Having confidence in your confidence: parameter estimation post-processing

LIGO Document P1400054-v2

Trevor Sidery,[1,*] Will Farr,[1] Jon Gair,[2] and Ilya Mandel[1]

[1]*School of Physics and Astronomy, University of Birmingham, Edgbaston, Birmingham B15 2TT, UK*
[2]*Institute of Astronomy, University of Cambridge, Cambridge, UK*

(Dated: July 23, 2014)

Stochastic samplers are often used to perform Bayesian inference on data sets. It is often assumed that once the samples are produced then this is the end of the data processing and the treatment of the samples will not greatly affect the results. In this paper we re-emphasise some of the biases that can enter the extracted information, due to the processing of these samples. We suggest a straightforward method to solve these issues called the 2-stage kD-tree. This method has already been used as a necessary part of testing algorithms that are used to extract gravitational wave signals from ground-based interferometer data.

INTRODUCTION

There are many data analysis problems that are now being tackled by stochastic Bayesian based algorithms. These problems are often characterised by a need to recreate a complex probability density function (PDF) that describes the most probable values of a models input parameters. One set of these algorithms produce a collection of coordinates in the model parameter space, whose distribution maps the PDF. These 'samples' are either drawn directly from the PDF as in the case of Markov-chain Monte Carlo with a Metropolis-Hastings acceptance condition, or a re-weighting of previously collected samples, e.g. nested sampling.

Once these runs have completed we expect the samples to represent the sought after PDF and we have the task of reconstructing the PDF from this finite number of samples. Our ability to recreate the true PDF from the set of points will depend on the number of samples output. There is a large amount of literature on various ways to do this, be this histograms, kernel density estimators or kD-trees, all of which have their issues when implemented [1] . The preferred method will depend on the use of the PDF and the number of samples output by the code.

The problem we have been considering is that of data from gravitational wave detectors and using bayesian methods to estimate parameters from two inspiralling black holes. For this paper the important properties of the PDF of the signal model parameter values are that it is a multi-modal, high (9-15) dimensional function. We are often interested in just a couple of these dimensions, for example, the location of the source in the sky and we can use this to our advantage.

As part of testing our sampling algorithms we noted that the recovered PDFs could be biased by the handling of their output. For parameter estimation there are a couple of questions we need to answer with our PDFs. What are the regions in parameter space that we are $x\%$ sure that the source can be found. While this in itself is a simple question, we usually require the minimum region where this is true. Coupling this requirement with the fact that our PDF is represented by a finite number of samples can very quickly introduce bias.

This can be important for a few different reasons. Firstly, we want to be able to publish accurate scientific results that we can trust. Secondly, we would like to be able to have ways in which we can test our sampling codes on large numbers of simulations and this depends on having reliable post-processing (see [2]). We may also need to use the output of our code as the input to new runs, with a new data set, propagating any errors.

These are all issues that have been found before, and even solved, but are regularly ignored. We present an overview of the problem, some examples of how easily this bias can enter an analysis, as well as a new and particularly straightforward way of solving it.

OPTIMALITY VS CONSISTENCY

Given data output from a suitable sampler we must recreate the PDF that the samples are drawn from; we call this *post-processing*. We describe the PDF via credible levels (CL): the integrated probability over a given region of the parameter space. In particular we consider the smallest region, or minimum credible region ($\mathrm{CR}_{\min}$), for a given CL; the smallest region in the parameter space for which there is probability of CL that the true parameter values are contained in this region. More formally, for a given CL, any credible region (CR) must satisfy

$$\mathrm{CL} = \int_{\mathrm{CR}} p(\vec{\Omega}|d)\,d\vec{\Omega}\,. \tag{1}$$

We can then find the smallest region such that this still holds, which we call $\mathrm{CR}_{\min}$. By considering the full range of probabilities we can map out the PDF with a set of contours that bound each $\mathrm{CR}_{\min}$. When producing $\mathrm{CR}_{\min}$ we strive for 2 qualities, consistency and optimality. The former is a necessary condition and is the property that produced $\mathrm{CR}_{\min}$ satisfy Eqn. 1. Consistency is checked over multiple runs by confirming that the desired fraction of injections falls within the credible region corresponding to the stated credible level; see [2] and below for a discussion of p-p plots. Optimality relates to our ability to minimise this region, which is important for increasing the usefulness of our results. When setting up algorithms we must be cautious that our optimising methods do not break consistency; consistency is necessary, while optimality is desired. This is particularly true if we are testing the sampling algorithm as we must have confidence that the post-processing is never at fault if the resultant PDFs are not consistent.

KD-TREES AND THEIR BIAS

The simplest approach to recovering a PDF from a set of samples is to impose a grid on the parameter space as a set of bins. We count the number of samples that fall in each bin, the result of which will be proportional to the probability density. The CR of a given CL is the region that covers a set of bins that contain a CL fraction of the samples. To optimise the size of this region we apply a *greedy* algorithm that starts from the bin with the most samples and, keeping a running total of samples, counts from the most populated (highest density) bin towards the least. Once we have counted the necessary fraction of samples we have found the $\mathrm{CR}_{\mathrm{min}}$.

This simple method has a few important drawbacks. The most obvious being that for any non-trivial PDF there is no single resolution that behaves well over the whole space. Bins that are too large will hide any detail, while small bins will often be empty and so are unorderable. The former significantly hampers our ability to optimize, while the latter leads to inconsistency. Pushing these 2 scenarios to their limit, the lowest resolution (1 bin) will only return the 100% $\mathrm{CR}_{\mathrm{min}}$ which covers the entire parameter space, while as the number of bins goes to infinity the volume of all $\mathrm{CR}_{\mathrm{min}}$ tends to 0.

Instead we turn to a kD-tree method of binning [3], shown pictorially in figure . This method is similar to the gridded binning idea except that bin sizes are adaptive; chosen so that they all contain a similar number of samples. This means boxes will be large in regions where the samples are sparse but small where the samples are densely packed. This is achieved by taking the parameter space, choosing a particular parameter, ordering the samples by this parameter and dividing them into 2 bins using the median. The next parameter is then chosen and the samples from each bin are again divided according to the samples median of the new parameter. We continue this process, looping through all the parameters, until there are a required number of samples left in each bin. To find credible regions the same greedy algorithm that was applied to the gridded space is used, with the ranking of bins by sample density.

Using kD-trees can greatly improve on our ability to optimize, while never having empty, unorderable, boxes. We do not claim that kD-trees are the perfect method for optimising the $\mathrm{CR}_{\mathrm{min}}$, but they are a straightforward method of doing so and produce resultant PDFs in an easy to use format. While we have addressed the issue of optimality, the necessary property of consistency has not been checked.

We now show that the use of a greedy algorithm to assign credible levels introduces a bias that breaks the requirement of consistency, and so will affect both the kd-trees and the gridded binning. Consider figure 2; if we have a number of samples that are taken from a uniform distribution (24 in this case) and apply the usual kd-tree algorithm once, we will have 1 bin that is larger and 1 smaller due to Poissonian noise. While some variance in box size is expected, as we have a finite number of samples, a bias is introduced in which the smaller box is always chosen as the 50% credible level. A new sample picked from the same distribution would not have a 50% chance of falling in the designated 50% $\mathrm{CR}_{\mathrm{min}}$. Consistency is not broken due to the CR_{min} being too small in this one case, but that a greedy algorithm

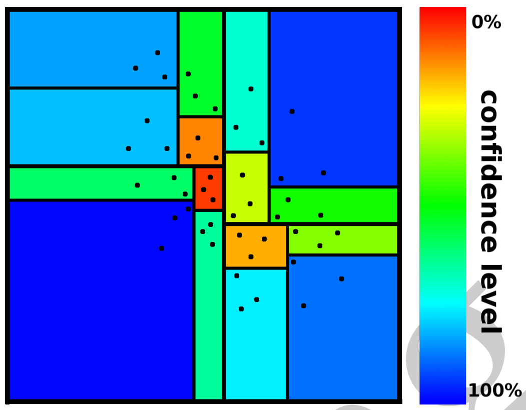

FIG. 1. To construct a kD-tree, the median of the samples is found and used to divide the parameter space. Repeating this process in alternating parameters, the parameter space is further subdivided. Confidence levels are then assigned 'greedily' by counting samples in order of bin density.

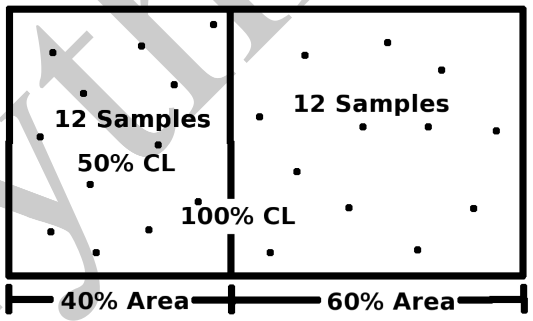

FIG. 2. In the case of samples taken from a uniform distribution there will be some statistical fluctuation on where the median is located. A bias is introduced that results in the recovered 50% CR_{min} covering less than half of the parameter space.

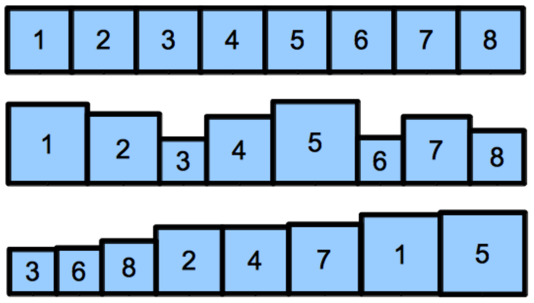

FIG. 3. The largest bias is introduced for a uniform distribution. The boxes are numbered to track each bin as the kD-tree algorithm is applied. Row 1 shows the bin sizes that would be recovered in an ideal case. Row 2 is an example of the actual returned bin sizes after applying the kD-tree algorithm. Row 3 shows the result of applying a greedy algorithm to order the bins, where we can see that the first x% of samples do not correspond to an area of x% of the total area.

will systematically choose the bin that is too small.

Impact of Bias

By considering the uniform distribution case we can show how the bias that we considered for 2 bins scales to many bins. In figure 3 we show how the bins are created and ordered. If the binning corresponded perfectly to the underlying uniform distribution then the bin sizes would all be equal (row 1). In practice we have finite samples and as was shown in the 2 bin case there will be a range of bin sizes produced by the kD-tree algorithm due to Poissonian noise (row 2). Finally we order these bins (row 3) and count the areas starting from the smallest. At the point we have counted the samples from the smallest $x\%$ boxes, we have clearly not counted $x\%$ of the area. Note in the figure that the numerical order of the boxes at the end is entirely random and depends only on the stochastic fluctuations.

For a non-uniform distribution there is an inherent ordering of the bins. We can see in figure 4 how this ordering counteracts the stochastic bias. Bins which are shrunk most by stochastic variations no longer necessarily end up at the head of the ordering and similarly enlarged bins wont necessarily end up in the tail. There is still a bias towards having confidence levels that are too small, but its not so pronounced.

On one hand this means that the severity of the bias depends on the background distribution and so we cannot apply a universal correction. On the other hand we can test the uniform distribution case with varying numbers of total samples, and numbers of samples in each bin, to quantify the bias in the worst case.

Using this we may state a worst case correction in which we relabel the $\mathrm{CR}_{\mathrm{min}}$. This would still not be consistent, though credible regions would now be too conservative in

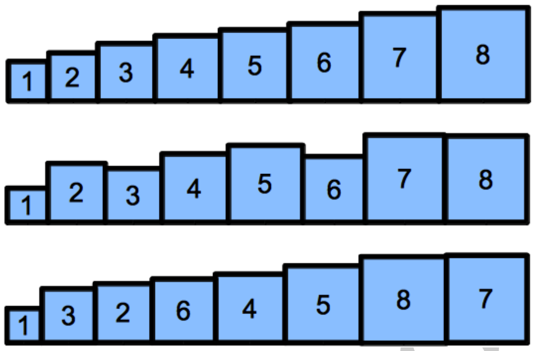

FIG. 4. The non-uniform case has some inherent ordering which counters the stochastic bias. The boxes are numbered to track each bin as the kD-tree algorithm is applied. Row 1 shows the bin sizes that would be recovered in an ideal case. Row 2 is an example of the actual returned bin sizes after applying the kD-tree algorithm. Row 3 shows the result of applying a greedy algorithm to order the bins. The bias towards small credible regions still shows but is less pronounced than the uniform distribution case.

contrast to the ‘uncorrected’ results.

Test distributions

To show some examples of how this bias can be seen in practice we created a large number of test cases (4000). Each test case consists of a large number (2^{15}) of samples drawn from a given distribution. We consider the uniform and gaussian distributions as examples of poorly and accurately found signals. To see the bias in practice we took the test cases and applied the kD-tree algorithm to them. Taking a new point from the same distribution, we treat this test sample like a signal and ask which credible region it falls in. As stated before we know that the test sample should fall within the $x\%$ credible region $x\%$ of the time. We test to see if this fraction vs CL relation holds. By using a large number of test cases and with a test sample that is drawn from the same distribution as the bulk of the samples, any observable deviation from the expected relation comes from the systematic bias we have just discussed. Figures 5 and 6 show that there is significant bias in both cases and the uniform curve does indeed deviate more than the gaussian one, as expected from our previous arguments. The sagging from the expected diagonal line on these p-p plots indicates a bias towards small CR_{min}. We also show the dependence on samples left in each bin; fewer samples in each bin exacerbates the effect of Poissonian noise and increases the bias towards small $\mathrm{CR}_{\mathrm{min}}$.

2-STAGE PROCESS

Our suggestion for a consistent binning method is to have a 2-stage process that separates the ordering of the

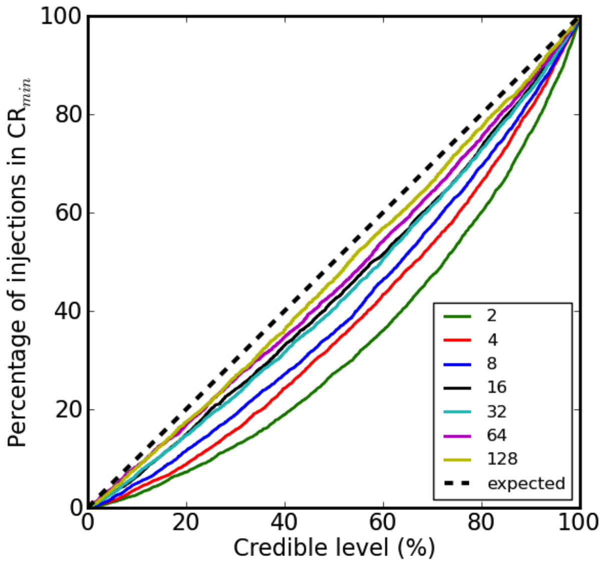

FIG. 5. For a range of CLs, the fraction of runs in which the test sample fell within the CR_{min} computed via the standard kD-tree algorithm is shown. This was done using 4000 runs, each with 2^{15} samples drawn from a 2d *uniform* distribution. Each line denotes the number of samples per kD-tree bin.

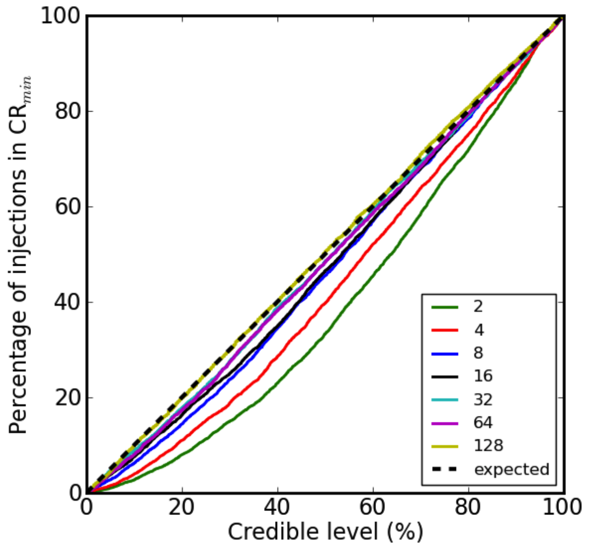

FIG. 6. For a range of CLs, the fraction of runs in which the test sample fell within the CR_{min} computed via the standard kD-tree algorithm is shown. This was done using 4000 runs, each with 2^{15} samples drawn from a 2d *gaussian* distribution. Each line denotes the number of samples per kD-tree bin.

bins from the determination of credible levels. Randomising the list of samples and splitting them into 2 equal sized groups, we use the first to construct the bins using the kD-tree algorithm. We then throw away the samples, but keep the ordering of the bins stored which will be used later to construct the confidence levels. We then take the remaining

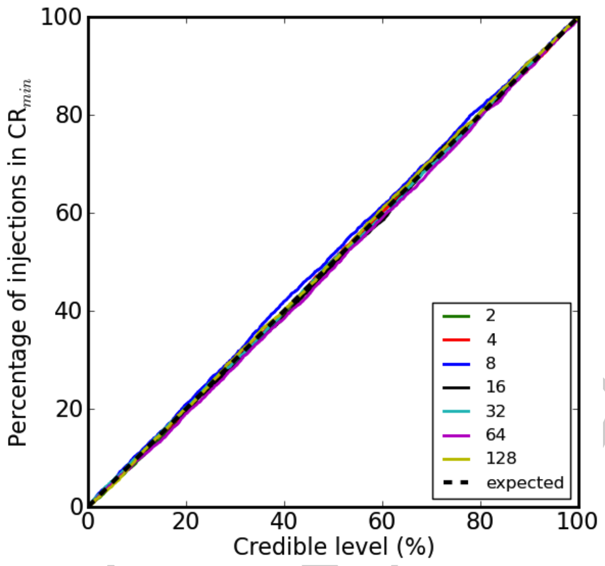

FIG. 7. For a range of CLs, the fraction of runs in which the test sample fell within the CR_{min} computed via the 2-step kD-tree algorithm is shown. This was done using 4000 runs, each with 2^{15} samples drawn from a 2d *gaussian* distribution. Each line denotes the number of samples per kD-tree bin. It can be clearly seen that the results follow the expected diagonal.

samples and 'fill up' the empty tree. Finally, to construct the confidence levels we follow the previous method where the bins are ordered and we count samples until we reach the appropriate fraction. The difference from before is that the order of the bin list comes from the first stage while the samples are counted from the second. To show that the method works we have recreated the previous test with a gaussian distribution but using the 2-stage process to produce CR_{min} and it can be seen in figure 7 that the method is consistent for any number of samples per bin.

Why it works

In the first stage we have used the samples to do 2 jobs. We have created bins that change in size according to the probability density and so have still fixed the resolution issue that KD trees were originally chosen for. There is also an appropriate choice of bin ordering, that reflects the general shape of the PDF. Then, because the second set of samples, that we use to fill the bins, are independent of the first step then the systematic bias is gone. Any bins that are too small will have an expected sample count less than the average samples per bin, while bins that were created slightly too large will have a larger expectation value. Consider again figure 2 for a uniform distribution. The first stage will again create 1 bin that is too large and another that is too small. Again we find that the smaller bin will be counted first, but this time the samples are thrown away before we count. The second set of samples is put in these bins, with

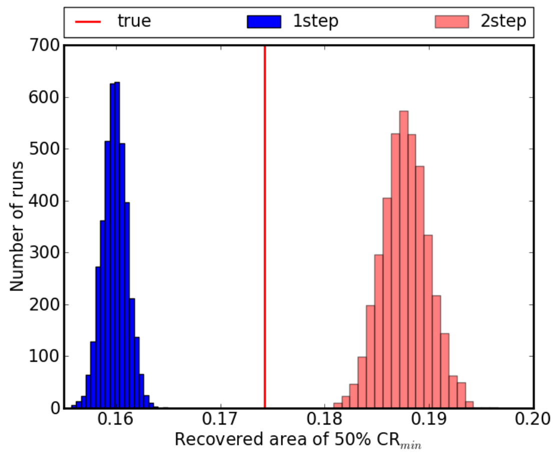

FIG. 8. Distribution of the 50% CR$_{min}$ areas recovered by the 1-step and 2-step binning process from 4000 runs. For both algorithms the number of samples per bin is fixed at 8. The 2^{15} samples produced for each run are taken from a 2-dimensional *gaussian* distribution with a standard deviation of 0.2

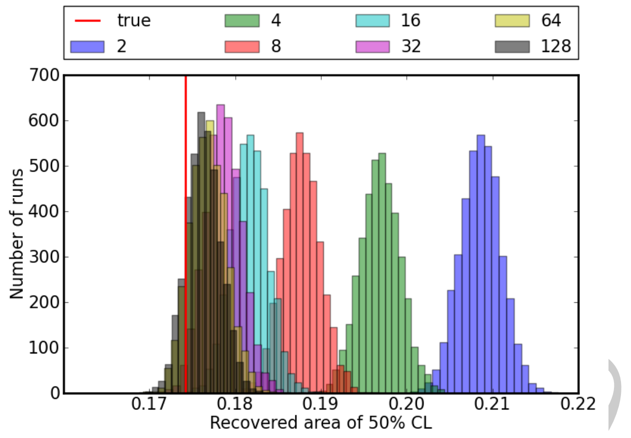

FIG. 9. Distribution of the 50% CR$_{min}$ areas recovered by the 2-step process from 4000 runs. The 2^{15} samples produced for each run are taken from a 2-dimensional *gaussian* distribution with a standard deviation of 0.2. The colours correspond to the number of samples in each bin.

an expectation value of 40% of the samples falling in the smaller bin, with some variance around this value. While the variance of the returned credible level of this bin will have increased due to less samples being used (half were thrown away), the expectation value for the credible level is now correct.

Areas

Until this point we have concentrated on whether the credible levels produced by the 2-stage kD-tree algorithm are consistent. What we would like is recovered CR$_{min}$ that can be considered close to optimal. Turning again to the 2-d gaussian distribution case we will now consider the 50% credible level as a useful example. In figure 8 we show the distribution of recovered areas of the 50% CR$_{min}$ for the 4000 realisations of the gaussian curve we used previously. For this simple case we can calculate the area of the (unique in this case) optimal CR$_{min}$ against which we can compare our methods. In figure 8 we plot this for both the standard kD-tree algorithm that uses the greedy algorithm to calculate CR$_{min}$, and the 2-stage version. The standard kD-tree results in consistently too small areas, which can be understood simply by remembering that the calculated credible levels are systematically too low. As this issue was fixed by the 2-stage kD-tree algorithm then we aren't biased towards small areas. While we now pick consistent credible levels we may not always choose the optimal one and this results in a tendency towards higher areas. The variance on the recovered area is greater for the 2-stage method than for the standard one. This is because we only use half the samples to calculate the credible levels.

In figure 9 we plot the same test, but only using the 2 stage algorithm and with differing numbers of samples per bin. This shows, as expected, that the more samples per bin the closer we get to the optimal case as the effect of Poissonian noise on samples per bin goes down. In practice we cannot always use so many samples per bin as it may be computationally expensive to produce so many independent samples.

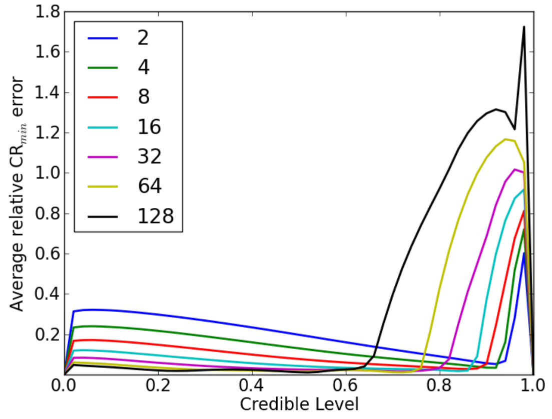

FIG. 10. Plot of the fractional errors on areas, (CR$_{min}$ - true)/true), vs CL. This was done using 4000 runs, each with 2^{15} samples drawn from a 2-dimensional *gaussian* n distribution. Colours denote the number of samples in each bin.

In figure 10 we plot the relative difference between calculated and true CR$_{min}$ for the full range of CLs. We can see that the calculated CR$_{min}$ are always greater than the true value, as expected from the consistency condition. Optimality is generally improved by increasing the number of samples, but for the larger CLs there is a sudden rise in the areas calculated. This is a result of there being too few bins

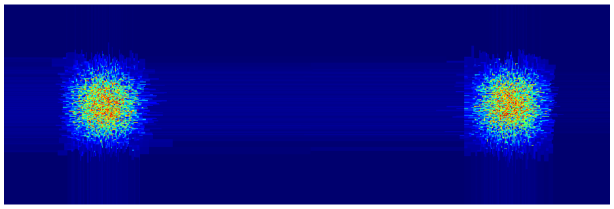

FIG. 11. The density of 2^{15} samples taken from the *bimodal gaussian* distribution has been plotted using the kD-tree algorithm. The standard deviation of each gaussian is 1/20th of the distance between the two peaks.

and so bins that extend to the parameter bounds become relevant. This is a problem because the edges of parameter space can often be chosen reasonably arbitrarily. With more samples and bins we can often find a more optimal CR$_{min}$ that doesn't include an edge bin.

The lower limit of the number of bins that are required such that no edge bins are used in the calculation of CR$_{min}$ is simply found. For any given setup with n^2 boxes there will be $4n-4$ boxes that touch the boundary. If for example we want the 90% CL then we need the fraction of bounding boxes to be < 0.1. i.e.

$$\frac{4n-4}{n^2} \leq 1 - \mathrm{CL} \tag{2}$$
$$\text{for CL} = 0.9 \quad \Rightarrow \quad n \geq\approx 39$$

so that if we want 64 samples per bin we are going to need at least $\approx 200,000$ samples, and $> 24,000$ for 8 samples per bin, remembering that we had to double the number of samples needed to apply the 2-step algorithm.

Bimodal Gaussian Distribution

As an example of a less trivial example we consider a bimodal distribution, created from two 2-dimensional gaussian distributions, each with a standard deviation of 0.2 and separated by 10, as seen in figure 11. The separation was chosen to be large so that the two gaussians do not overlap in any significant way. We can see how well the kD-tree copes with this by again testing the errors in areas over 4000 runs, each with 2^{15} samples, see figure 12. The performance of the 2-stage kD-tree algorithm seems to be reasonably consistent with previous results, with the exception of the placing of the breaking point in CL, at which the algorithm performs badly. This point is for smaller CL than before as we have effectively increased the number of edge bins. For the bimodal distribution there are a number of bins that stretch between the 2 peaks, that have a similar effect to edge boxes in that they result in over sized CR$_{min}$s. This is the reason that the previously calculated required number of bins (equation 2) was a minimum. Distributions that are more complicated than a simple gaussian, particularly multi-modal distributions, will require more bins to reach the required optimality.

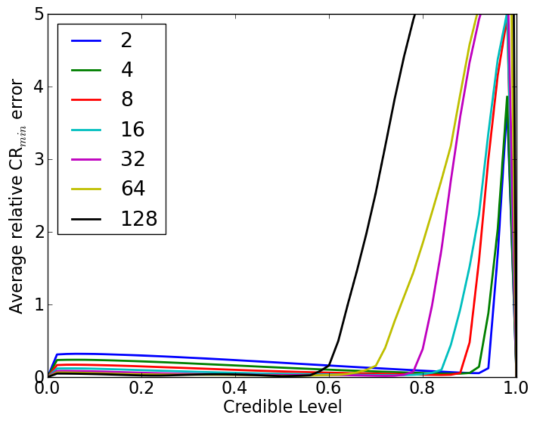

FIG. 12. Fractional errors on areas. This was done using 4000 runs, each with 2^{15} samples drawn from a 2-dimensional *bimodal gaussian* distribution. Colours denote the number of samples in each bin.

DISCUSSION

We have presented a post-processing algorithm that satisfies the condition of consistency. We have shown the conditions under which this 2-stage kD-tree produces reasonably optimal credible regions. There is clearly a balance to be struck between increasing the samples per bin to reduce the effects of Poissonian noise and having enough bins to both avoid parameter space edges and have good resolution. The structure of a kD-tree makes it particularly simple to use, in that it is still constructed from rectangular bins and integrating over regions is therefore straightforward.

We do not claim that this method is always the one to choose, but the issues we have considered must always be taken into account when constructing the post-processing analysis. One possible example is to use the standard kD-tree algorithm, but use the value of the posterior to order the bins. Within each bin will be a set of samples with attached values of the posterior which can be used to estimate the integrated posterior within that bin. This is less straightforward than first appears as we may have marginalised over a few parameters, and so the integral will be over a few dimensions, with only a few samples to evaluate this region. But again, the important feature is that the bin order is separated from the construction of the bins.

A very common method of recreating density functions from samples is kernel density estimation, where each sample is replaced with a function, often a multi-dimensional gaussian. The density function is then just the sum of all these. In reality this method has its own issues, the width of each gaussian must be calibrated and may not accurately represent the background distribution. A modified version with a 2-stage algorithm that also identifies clustering of samples has been implemented for the analysis of gravitational-wave data [4].

ACKNOWLEDGEMENTS

We are grateful to members of the LIGO-Virgo collaboration parameter estimation group, and particularly to Birmingham colleagues, for many useful discussions.

* tsidery@googlemail.com

[1] B. Silverman, *Density Estimation for Statistics and Data Analysis* (Density Estimation for Statistics and Data Analysis, 1986).
[2] T. Sidery *et al.*, Phys. Rev. D **89**, 084060 (2014).
[3] J. L. Bentley, Commun. ACM **18**, 509 (1975).
[4] L. P. Singer *et al.*, ArXiv e-prints (2014), arXiv:1404.5623 [astro-ph.HE].