute the transfer functions or specify non–linear behaviour. The transfer functions are then applied to the power spectrum to compute the spectra we see today. This means that the primordial power spectrum can be changed without needing to recompute the transfer functions, so can be sampled independently.

Changing dark energy is less straightforward as `Cobaya` only supports the cosmological constant with $w = -1$, $w =$ constant with some other value, or $w(a) = w + (1-a)w_a$. `CAMB` includes a `set_dark_energy_w_a()` method to supply a table of w values. My goal was then to echo the primordial power spectrum functionality with a `external_wa` option, which takes values of w and a from a `cobaya.Theory` with a `get_dark_energy()` method, and provides these to `CAMB`. Like $\mathcal{P}_\mathcal{R}(k)$, `CAMB` internally performs a cubic interpolation between these values.

`CAMB` suffers segmentation faults if the $w(a)$ table starts at $a = 0$. The workaround suggested by the developers was to start at a low value of a, they suggested 10_{-10}. It also only accepted float values for a and w, using $a = 1, w = -1$ would also cause a fault. Therefore, I added a validation step to the `CAMB` Python interface which checks that the array of a are all greater than zero, and casts them to float. This is now part

of the master branch of `CAMB`[2].

I chose a prior of a_i sorted-uniform in $[0.1, 1]$, and w_i uniform in $[-2, -0.01]$. As this range extends below -1, termed "Phantom energy", this requires using the Parameterised Post-Friedmann dark energy, as a model with a single scalar field cannot cross $w = -1$ [**PPF**].

The challenging part here was to understand the correct place in the `Cobaya` code to add this change. As $w(a)$ needs to be set with the other `CAMB` transfer function parameters, I could not copy the existing code for $\mathcal{P}_\mathcal{R}(k)$. Instead, I ran `Cobaya` in debug mode with $w =$ const. to find where it provides this value to `CAMB`. Once I found it (`cobaya.theories.camb.camb.CAMB.set()`), I appended a check for `external_wa`, and if so then pass the values of w and a to the instance of `CAMBparams`, the object which stores the parameters for `CAMB`. This didn't work, which manifested itself as either segmentation faults or getting very different results with a flat *linf* ($N = 0$) when compared to varying w directly. This is because the order in which parameters are provided to `CAMB` is important. Dark energy should be set **first**, as when `CAMB` is provided with `theta_MC_100` ($100\times$ the ratio of the sound horizon to the angular diameter distance), it then uses dark energy to calculate H_0. It is more efficient to sample `theta_MC_100` than H_0 as it is less correlated with other parameters [**CosmoMCReadMe**]. Therefore, I changed `set()` so it creates a fresh instance of `CAMBparams`, sets the table of w and a, then sets the the rest of the cosmology so that H_0 is computed correctly.

To avoid any issues arising from using the internal cubic spline (which the `CAMB` documentation notes is quite slow [**CAMBdocs**]), I created several branches of `Cobaya` in parallel. *external_wa* matches the description above, *flat_wa* takes the last value from the *linf* and sets w directly, and *linear_wa* takes the first and last values from the *linf*, uses these to calculate w and w_a, and sets those. I compared the *flat_wa* branch against the Planck 2018 MCMC chains, and *linear_wa* against an unmodified `Cobaya` allowing w_a to vary, with present day $w = -1$. Only once I was convinced these were consistent did I then use the full *external_wa* version.

At the start of each run, `Cobaya` times each of the theories, and works out the appropriate oversampling factor for each of them. At the moment, I have been manually specifying in the input yaml file for `Cobaya` that the dark energy parameters should be sampled with the rest of the cosmological parameters, rather than separately, as I have found this roughly halves the runtime. I would like to automate this, as doing this manually requires the user to also specify the oversampling factors.

Due to the large computational cost associated with performing each run, and the known issues with `PolyChord` clustering, I have only produced low resolution runs so far. I used 80 live points rather than the 100 for the primordial power spectrum, since this could reliably finish within a single 12 hour run for *flat_wa* and *linear_wa*. I am also finding that, when `anesthetic` reads the `PolyChord` chains, a few ($\mathcal{O}(1\%)$) of the samples have NaN weight and loglikelihood. This can be worked around by simply dropping those rows and recalculating, but it is important I understand why this is happening.

I have included some provisional posterior plots in figure **??**, for $N_{\max} = 1$, 2, 3, 4 and 9. The posterior distributions of N are shown in figure **??**. To also save running time while working and testing, I continued to only use the Planck 2018 likelihoods that I used with the primordial matter power spectrum. However, these do not provide a tight bound for dark energy, with $w = -1.57^{+0.50}_{-0.40}$ from Planck's own analysis. [**planck'6**]. This makes sense, the CMB is from the early universe, and therefore not strongly coupled to the late–time acceleration of the universe's expansion due to dark energy. Functional posteriors are shown in figure **??**. In S. Hee's work, they also use the Baryonic Acoustic Oscillation (BAO) and IA supernovae likelihoods, which both provide measurements of the Hubble parameter, which is strongly dependent on dark energy [**Sonke**, **BAO1**, **BAO2**, **BAO3**, **SNIa**]. The functional posterior including these likelihoods for $N_{\max} = 9$ is shown in figure **??**, and the N posterior is also shown in the sixth box of figure **??**. Here the cosmological constant ($N = 0$, $w = -1$) case dominates. I will not draw any definitive conclusions until the clustering issues with `PolyChord` have been solved, so that I can rerun this with higher resolution.

[2] https://github.com/cmbant/CAMB

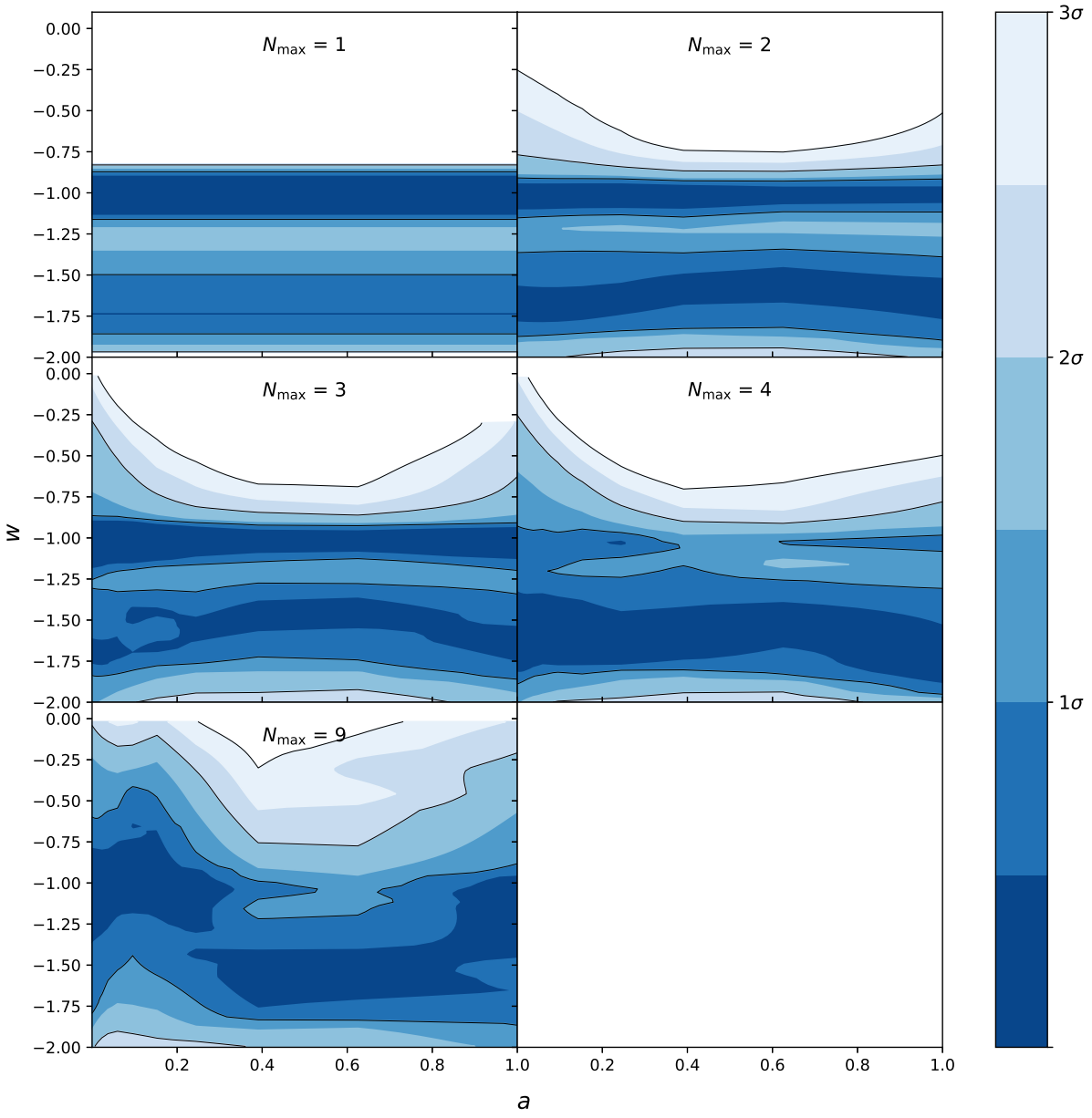


Figure 6: Posterior plots using an `AdaptiveLinf` as a model for the dark energy equation of state parameter, with different values of $N_{\max}$. These use the Planck 2018 likelihoods only. For $N_{\max} \leq 3$, $w = -1$ is still a dominant mode, as $N_{\max}$ increases further, values around -1.7 to -1.5 begin to appear. This is in line with $w = -1.57^{+0.50}_{-0.40}$ calculated by Planck 2018 [**planck˙6**].

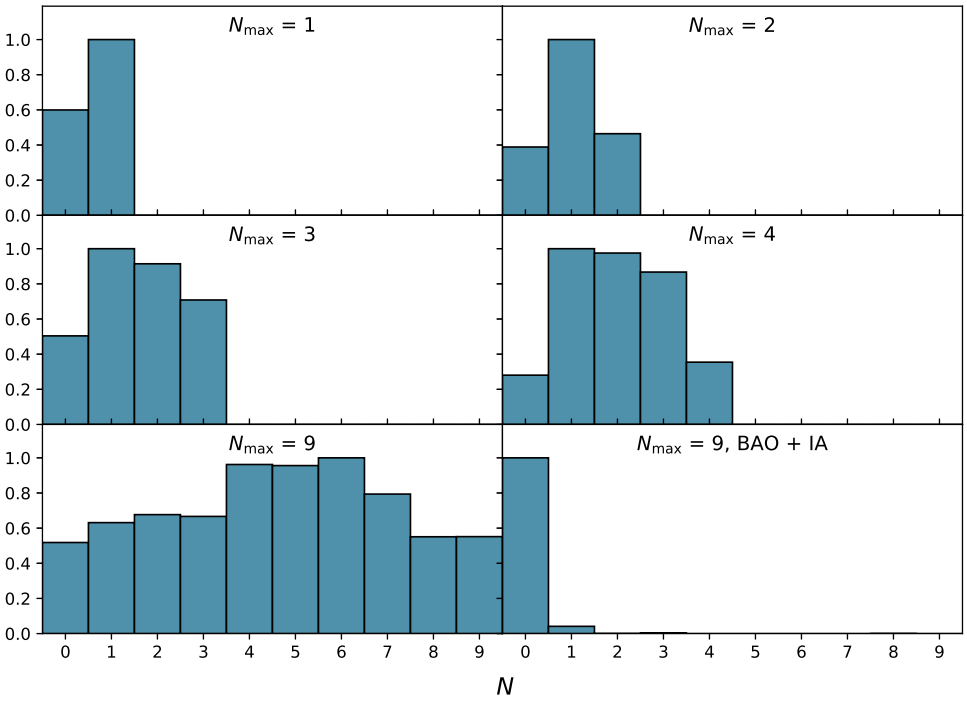


Figure 7: Histograms of posterior samples of the number of nodes of an `AdaptiveLinf` used to reconstruct the dark energy equation of state parameter. $N = 0$ corresponds to $w = -1$, the cosmological constant. The first five cases use different maximum N, and do not favour the cosmological constant. Adding in Baryonic Acoustic Oscillations and IA supernovae data recovers the cosmological constant in the sixth plot.

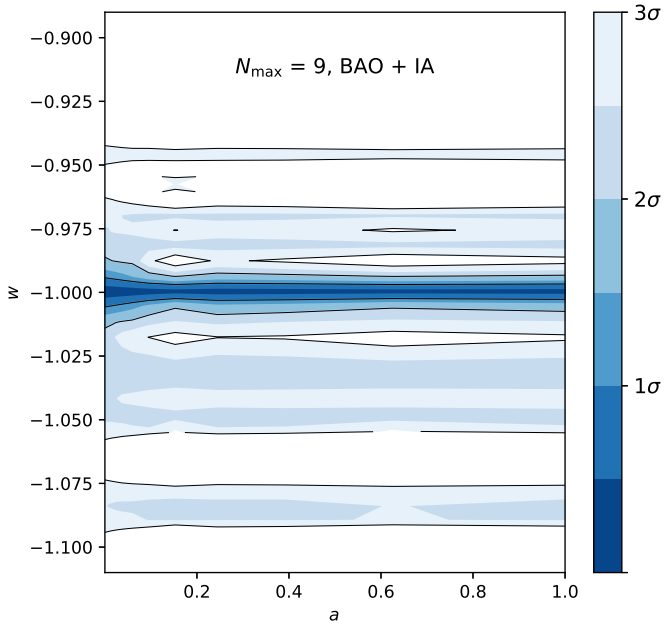


Figure 8: Posterior plots using an `AdaptiveLinf` as a model for the dark energy equation of state parameter, using the Baryonic Acoustic Oscillations and IA supernovae data in addition to the Planck likelihoods. This recovers the cosmological constant $w = -1$.

7 PolyChord Revisited: Clustering

To produce results more quickly I used 100 `PolyChord` live points, intending to increase this to 25 × number of dimensions = $25N_{\text{dims}} \approx 800$ for the final run to put in this report, as advised [**PolyChord˙1**, **PolyChord˙2**, **Sonke**]. However, for the three-node case this causes a problem. $\mathcal{P}_{\mathcal{R}}(k)$ is most tightly constrained to a linear slope for $k = 10^{-2.5} - 10^{-0.8}$, and much less so outside of this. Two high-likelihood modes are expected, one with a node to the left of this range, and one to the right. The posterior plot of the *linf* should therefore consist of a narrow centre section, with the less constrained parts fanning out at either side from each of the modes. This is what is seen with 100 live points. However, when the resolution is turned up to $25N\text{dims} = 725$, only the left hand mode is present. We believe this to be because the greater density of live points means the parameter space gap between the two modes is bridged, so `PolyChord` fails to identify them as two separate clusters, which is necessary for it to function properly. The posterior samples for the internal node for both resolutions and their respective functional posteriors are shown in figure **??**

Since the original paper [**Handley2019**], improvements have been made to `CAMB`, in particular its ability to account for the non–linear lensing effects due to matter between the surface of last scattering and Earth. However, there is an issue with the `Cobaya` documentation with respect to how to turn off these non–linear effects, so I have been unable to test whether removing them recovers the right–hand mode. This is apparent as turning this off should increase `CAMB`'s speed and therefore its oversampling factor, which has not happened.

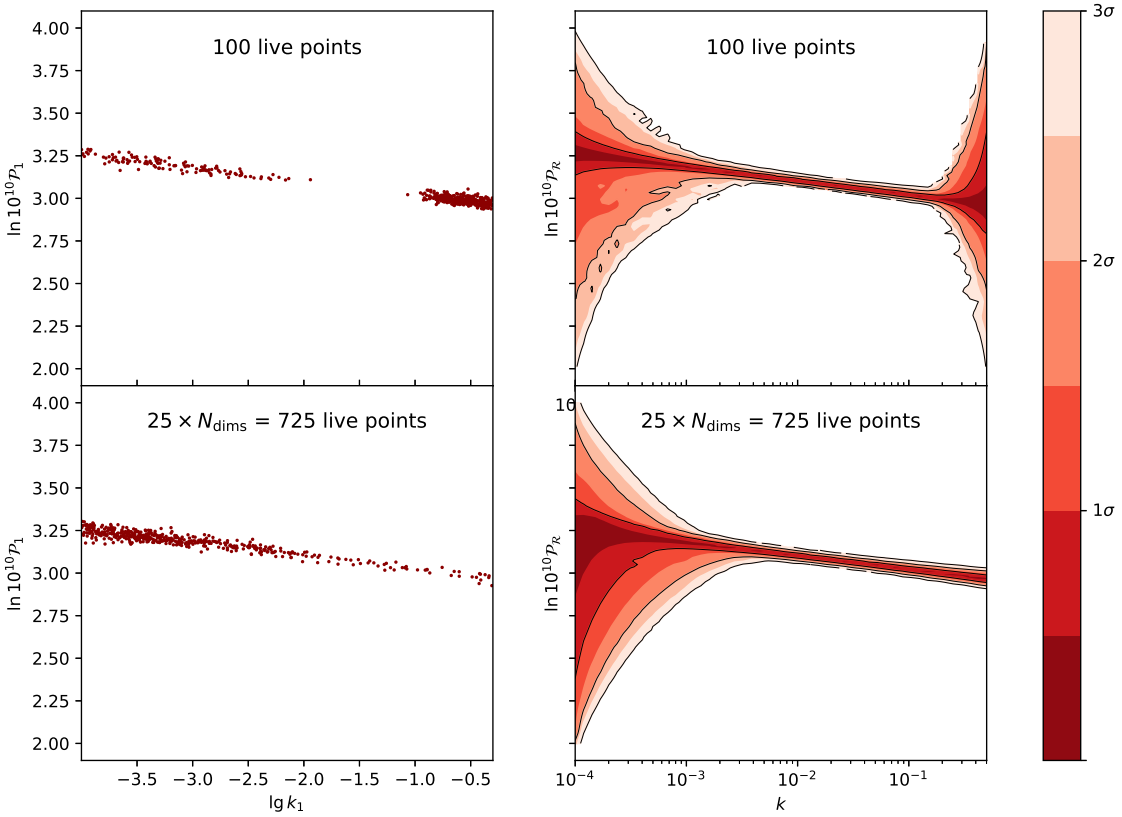


Figure 9: Left: posterior samples of the internal node of three for the *linf* primordial matter power spectrum. With 100 live points, there are two distinct modes, which `PolyChord` has identified as two separate clusters. Increasing the resolution to 25 × N_{dims} causes the mode with the internal node at high k to be lost. Right: posterior of the *linf* function for the primordial matter power spectrum. The loss of the mode with the internal node at high k means the distribution doesn't fan out on the right.

7.1 Custom Clustering

To investigate the clustering issue, Dr. Handley and I have modified `PolyChord` to use a user-defined clustering algorithm. By default, `PolyChord` uses a recursive version of the K–nearest–neighbours (KNN) algorithm:

```
function K_nearest_neighbours(distance2_matrix[n_points, n_points]):
  K = min(10, nlive)
  calculate the K nearest neighbours for each point
  do n = 2 to K:
    if two points in either of each other's n nearest neighbours:
      they are in the same cluster
    if cluster_list unchanged since n-1: return cluster_list
    if n == K:
      K = min(2*K, nlive)
      recalculate K nearest neighbours for each point

  if number of clusters >= 2: search recursively
  return cluster_list
```

Browsing though alternatives, K-means seems the most recommended and robust alternative [**MacKay2003**], and is included in the SciKit Learn Python package [**scikit-learn**]. The K in both names is misleading as their meaning is unrelated, in KNN it refers to the number of neighbours to consider, in K-means K is number of clusters to consider.

```
function K_means(position_matrix[n_points, d_dimensions], K):
  # initialisation
  responsibilities = zeros[K, n_points]
  means = K random positions

  # assignment
  while means are changed since previous pass:
    distance2 = (x-means)^2
    for j = 1 to n:
       i_closest = 1

       for i = 2 to k:
         if distance2[i, j] < distance2[i, j]:
           responsibility[i_closest, j] = 0
           responsibility[i, j] = 1
           i_closest = 1
         elif d[i, j] == d[i_closest, j]:
           assign to whichever has lower responsibility

    for i = 1 to K:
      R = sum(responsitlities[i, :])
      if R > 0:
        means[i, :] = dot(responsibilities[i, n], x[n]) / R
```

K-means is not guaranteed to find the optimal set of clusters; this can be helped by carefully choosing the intial guess for the means. K-means++ uses the following initialisation steps [**kmeans++**]:

1. Choose one of the data points as the first mean.
2. Calculate the squared distance of each of the remaining points to the nearest point that has already been chosen.

3. Choose another point, weighted inversely to these squared distances.

4. Repeat steps 2 and 3 until K inital means have been chosen.

While this algorithm itself takes time, it reduces the error of the result, and speeds up the convergence relative to initialising the means randomly [**kmeans++**].

The other issue with K-means is that it needs to know the number of clusters to look for in advance. X-means is a variant of K-means which uses subdivision to refine cluster assignments, to find the numbef of clusters which optimizes the Bayesian Information Criteria [**bic**]. This is the clustering algorithm used by `MultiNest` [**Feroz˙2008**].

`PolyChord` is advertised encouraging the user to experiment with different clustering algorithms [**PolyChord˙1**, **PolyChord˙2**], however this is not user-friendly. To familiarise myself with `PolyChord`'s clustering code, I added my own K-means algorithm in FORTRAN90, while in parallel Dr. Handley created an interface to allow a user-defined clustering algorithm to be passed from Python. KNN only requires only the squares of the distances between live points, but a general clustering algorithm requires the absolute position matrix of the points (up to a constant offset or rotation), including K-Means. I therefore updated the interface to require the position matrix. This introduced complexity: `PolyChord` uses C++ to interface between FORTRAN and Python, C++ and Python use row-major ordering arrays, while FORTRAN is column-major. This was not a problem with the squared distance matrix as it is square and symmetric, so I had to take care to ensure the dimensions of the non-square position matrix appeared in the correct order. FORTRAN is also 1-based, and consequently the clusters need to be numbered from 1. Python clustering algorithms such as those in the SciKit Learn package number the clusters from 0, so the simplest workaround is to use a wrapper which adds 1 to the result! Since custom clustering will become a part of `PolyChord`, I will make this correction part of the interface.

This work is in progress, I have just finished checking that the interface is working correctly by passing `PolyChord` a Python copy of its default KNN algorithm, and ensuring the clustering behaviour is unchanged.

7.2 PolyChord vs MultiNest

`PolyChord` comes with `run_pypolychord.py`. This is an example used to check that the install has worked properly, which has a 4-D Gaussian likelihood, centred on the origin with uniform prior. To test clustering, I modified this to vary the number of dimensions D, and added an identical peak at $(0.5)^D$, so `PolyChord` should identify two clusters. The resulting posterior should then also be two peaks of equal height, the same as the likelihood. However, I noticed that the peaks have stochastic skew to their posterior mass.

To investigate the cause of this, I tried the same example with `MultiNest` [**MultiNest**], to see whether it exhibited the same behaviour. `MultiNest` consistently produces a posterior with two visually identical peaks, so this issue is not fundamental to nested sampling, but some difference between `PolyChord` and `MultiNest`. Nine repeats are shown in figure **??**.

`MultiNest` uses the live points to construct a set of intersecting ellipsoids which together aim to enclose the iso–likelihood contour, and then performs rejection sampling within these ellipsoids. While this is effective for a moderate number of parameters, rejection sampling scales exponentially with dimensionality which eventually dominates. As mentioned in the previous section, `MultiNest` also uses X-means clustering rather than `PolyChord`'s KNN clustering.

This issue is currently unsolved. At our last discussion, Dr. Handley suspected that it is an issue where, if one cluster has more live points than the other, new live points are more likely to spawn there and cause that cluster to dominate like a Pólya Urn — "the rich get richer" [**PolyaUrn**].

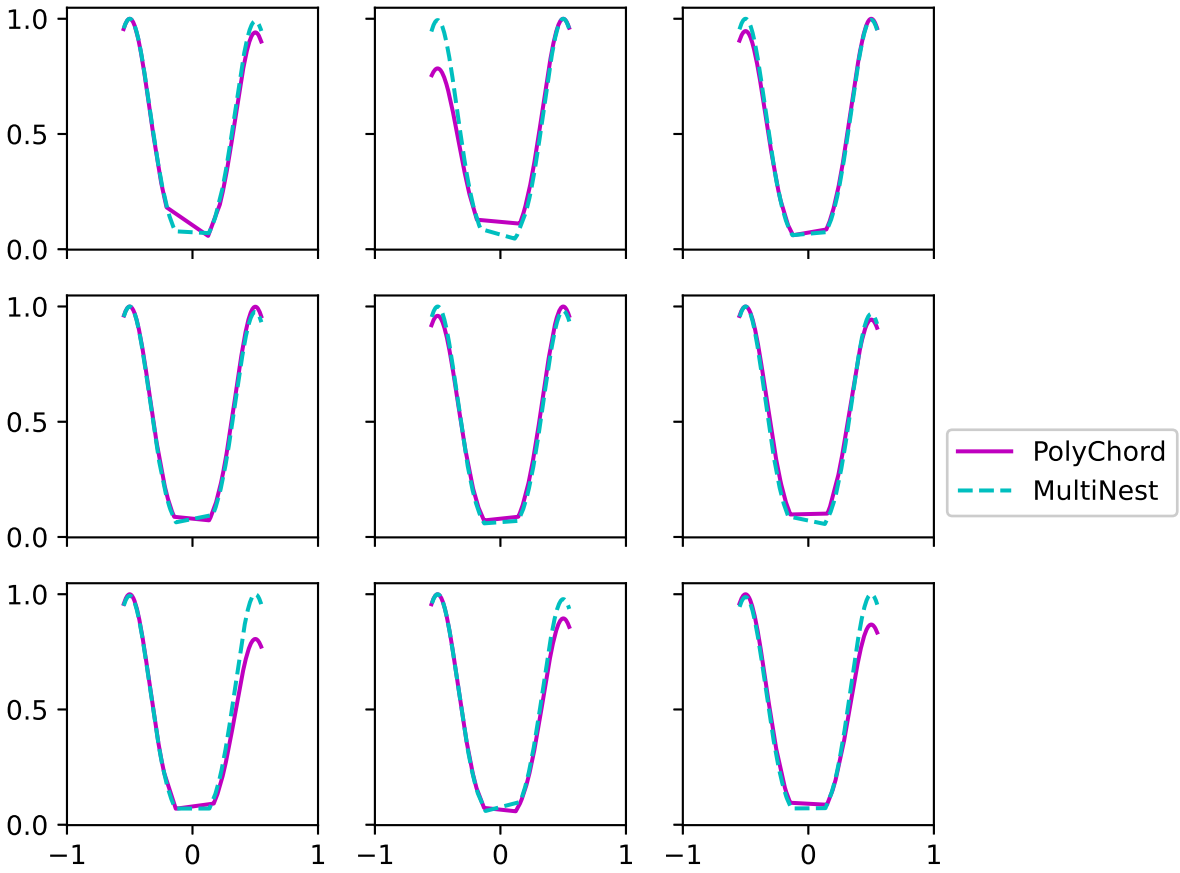


Figure 10: Posterior kernel density estimate plots for a double-Gaussian likelihood in 2D, showing the first parameter marginalised over the second, for both `PolyChord` and `MultiNest`. This is repeated nine times. `MultiNest` consistently produces peaks of equal height, while `PolyChord` fails to do so seemingly at random.

8 CDT courses

As my funding is from the Centre for Doctoral Training, I attended several courses. In Michaelmas I attended Part III maths' *Cosmology* and DAMTP's *Introduction to Research Computing*, and in Lent Part III maths' *Advanced Cosmology* and *Astrostatistics*. In Michaelmas I also intended to attend *Machine Learning for Fundamental Physics* in a partnership with the University of Munich, which was recorded in 2020. However, due to breaking my ribs in late October, I decided to leave this course for later to lighten my workload. I will attend this course over the summer once I have finished this report. I also plan to attend the second half of *Cosmology* again for the same reason.

Introduction to Research Computing has come in the most useful. As I began the year unfamiliar with Linux, the first few lectures was a good crash-course to translate the commands I knew from Windows PowerShell. As I was already very familiar with Python, the section on multiprocessing was the most useful element of the rest of the course. *Astrostatistics* was helpful in reminding me that there are other sampling techniques besides `PolyChord`! It also covered the basics of Gaussian process modelling, which I discussed briefly in section **??**.

9 Computing

A significant part of my learning this year has been gaining relevant computational skills. At the start of the year, I was comfortable using Windows, coding with my favourite editor Visual Studio Code. I was supposed to be provided with a MacBook, however this didn't arrive until early February. `PolyChord` is compatible only with Unix-based operating systems, so I wouldn't be able to learn how to use it on my Windows laptop. Therefore, I started using CSD3 (Cambridge Service for Data-Driven Discovery), a supercomputer accessed via SSH (secure shell). This overcame the compatibility issue with `PolyChord`, and was good practice for using the service. Initially I worked mainly using the login nodes, but these are intended to serve as an interface for submitting jobs to the compute nodes, which I would need to use to perform cosmological runs.

I have continued to use Visual Studio Code as my primary editor, as it has a `Remote - SSH` extension which allows near-seamless connection to a remote workspace. However, I found it limiting being unfamiliar with editing files directly in the terminal, as it is not convenient to have to create a VS Code workspace every time I want to edit a file. Therefore, I took some time to work through *10-Minute Vim* to learn to use the editor [**vim**]. I am starting to prefer using Vim keybindings, I use them in VS Code to edit text files, use `vimdiff` to compare git branches, and am using them in Overleaf right now!

I have used a range different programming languages over the years, but since the second year of my undergraduate degree I have almost exclusively used Python. The code I have worked with this year has been a mixture of pure Python and Python interfaces atop FORTRAN or C++ implementation. In December, I took part in the Advent of Code using C++, which I chose over FORTRAN as I am familiar with curly-brace syntax from Java. I think it will be a useful exercise to do it again this year, but using FORTRAN [**advent**].

I previously used git and github during my undergraduate degree, but only as the sole contributor to a project with two or three branches. This year I have learned about creating pull requests, and have become better at working on multiple branches and passing changes between them in both directions. I still feel I could be much better at collaborative development, but I am much more confident at changing existing code so I should get to practice collaboration much more.

CSD3 has three different types of Intel node. From oldest to newest: Skylake, Cascade Lake, and Ice Lake. By default, a login session connects to a Skylake node, which used to submit jobs to the Cascade Lake compute nodes. This worked fine initially when I was only using `PolyChord`, but when I started probing the dark energy equation of state, `CAMB` suffered from segmentation-faults. I realised that I needed to load the Cascade Lake modules onto the login nodes to compile `CAMB` correctly for job submission. I realised a good way to remember the different installation steps would be to write bash scripts, as these would not only serve as a record but do it automatically when I needed to create a fresh Python virtual environment.

As a SL3 tier user, I can submit jobs on up to 8 nodes. Cascade Lake nodes have 56 cores, versus the 76 on Ice Lake. There are also Ice Lake login nodes, so there would be no compatibility issues between login and compute sessions. I intended to move over to Ice Lake, but I could not install the Planck likelihoods. I will try again once I have submitted this report.

10 Conclusions

Over the course of this year, I have become familiar with `PolyChord`, `Cobaya` and `CAMB`, both in using and modifying them. I have learned to use the supercomputer CSD3, both in terms of using Linux over SSH, and submitting large jobs to the compute nodes.

I have found `PolyChord` to be a suitable way of fitting a *linf*, a linear spline between either both a fixed number of nodes, and also allowing the number of nodes to vary as a parameter itself. However, when the number of live points (effective resolution) is increased to its recommended setting, it is unreliable in correctly identifying clusters. This became apparent when using a *linf* to reconstruct the primordial matter power spectrum. I am working on adding a custom clustering argument to `PolyChord`, where the user can supply their own clustering algorithm.

The above was all intended as practice for investigating the dark energy equation of state parameter w. I have modified `Cobaya` to include an `external_wa` option, analogous to `external_primordial_pk`, where the user supplies their own `cobaya.Theory` subclass to provide a table of k, $\mathcal{P}_\mathcal{R}(k)$ or a, $w(a)$ values. This change is almost complete, but understanding the clustering issue with `PolyChord` is vital before any definitive conclusions can be made about dark energy.

11 Future Plans

11.1 Short term (less than three months)

- Watch the *Machine Learning for Fundamental Physics* course and complete the exercises.
- Switch to the Ice Lake partition of CSD3. The current issue is getting `Cobaya` to install the Planck Likelihoods. The greater core count will allow runs to complete more quickly.
- Work out how to turn off non–linear lensing in `CAMB` via `Cobaya`, and see whether this recovers the right–hand mode in the primordial matter power spectrum.
- Understand why `PolyChord` does not recover $N = 4$ for the "line" toy example.
- Finish testing `PolyChord` with custom clustering, and find a suitable replacement clustering algorithm.
- Work with Dr. Handley to fix the issue where `PolyChord` is unable to produce a balanced posterior for a symmetric double Gaussian likelihood. We plan next to experiment with ideas from dynamic nested sampling.
- Understand why anesthetic is creating samples with NaN weights and likelihoods when using `external_wa`.
- Modify anesthetic to give a suitable warning when it encounters NaNs.
- Modify my `external_wa` option in `Cobaya` so that it understands automatically that the parameters for the dark energy Theory are to be sampled with the rest of the cosmological parameters.
- Work through the Advent of Code, but this time in FORTRAN [**advent**].

11.2 Medium to long term (more than three months)

11.2.1 Tasks conditional on fixing and understanding `PolyChord`

Once the different behaviour of `PolyChord` and `MultiNest` with multi-modal likelihoods has been understood, I will write this into an article. The custom clustering version of `PolyChord` will be merged into the master branch, and I will try and get the necessary changes to `Cobaya` also added to match, so that others may benefit from this analysis.

When this is finished, and a suitable clustering algorithm has been chosen, I will finish reanalysing the primordial matter power spectrum. This may form a separate paper to the `PolyChord`–specific changes, as the improvements to non–linear lensing in `CAMB` which are responsible for the disappearance of the mode then it must also be demonstrated how to fix this.

After I have finished studying the dark energy equation of state, it is natural to probe those of dark matter, and the effective dark radiation proposed by W.E.V. Barker discussed in section **??**. The changes to `Cobaya` should be straightforward, as they will be very similar to the changes made to provide an external dark energy model. There are three paths that could be taken with Boltzmann codes:

1. Modify `CAMB`. Until the Advent of Code, making changes to `PolyChord` is my only experience with FORTRAN, so I will find it most difficult to understand the existing code and where to make additions. However, this would be valuable experience, and the corresponding changes to `Cobaya` will be most straightforward.
2. Switch to and modify `CLASS`. This is written in C++, which I have more experience in, and I am also used to the curly-brace syntax from Java in my first undergraduate year. However, I may find `CLASS` suffers from idiosyncrasies similar to non–linear lensing in `CAMB`. I am also not familiar with the `CLASS` interface in `Cobaya`, which I would need to learn to modify.

3. Switch to `PyCosmo` [**pycosmo**]. This is an alternative Boltzmann code which generates efficient C/C++ code from `SymPy` (Symbolic Python). This represents a much faster way of adding the new components to the Boltzmann code. However, there is no interface for `PyCosmo` in `Cobaya`, which I would need to add myself. Its relative youth also means it lacks the decade(s) of testing of `CAMB` or `CLASS`.

I will revisit this decision once the multi–modal issues with `PolyChord` have been resolved, and the priomoridal matter power spectrum and dark energy investigations are running.

11.2.2 `anesthetic.2` and Pandas

Dr. Handley and Dr. Hergt (a past member of the group) plan to `anesthetic.2` in the next few months. As well as performance improvements, they have updated the interface to more closely match that of `Pandas`, which `anesthetic` extends. I have found that some of the recent changes have had the side–effect of preventing some formatting parameters, such as the colour of a plot, to not be applied properly. I have corrected these issues by more closely following the structure of the overriden `Pandas` methods. However, this has led to me finding a bug in `Pandas` itself. Nested samples are weighted, which `Pandas` should be able to plot histograms and kernel density estimates out of the box. However, as discussed in section **??**, nested samples can contain NaN values, `Pandas` will drop when plotting. However, it cannot handle both weights and NaNs at the same time. I am working on a pull request, which will take several weeks to get merged first into the master branch, then months into the release version. Until this happens, I have provided a workaround by overriding the same methods in `anesthetic`.

By adding a new component with a dynamic equation of state to my analysis, I will be able to assess whether there is evidence to support a new component, A *linf* can model both the periods before and after the transitions, and provide insight into the nature of the transition itself. As matter-radiation equality occurs at a redshift of around 3400 [**planck'6**], a *linf* in $w_{\text{dark radiation}}(\log a)$ would be more appropriate.

As a part of this work, I would similarly like to look further back with the dark energy equation of state. Once I have successfully reproduced S. Hee's results [**Sonke**], I will re-parameterise w as $w(\log a)$.

11.3 Long term

There are other functions ripe for similar non-parametric reconstructions, for example the reionisation history $x_{\text{e}}(z)$, late–time matter power spectrum $P(k)$, inflation potential $V(\phi)$ CMB power spectra C_ℓ and lensing spectra $C_l^{\phi\phi}$. A more ambitious reconstruction will be the quantum mechanical initial conditions for inflation, A_k and B_k. These are complex functions of k, subject to the constraints look at advanced cosmology notes! so there are two degrees of freedom.

The final aim will be to perform simultaneous reconstructions, for example $x_{\text{e}}(z)$ and $\mathcal{P}_{\mathcal{R}}(k)$, (A_k, B_k) and $V(\phi)$, and $\mathcal{P}(k)$ and $w(a)$. Allowing simultaneous changes to cosmological modelling may provide clues as to the cause of the underlying physical causes of parameter tensions.

12 Acknowledgements

Thank you to Cambridge Machines Access Management and Centre for Doctoral Training in Data Intensive Science for awarding the funding for my PhD.