A simulated annealing approach to parameter inference with expensive likelihoods

Aseem Paranjape

1 Inter-University Centre for Astronomy & Astrophysics, Ganeshkhind, Post Bag 4, Pune 411007, India

18 May 2022

ABSTRACT

We present a new approach to parameter inference targeted on generic situations where the evaluation of the likelihood \( L \) (i.e., the probability to observe the data given a fixed model configuration) is numerically expensive. Inspired by ideas underlying simulated annealing, the method first evaluates \( \chi^2 = -2 \ln L \) on a sparse sequence of Latin hypercubes of increasing density in parameter (eigen)space. The semi-stochastic choice of sampling points accounts for anisotropic gradients of \( \chi^2 \) and rapidly zooms in on the minimum of \( \chi^2 \). The sampled \( \chi^2 \) values are then used to train an interpolator which is further used in a standard Markov Chain Monte Carlo (MCMC) algorithm to inexpensively explore the parameter space with high density, similarly to emulator-based approaches now popular in cosmological studies. Comparisons with example linear and non-linear problems show gains in the number of likelihood evaluations of factors of 10 to 100 or more, as compared to standard MCMC algorithms. As a specific implementation, we publicly release the code \texttt{picasa}: Parameter Inference using Cobaya with Anisotropic Simulated Annealing, which combines the minimizer (of a user-defined \( \chi^2 \)) with Gaussian Process Regression for training the interpolator and a subsequent MCMC implementation using the \texttt{cobaya} framework. Being agnostic to the nature of the observable data and the theoretical model, our implementation is potentially useful for a number of emerging problems in cosmology, astrophysics and beyond.

Key words: methods: numerical

1 INTRODUCTION

Statistical inference in the context of parametrized models of observable quantities forms a cornerstone in addressing a variety of scientific questions. Focusing on astrophysics and cosmology, in particular, the advent of targeted experiments and observatories (Komatsu et al. 2011; Planck Collaboration et al. 2018; Borucki 2016; Ricker et al. 2014; Hobbs et al. 2010; Abbott et al. 2019) and large surveys (eBOSS Collaboration et al. 2020; Abbott et al. 2022) has led to unprecedented statistical precision being achieved on a number of observables spanning a large range of physical conditions and length scales in, both, the local and distant Universe. Ongoing and upcoming surveys and experiments are set to further enhance the statistical precision of measurements (Merloni et al. 2012; Abazajian et al. 2016; Laureijs et al. 2011; LSST Science Collaboration et al. 2009; DESI Collaboration et al. 2016; Baker et al. 2019), as well as open up new avenues for exploring the Universe (see, e.g., Koopmans et al. 2015), thus firmly placing the onus on model builders to produce ever-more-accurate models of the Universe.

In this context, the use of semi-analytical techniques and full-fledged numerical simulations as theoretical machines has now become common in data analysis and statistical inference (Kitaura et al. 2016; Yang et al. 2018; Alam et al. 2021b; Yuan et al. 2022b). These numerical tools are typically much more time-consuming and computationally expensive than (much simpler, but less accurate) analytical models, as is needed in order to correctly capture a large number of non-linear and uncertain model elements. For example, it is now common to have to describe complex physical phenomena such as galaxy formation and evolution using semi-analytical or heuristic models involving between 10-20 free parameters (Guo et al. 2011; Henriques et al. 2015; Alam et al. 2021a).

Within the cosmology community, such situations have led to several parallel lines of investigation, all aimed at robust statistical inference using high-precision data. The recognition that not all aspects of a full numerical simulation may be relevant for modelling specific observables has led to fast but approximate simulation techniques (typically with free parameters calibrated using full simulations) being developed to model cosmic structure formation (Scoccimarro & Sheth 2002; Monaco et al. 2002; Mesinger et al. 2011; Monaco et al. 2013; Kitaura & Heß 2013; Tassev et al. 2013; Howlett et al. 2015; Feng et al. 2016; Choudhury & Paranjape 2018). The requirement of precision of a few percent in predicting observables such as the power spectrum of cosmic matter fluctuations has led to the development of ‘emulators’, which smoothly interpolate the observable in multi-dimensional parameter space using full numerical simulations performed on a sparse
set of carefully chosen parameter locations (Heitmann et al. 2014, 2016; Lawrence et al. 2017; Knabenhans et al. 2019). More recently, several groups have begun exploiting machine learning techniques (e.g., Agarwal et al. 2018; Lucie-Smith et al. 2018; Villaescusa-Navarro et al. 2021; Dai & Seljak 2021; Delgado et al. 2021; Schaefer et al. 2021) to train artificial neural networks to reproduce the results of a full simulation at a fraction of the cost. Similar lines of approach can also be found in the gravitational waves (Schmidt et al. 2021), strong gravitational lensing (Hezaveh et al. 2017) and exoplanets communities (Cranmer et al. 2021).

The techniques listed above are typically optimized in a highly model- and/or observable-specific manner. It is therefore also useful to explore more general-purpose techniques for parameter inference that combine accuracy and speed. For example, the Derivative Approximation for Likelihoods (DALI, Sellentin et al. 2014) uses analytical expansions around a multivariate Gaussian (which would be the exact answer for a linear model with Gaussian data errors), to model the non-Gaussian posterior of non-linear models. Schemes invoking DALI have been used in recent analyses of specific data sets (Rizzato & Sellentin 2022; Wang et al. 2022). To account for arbitrary levels of non-Gaussianity in the parameter posterior, however, accurate likelihood calculations are essential. In situations where each likelihood calculation requires an expensive (semi-)numerical simulation, this can make efficient sampling of parameter space – using, e.g., Markov Chain Monte Carlo (MCMC) techniques – problematic.

There have already been several attempts to circumvent these problems. E.g., Yuan et al. (2022a) have explored a hybrid MCMC, in which an emulator for the 2-point correlation function observables is fed into a standard MCMC implementation, for analysing data from the BOSS survey. Bécsey et al. (2022) have used a Metropolis-within-Gibbs scheme to accelerate the sampling of nuisance parameters when searching for gravitational waves in pulsar timing array (PTA) data. The distinction between fast (typically nuisance) and slow (typically physically interesting) parameters has also been exploited in the construction of standard MCMC samplers based on the Metropolis-Hastings algorithm (Lewis 2013).

In this work, we present a new algorithm to accelerate the exploration of parameter space for generic problems with a calculable likelihood (or, equivalently, a cost function). Unlike DALI-based schemes such as the one proposed by Rizzato & Sellentin (2022), we do not assume any specific functional dependence of the cost function on the parameters. Our method instead is inspired by the ideas underlying simulated annealing – an established technique for approximate optimization which we recapitulate below – and essentially involves a simultaneous minimization and sparse multi-dimensional exploration of the cost function landscape. Building on this, our algorithm then further sparsely samples the parameter space in the vicinity of the minimum, followed by the construction of an interpolation scheme to enable inexpensive evaluations of the cost function at arbitrary parameter values. This interpolator is finally fed into a standard MCMC implementation to perform rapid exploration of the parameter space. We will demonstrate that our method uses fewer cost function evaluations than a standard MCMC solution by factors ranging from $\gtrsim 10$ to $\gtrsim 100$ depending on the problem; this is clearly particularly useful when the cost function (or likelihood) is expensive to compute. Our implementation is released publicly in the form of a code PICASA whose main features we describe later.

The paper is organised as follows. We describe our new method in section 2, followed by a description of the PICASA framework in section 3. We present applications to some example linear and non-linear problems in section 4 and conclude with a discussion in section 5.

## 2 METHOD

### 2.1 Recap: simulated annealing

Simulated annealing (Kirkpatrick et al. 1983) is a technique for locating the global optimum of a cost function that can depend on several parameters, useful for cases when the function evaluation is difficult. The idea involves starting a search at some finite ‘temperature’, leading to high probabilities of accepting non-optimal parameters, and slowly decreasing this temperature as better solutions are explored. The technique therefore allows for a stochastic exploration of parameter space that can effectively ‘jump out’ of local optima, unlike standard optimization algorithms such as gradient descent.

The original implementations of simulated annealing were closely connected to the Metropolis-Hastings algorithm (Metropolis et al. 1953; Hastings 1970), and probabilistically accept or reject parameters sequentially (Kirkpatrick et al. 1983, although see Dueck & Scheuer 1990; Moscato & Fontanari 1990 for a deterministic updating approach). The gradual decrease of the ‘temperature’ of the system ensures that the probability for accepting unlikely parameter values steadily decreases as the algorithm progresses. Adjusting the rate of this decrease (or the annealing schedule) leads to some level of control over the amount of resources that will eventually be spent in the cost function evaluations.

Our new method, described in the subsequent sections, is conceptually inspired by the idea of a steadily decreasing temperature. In particular, the algorithm we describe below progressively zooms in on smaller and smaller regions of parameter space enclosing the global minimum of a given cost function. However, we replace the probabilistic accept/reject scheme with a stochastic sampling strategy that leads to multiple parameter values being simultaneously sampled and stored. This is useful when we interface our algorithm with a standard MCMC implementation using interpolation schemes for evaluating the cost function. We also introduce several additional features that enable the algorithm to efficiently find a path towards the global optimum in multi-dimensional parameter space.

### 2.2 Anisotropic Simulated Annealing

In the following, we will use $\chi^2(\mathbf{a})$ to denote the cost function in the $D$-dimensional space of parameters $\mathbf{a}$. Throughout, we interpret $\chi^2$ as

$$\chi^2(\mathbf{a}) \equiv -2 \ln \mathcal{L} + \text{constant}, \quad (1)$$

where $\mathcal{L} \equiv p(\mathcal{O}|\mathbf{a})$ is the likelihood, or probability density (in observable space) of observing the data set $\mathcal{O}$ given a model with fixed parameters $\mathbf{a}$. For many applications of interest, when data errors are assumed to be (approximately) normally distributed with a parameter-independent covariance
matrix, the constant can be assumed to remove the effect of determinant factors, so that $\chi^2(a)$ corresponds to the quantity usually minimized in least squares applications. This allows us to cleanly connect to standard Bayesian MCMC applications later.  We emphasize again that we are not assuming any specific dependence of $\chi^2$ on the model parameters $a$ (e.g., unlike DALI, we are not assuming that $L$ is close to a multivariate Gaussian in the parameters).

2.2.1 Iterative sampling

The anisotropic simulated annealing (ASA) algorithm is built around the idea that an approximate minimum of $\chi^2(a)$ can be obtained using a sparse stochastic sampling in parameter space, and then iteratively refined. With some assumption on parameter priors, the samples obtained over multiple iterations can then also be used to map out the parameter posterior distribution around the minimum of $\chi^2$, using appropriate D-dimensional interpolation.

To this end, the ASA algorithm first evaluates $\chi^2(a)$ on Latin hypercube samples generated in progressively zoomed regions of the D-dimensional parameter space, with progressively higher density, for some number of iterations. The zoomed region at iteration $n$ is centered on the evaluated minimum of iteration $n-1$, with a volume decided by the estimated width of $L = e^{-\chi^2/2}$ in each parameter direction at iteration $n-1$, and having the same number of samples $N_{\text{samp}}$. The shape of the zoomed region at iteration $n$ is allowed to respond to anisotropic gradients of $\chi^2$ by setting the parameter range in the $p^{th}$ direction to the dispersion $\sigma_p^{(n-1)}$ estimated by treating $L$ defined by the points in iteration $n-1$ as a probability density along each parameter direction and numerically evaluating the corresponding variance: $\sigma_p^{(n-1)} = \int da_p L_{(n-1)}(a_p - \langle a_p \rangle)^2 / \int da_p L_{(n-1)}$, where $\langle a_p \rangle = \int da_p L_{(n-1)} a_p / \int da_p L_{(n-1)}$. The use of points only in iteration $n-1$, rather than in all previous iterations, means that the algorithm explores regions of $L$ progressively closer to its maximum. The choice of setting the width of subsequent iterations exactly equal to the dispersion calculated above effectively fixes the rate of decrease of the ‘temperature’ of the annealing process.

Since the $N_{\text{samp}}$ evaluations in each iteration are completely independent, they can be trivially parallelized over $N_{\text{proc}} \geq 1$ processors. Maximum efficiency is then clearly achieved when $N_{\text{proc}}$ is a multiple of $N_{\text{samp}}$.

2.2.2 Detecting edges

If the estimated minimum is too close to the edge of the parameter range in the $p^{th}$ direction at any iteration $n$, the estimate of the dispersion along that direction may be inaccurate, leading to errors that could propagate into subsequent iterations. To avoid this, in such a situation the algorithm forces the width of the parameter range in the $p^{th}$ direction for iteration $n+1$ to be the same as in iteration $n$ for that direction, but centered on the updated estimate of the minimum. Thus, the algorithm can automatically ‘claw’ its way outside ranges that are too narrow, in its search for the true minimum.

The condition of being ‘too close’ is parametrized by a variable $\epsilon_{\text{close}}$. If the absolute difference between the location of the minimum and either edge of the current parameter range along the $p^{th}$ parameter direction is less than $\epsilon_{\text{close}}$ times the width of the current parameter range along that direction, then the minimum is declared ‘too close’ to the edge. Unless specified, we set $\epsilon_{\text{close}} = 0.1$ in what follows.

2.2.3 Convergence criterion

The iterations are terminated when a pre-decided convergence criterion is met. Below, we use two criteria to define convergence:

(a) The values of the minimum $\chi^2$ between iterations $n-1$ and $n$ change by a relative amount smaller than $\epsilon_{\text{conv}}$, i.e.,

$$\frac{\chi^2_{\text{min},(n-1)}}{\chi^2_{\text{min},(n)}} - 1 \leq \epsilon_{\text{conv}},$$

where $0 < \epsilon_{\text{conv}} < 1$. We demand that this criterion be met over two successive iterations.

(b) The flow towards the minimum of $\chi^2$ is converging, i.e., the estimated dispersion along each parameter direction must have decreased from iteration $n-1$ to $n$.

Convergence is declared to be achieved when both these criteria are simultaneously met.

2.2.4 Escaping local minima

In practice, it is possible for the above algorithm to become stuck in a local minimum, with the convergence criterion being properly met but at relatively large $\chi^2$. This is because, as we mentioned earlier, the rate of decrease of the annealing ‘temperature’ in our case is fixed by our choice of how the width of subsequent iterations is made. This can sometimes lead to the temperature decreasing ‘too quickly’ for a given data and model setup. (This is not surprising, since the annealing schedule in the original simulated annealing algorithm is also highly model specific.)

To detect such situations, our implementation in section 3 performs a simple goodness-of-fit test using the evaluated minimum $\chi^2_{\text{min}}$ and the number of degrees of freedom $d \equiv N_O - D$, where $N_O$ is the length of the data (or observable) vector $O$. We demand that the $p$-value assuming that the converged value $\chi^2_{\text{min}}$ is $\chi^2$-distributed with $d$ degrees of freedom is larger than a certain threshold. If not, then the algorithm attempts to break out of the local minimum by starting a new iteration centered on the current minimum but with a large parameter range.

In practice, we have found that such situations can also
occurs if the starting range is very broad and the sampling density is too low, in which case the algorithm can potentially get stuck in an erroneous local minimum created by numerical errors. In such situations, it can become difficult for the algorithm to settle onto a track towards the true minimum and, instead, resources are wasted in very unlikely regions of parameter space. It then becomes advisable to quickly detect such pathological situations and restart the algorithm with a narrower starting range and/or higher sampling density. This is included in our implementation below.

2.2.5 Quadratic form estimate

Once an acceptable minimum $\mathbf{a}_{\text{eval}}$ over the sampled parameter points is located, we fit a positive-definite quadratic form $Q(\mathbf{a})$ in its vicinity:

$$Q(\mathbf{a}) = \chi^2 + (\mathbf{a} - \mathbf{a}_i)^T \mathbf{C}^{-1} (\mathbf{a} - \mathbf{a}_i),$$

where the scalar $\chi^2$, the D-vector $\mathbf{a}_i$, and the symmetric $D \times D$ matrix $\mathbf{C}$ are obtained using a standard least squares criterion by minimizing $(Q(\mathbf{a}^{(i)}) - \chi_i^2)^2$ for a given sample of points $\{\chi_i^2, \mathbf{a}^{(i)}\}$. In practice, we ensure that $\chi^2$ is positive definite by starting with a small number of points to fit for $Q$ and gradually increase the sample size onwards from $\mathbf{a}_{\text{eval}}$ until a positive definite $\mathbf{C}^{-1}$ is obtained, inverting which gives us $\mathbf{C}$.

This fitting exercise has the dual advantage of not only refining the estimate for the true minimum (i.e., we use $\mathbf{a}_i$ rather than $\mathbf{a}_{\text{eval}}$ as our best estimate), but also providing a zeroth order estimate for the errors on the best-fit (assuming broad priors) through the covariance matrix $\mathbf{C}$. In section 3.1, we also discuss how failures in estimating a positive definite $\mathbf{C}$ can be handled.

2.3 Visual demonstration

To demonstrate the basic behaviour of the ASA algorithm, consider some predicted scalar observable $y$ as a function of a 1-dimensional predictor variable $x$, sampled at $N_O$ points $\{x_i\}, 1 \leq i \leq N_O$ (the algorithm is insensitive to this choice). The data $\{y_i\}$ are assumed to be normally distributed around the ‘truth’ $y = f(x)$ with independent errors $\{\sigma_i\}$ (i.e., a diagonal covariance matrix with diagonal entries $\sigma_i^2$):

$$y_i \sim \mathcal{N}(f(x_i), \sigma_i^2).$$

The cost function minimized by the ASA algorithm is then taken to be

$$\chi^2(\mathbf{a}) = \sum_{i=1}^{N_O} \frac{1}{\sigma_i^2} (y_i - M(x_i; \mathbf{a}))^2,$$

where $M(x; \mathbf{a})$ is the model function which depends on the $D$-dimensional parameter vector $\mathbf{a}$.

As a warm-up, and to visually display how the ASA algorithm locates the minimum of the cost function, consider the simple case $f(x) = x$ which we fit using the straight line $M(x; \mathbf{a}) = a_0 + a_1 x$. For this exercise, we sampled the data at 11 linearly spaced values between $0 \leq x \leq 2$, assigning errors $\sigma_i = 0.5$ for each point. The minimization of $\chi^2$ (equation 4) can be performed analytically (e.g., Press et al. 1986) and gives the following best fit $\mathbf{a}$ and covariance matrix $\mathbf{C}$ for the particular data realisation $\{y_i\}$ we used:

$$a_0 = -0.495253; \quad a_1 = 1.497192,$$

$$C_{00} = 0.079545; \quad C_{01} = -0.056818; \quad C_{11} = 0.056818.$$

Notice that, although the data errors were assumed constant and independent (hence isotropic in data space), the resulting parameter covariance is not even diagonal, much less isotropic. The ASA algorithm must therefore explore a genuinely anisotropic parameter landscape. This is generally true for all the examples below.

To implement ASA, we must choose the starting range of parameters and the value of $N_{\text{samp}}$, which then also decides the starting sampling density. Fig. 1 shows the sampled points when starting with a broad (left panel) or narrow (right panel) starting parameter range, using $N_{\text{samp}} = 50$. In the first case, where the starting range comfortably enclosed the analytical values $(a_0, a_1)$ from equation (5), the algorithm quickly (within 2 zooms) centered itself close to the analytical values and converged to better than 0.1% variation in the minimum value of $\chi^2$ after 8 zoom iterations, i.e., 400 $\chi^2$ evaluations in total. In the second case, where the analytical values were far outside the starting range, the algorithm took 8 zooms to discover the vicinity of the analytical values and subsequently converged after the 14th zoom level, i.e. 700 $\chi^2$ evaluations in total. (The visualisation also demonstrates how the algorithm ‘clawed’ its way to the minimum in this case; see section 2.2.2.) Thus, a broader starting range (analogous to a higher starting ‘temperature’) actually led to fewer overall evaluations for this simple case. The fitted quadratic form in the vicinity of the evaluated minimum $\chi^2$ for the first (second) case returns a best fit and covariance within one part in $10^3 (10^9)$ of the analytical values. This can also be seen from the excellent agreement between the red and blue stars and corresponding $1\sigma$ ellipses in Fig. 1.

We also repeated this exercise with lower values of $N_{\text{samp}}$ for the same starting parameter ranges for each case. For the first case (broad starting range), we find that $N_{\text{samp}}$ as small as 10 still leads to excellent convergence centered on the analytical values with a correspondingly smaller number of function evaluations ($\approx 90$), and a similar recovery of the best fit and covariance matrix. The second case (narrow starting range) shows interesting behavior: while the algorithm itself converges increasingly farther from the analytical values as $N_{\text{samp}}$ is decreased ($\approx 2\sigma$ away for $N_{\text{samp}} = 10$), the resulting best fit and covariance matrix estimated from the fitted quadratic form remain within one part in $10^4$ of the analytical values.

This demonstrates the interplay of $N_{\text{samp}}$ and the starting parameter width for a simple 2-dimensional linear problem, and already suggests that, to ensure good sampling around the true minimum of $\chi^2$ in a generic situation, it is always useful to perform a few sparse iterations post-convergence, centered on the best-fit returned by the quadratic form fitting. Below we will see how this behaviour is affected by an increase in the dimensionality and/or non-linearity of the problem.

2.4 Interpolating the cost function

Once the ASA algorithm converges, possibly along with additional iterations to improve the uniformity of sampling around the estimated minimum, we can use the resulting sample of
Any GPR is defined by the choice of left (right) panel (narrow) starting parameter range in the

While the specific choice of kernel can be model-dependent, with one of two kernels as discussed below works adequately, approximates the observable rather than the likelihood, and its kernel, which defines the variance of the Gaussian probability density in function space which characterises the GPR.

Each GPR kernel is specified by a ‘hyper-parameter’ vector \( \mathbf{h} \): e.g., an anisotropic Gaussian kernel describing \( \chi^2(\mathbf{a}) \) for a D-dimensional parameter space \( \mathbf{a} \) requires a \( D+1 \)-dimensional \( \mathbf{h} \) to be specified (the dispersions in each parameter direction, along with an overall amplitude), while the isotropic rational quadratic kernel requires 3 parameters, namely, an amplitude, a dispersion scale and a shape parameter (see chapter 4 of RW05). The values of the components of \( \mathbf{h} \) are model-specific and need to be fixed by an appropriate comparison (or ‘training’) with the provided sample \( \{ \chi^2_i, \mathbf{a}^{(i)} \} \). We use Algorithm 2.1 of RW05 as implemented in Scikit-Learn, which maximises a (hyper-)likelihood function for a given training set, to set up an iterative training procedure as follows.

- We start with a small fraction of the available sample as the training set to estimate \( \mathbf{h} \), using the ASA algorithm to explore hyper-parameter space.
- We cross-validate this choice of \( \mathbf{h} \) using the remaining samples and compute the 1 and 99 percentiles of the relative difference \( \chi^2(\mathbf{a})/\chi^2(\mathbf{a}) - 1 \).
- If the magnitudes of these percentiles are smaller than a pre-defined threshold \( \epsilon_{cv} \), then we declare the GPR as trained with the hyper-parameter \( \mathbf{h} \). If not, we increase the size of the training set by a small amount (20%, below) and repeat.

If the GPR cannot be trained with more than, say, 80% of the sample, then the interpolation may not be accurate. This typically corresponds to situations where the \( \chi^2 \) sampling near sharp features is not fine enough, or when the overall sampling density is too low. In this case, one could either weaken the requirements for converged training (e.g., by increasing \( \epsilon_{cv} \).
We have implemented the ASA algorithm along with GPR training and subsequent sampling using standard MCMC, in the Python code PICA$^S$A (Parameter Inference using Cobaya with Anisotropic Simulated Annealing). We use the publicly available COBAYA framework (Torrado & Lewis 2019, 2021) to interface the ASA algorithm with MCMC sampling. We describe the main functionality of PICA$^S$A in this section and discuss a few examples in section 4.

### 3 PICA$^S$A Framework

We have implemented the ASA algorithm along with GPR training and subsequent sampling using standard MCMC, in the Python class ASAUtilities which, by default, are called by the black-box methods defined in the class PICA$^S$A, but can also be directly accessed by the user. The user provides a dictionary (or ‘data pack’) containing the function objects and data arrays required to calculate the cost function $\chi^2(a)$ for any specified parameter vector $a$. In other words, both the model as well as shape of the likelihood are entirely user-defined, making our implementation generic. The data pack also requires the specification of $N_{\text{samp}}$ (key $N_{\text{samp}}$) and the starting range of parameter values (keys $\text{param}_{\text{mins}}$ and $\text{param}_{\text{maxs}}$), along with a writable output file location (key $\text{chi2}_\text{file}$).

Keeping in mind the discussion in section 2, the default implementation performs two basic passes over parameter space to locate the minimum $\chi^2$, followed by a nested sequence of independent ‘last iterations’ that sample $\chi^2$ around its minimum. In detail, these steps proceed as follows:

(i) In the first pass, the ASA algorithm starts with the user-specified parameter ranges and value of $N_{\text{samp}}$ to crudely estimate the minimum of $\chi^2(a)$. This is done by setting $\epsilon_{\text{conv}} = 0.1$ (a relatively large value), allowing for quick convergence to a possibly inaccurate minimum.

(ii) Given this ‘coarse’ sampling, the algorithm sets up a higher resolution second pass as follows. First, it attempts to perform the quadratic fit discussed in section 2.2.5. If successful, the parameters are rotated into the eigenspace of the covariance matrix $C$, and the parameter ranges for the second pass are set to be $1\sigma$ wide in each eigen-direction around the estimated minimum. If a positive definite quadratic form cannot be obtained, the parameters are not rotated and the ranges for the second pass are set to be $10\%$ of the user-specified width, centered in this case on the currently evaluated minimum. We refer to this setup as the ‘minimizer mode’ of the ASA algorithm.

(iii) The second pass of the ASA algorithm proceeds in this minimizer mode with the same $N_{\text{samp}}$, but with a more demanding convergence criterion $\epsilon_{\text{conv}}$. By default this is set to be $\epsilon_{\text{conv}} = 0.001$ (i.e., only $<0.1\%$ variations in minimum $\chi^2$ are tolerated), but can also be specified by the user. In this mode, the algorithm uses high-density samples in small parameter volumes (‘clawing’ its way, if needed, towards the minimum; see section 2.2.2) to find a well-converged answer.

(iv) If the minimizer mode is successful, a new estimate of the quadratic form is obtained using, by default, the full set of samples now available. This is already expected to give a high-precision estimate of the true minimum. To facilitate further parameter exploration, the algorithm now proceeds to set up the ‘last iterations’ sampling the vicinity of the true minimum.

This is done by performing a series of single ASA iterations, each using fixed hyper-rectangles in parameter eigenspace having widths ranging from 0.25$\sigma$ to $\delta_2\sigma$ in steps of 0.25$\sigma$ along each eigen-direction. Here, the dispersion along each eigen-direction is determined by the eigenvalues of the covariance matrix $C$, and $\delta_2$ (set by the data pack key $\text{dsig}$) is a user-defined constant. We have found that $\delta_2 \approx 5-6$ is sufficient for linear or nearly linear problems, while sufficiently non-linear problems may need $\delta_2 \gtrsim 10$ (since the Gaussianised estimate of parameter error might not capture the true width of the non-Gaussian posterior). Working in eigenspace is very useful in ensuring efficient sampling when two or more parameters have strong degeneracies.

The number of samples evaluated in each of these last iterations is controlled by a variable $f_{\text{last}}$ (set by the data pack key $\text{last}_\text{frac}$, with a default value $f_{\text{last}} = 0.05$), which specifies the fraction of the total sample size at the end of the minimizer mode that should be used to sample each iteration. Along with the value of $\delta_2$, $f_{\text{last}}$ can be adjusted by the user to ensure that the total number of samples is dominated by those in the last iterations, which helps the GPR training below to be performed efficiently.

(v) Steps (i) and (iii) above are further constrained by demanding that the total number of iterations in each not exceed 50 and 100, respectively. This ensures that, in cases where the algorithm gets pathologically stuck in local (numerically generated) minima, precious computational resources are not wasted in unlikely parameter regions. If either of these maximum iterations are exceeded, the code exits with a recommendation to the user to either decrease the width of the initial sampling range, or increase $N_{\text{samp}}$, or both.

(vi) The user can optionally set the data pack key parallel to True, which invokes parallelization of the $N_{\text{samp}}$ evaluations in any single iteration using the multiprocessing package. For optimal efficiency, the attribute PICA$^S$A.NPROC may be set to a multiple of $N_{\text{samp}}$.

(vii) Finally, the code discards any samples where the value of $\chi^2$ is not finite. This can be utilised by the user to impose

---

3 If a positive definite quadratic form cannot be obtained after the minimizer mode, the algorithm attempts to adjust by restarting the optimization procedure around the current minimum, using half the current starting range. The maximum number of such restarts is controlled by the data pack key max_opt_iter (default value 10).
sharp priors, by simply requiring the model function to evaluate to infinity (numpy.inf) in undesirable or unphysical regions of parameter space.

The end result of this exercise is a complete sample of finite \( \chi^2 \) values and parameter vectors, along with a final, more accurate estimate of the quadratic form close to the minimum, which can be fed into the GPR training procedure described in the next subsection. In cases where the problem is known to be nearly linear in the parameters, or where the location of the minimum is more important than its error, the ASA output at this stage may already provide all the required information. For this reason, the above steps are collected together in a single black-box method PICASA.optimize.

3.2 GPR training

We use the output of the ASA implementation above to train a GPR for interpolating \( \chi^2(a) \) using the technique outlined in section 2.4. This is implemented by the method PICASA.gpr_train_and_run. The choice of GPR kernel is specified by the data pack key kernel which can take values ‘rbf’ (this is the default) for the anisotropic Gaussian kernel or ‘rq’ for the isotropic rational quadratic kernel.

The starting training set size is set to the minimum of 1000 and 5% of the full sample, with a lower cap at 100. The training set is chosen by uniformly downsampling the full sample. The method invokes the ASA algorithm to optimize the GPR hyper-parameter vector \( h \), setting \( eps_{conv} = 0.005 \) and \( eps_{close} = 0.2 \). The starting training iteration employs ASA with \( N_{samp} = 50 \) and a logarithmic hyper-parameter range centered on zero for each component of \( \ln(h) \) with a width of \( \pm \ln(2) \). While the cross-validation threshold \( \epsilon_{cv} \) (data pack key cv_thresh) is exceeded, further training iterations are performed, using training sets successively larger by factors of 1.2, \( N_{samp} \) successively larger by factors of 1.15 and ASA starting ranges successively centered on the current best estimate of \( \ln(h) \) with widths successively larger by factors of 1.25. Thus, the training iterations use gradually increasing training sets and explore gradually increasing regions of hyper-parameter space with gradually increasing \( N_{samp} \). If the GPR training and cross-validation has not converged for a training set size larger than 80% of the sample size, or if a maximum number of training iterations (by default 25) has been exceeded, the method exits with the last available hyper-parameter values and training size. In such cases, it is recommended to either alter the training thresholds and/or change the GPR kernel and/or increase the size of the original \( \chi^2 \) sample (say by performing additional last iterations, see also section 2.4). This introduces an element of problem-specific user intervention in the GPR training, which we comment on in section 5.

The user must supply a writable directory location (data pack key chain_dir) which is used (among other things) to store the output of the training iterations, namely, the training size, values of hyper-parameter components and cross-validation percentiles. The final such outputs are used below for performing an MCMC run by feeding a trained GPR to the COBAYA framework.

3.3 MCMC exploration

We use the publicly available COBAYA framework to perform MCMC sampling using a trained GPR to evaluate \( \chi^2(a) \). We use the Likelihood class of COBAYA to define the class ASALike, in which the method ASALike.logp calculates \(-0.5 \chi^2 \) using the provided trained GPR. This setup is accessible to the mcmc sampler (Lewis 2013) available in COBAYA, which we use for our MCMC exploration. The functionality for running COBAYA’s mcmc sampler using the trained GPR is collected in the method PICASA.run_trained_gpr. This performs and stores the MCMC chains at the same writable location specified by the data pack key chain_dir that was used by the GPR training above. We refer the reader to the COBAYA documentation page for implementation details of the mcmc sampler and other usage.

We optimize the MCMC run by utilising the information collected during the ASA implementation, as follows.

- **Priors** are chosen to be uniform in ranges \( \pm \delta_\sigma \sigma \) along each parameter direction, centered on the best-fit parameter vector \( a \), obtained from quadratic form fitting on the full sample, where \( \delta_\sigma \) is the same variable specified by the data pack key dsig (see section 3.1). Note that the ASA optimization used \( \delta_\sigma \) in eigenspace, in contrast to the parameter space priors here.
- The **reference point** (starting location of each chain) is set equal to \( a \).
- The **proposal covariance matrix** is initialised to be a small factor (default 0.1, set by the data pack key prop_fac) times the best-fit covariance matrix \( C \) from the quadratic form estimate.

The mcmc sampler is run in its standard mode where it learns the proposal covariance matrix dynamically. The user can adjust the MCMC stopping criterion (data pack key Rminus1_stop, default value 0.005). We emphasize that, since the MCMC sampling uses the trained GPR to estimate \( \chi^2(a) \), rather than performing expensive actual calculations of \( \chi^2(a) \), a demanding value of Rminus1_stop such as 0.005 is not computationally challenging. Rather, it is the ASA parameter \( \epsilon_{conv} \) (accessible through the optional data pack key eps_conv which specifies the value of 100 \( \epsilon_{conv} \)) that behaves analogously in PICASA to Rminus1_stop in a standard MCMC run.

For the user’s convenience, we have collected together all aspects of this parameter inference framework in the single black box method PICASA.mcmc. This calls PICASA.optimize (if needed, repeatedly) to obtain accurate estimates of \( a \) and \( C \), then calls the GPR training and MCMC running routines described above to produce and store a full MCMC sample. We have also provided some convenience flags using the data pack keys no_cost, read_file_mcmc and no_train that function as follows.

- If no_cost is True, the user must provide a single callable function that calculates \( \chi^2 \), rather than separate data and error arrays along with model and cost function callables. This allows for easier integration with pre-existing codes set up for MCMC sampling.
- If read_file_mcmc is set to a valid file path containing \( \{ \chi^2, a_i \} \) samples, then the ASA optimization step is skipped by PICASA.mcmc and the code directly proceeds to GPR training after reading the specified file. This is useful if the outputs of
We now turn to a more detailed comparison of the perfor-
work, although we expect similar results to hold.

4 EXAMPLES

We now turn to a more detailed comparison of the perfor-
mance of the ASA algorithm and its integration with MCMC
sampling using PICASA, as compared to standard MCMC sam-
pling. For simplicity, we use the same data structure discussed
in section 2.3 (see the text around equation 4). Throughout,
whenever discussing standard MCMC results, we discard only
1000 accepted steps as burn-in. While this appears overly
generous when comparing the total number of likelihood calls
with PICASA, it is offset by the fact that we will use a rather
stringent convergence criterion for MCMC chains in all our
comparisons. PICASA does not suffer from burn-in, so that
the total number of chi^2 evaluations is relatively easy to track.
Throughout, to define the standard MCMC, we will use the
mcmc sampler (Lewis 2013) in the COBAYA framework (Tor-
rado & Lewis 2019, 2021). We leave a comparison with other
popular MCMC frameworks such as emcee (Foreman-Mackey
et al. 2013) or zeus (Karamanis & Bentler 2020) to future
work, although we expect similar results to hold.

4.1 Linear models

We first consider polynomial forms for the ‘truth’ f(x) as pro-
totypical of generic linear models: for a degree P polynomial,
we have

f(x) = \sum_{p=0}^{P} a_p x^p, \quad (6)

with \( a = \{a_p\}_{p=0}^{P} \) being the true values of the \( P + 1 \) model
parameters. We show results for the cases \( P = 3 \) and \( P = 5 \),
leading to 4- and 6-dimensional parameter spaces, respectively.
We use the following forms for the truth f(x):

\[
P = 3: \quad f(x) = 2 - 3x/2 - x^2/2 + x^3/2 \quad (7)
\]
\[
P = 5: \quad f(x) = 2 - 3x/2 + x^3/5 - x^3/5 + x^4 - x^5/2 \quad (8)
\]

We fit these using polynomial models of the respective degree
\( P \), i.e., we assume that the polynomial degree is known. In
each case, we sample the data on 40 linearly spaced points in
the range \(-0.5 \leq x \leq 2\), with constant errors \( \sigma_i = 0.2 \). The
data realisation for \( P = 3 \) (\( P = 5 \)) is shown in the left (right)
panel of Fig. 2.

We use the full PICASA framework with starting ranges as
discussed below and the default set of last iterations out to \( 5\sigma \)
in each parameter eigen-direction in each case (see section 3
for details). The subsequent MCMC using a trained GPR
(defined using the default anisotropic Gaussian kernel) is then
compared with a standard MCMC implemented using COBAYA
with the mcmc sampler (Lewis 2013). The latter is performed
using uniform priors in the range \(-5 \leq a_p \leq 5 \) and arbitrarily
chosen reference points for each parameter, while the MCMC

\footnote{This is not a particularly restrictive assumption because, e.g.,
not knowing \( P \) and fitting with a polynomial of degree \( Q > P \)
is equivalent to fitting a model with known degree \( Q \) but having
\( a_{P+1}, \ldots, a_Q = 0 \).}
4.1.1 Cubic polynomial

For the case $P = 3$, we set $N_{\text{samp}} = 30$ with starting ranges $-5 \leq a_p \leq 5$ for each parameter for the ASA, and set a relatively low sampling for the last iterations using $f_{\text{last}} = 0.025$. The ASA optimization for the displayed example successfully completed using 850 samples in total (i.e., including all last iterations). The iterative GPR training converged (with the 1.99 cross-validation percentiles being smaller than 0.01) using 144 of the 850 samples. The resulting best fit and 68% confidence interval in data space is shown by the solid red curve and error band in the left panel of Fig. 2. The corresponding 1σ and 2σ confidence regions in parameter space after MCMC sampling are displayed by the red contours in Fig. 3.

These results can be compared with the output of the standard MCMC, shown by the dashed purple curve in the left panel of Fig. 2 and the purple contours in Fig. 3. The analytical best fit is shown by the dot-dashed blue curve in the left panel of Fig. 2. We see excellent agreement between the PICASA, standard MCMC and analytical results. The agreement between PICASA and standard MCMC extends to the pairwise 3σ contours, which is important for applications requiring the calculation of the Bayesian evidence. The key point to emphasize, though, is that the total number of likelihood evaluations $N_{\text{eval}}$ in the standard MCMC exceeded 92000, in PICASA is initialised using the ASA output as described in section 3.3. Both sets of MCMC chains are run until the modified Gelman-Rubin index $R - 1$ used by the mcmc sampler falls below 0.002.
Figure 4. Parameter constraints: quintic polynomial. Same as Fig. 3, showing results for the quintic polynomial \( P = 5 \) from equation (8). We again see excellent agreement between PICASA and the standard MCMC even at the 3\( \sigma \) level despite strong parameter degeneracies, with PICASA outperforming the standard MCMC by a factor \( \sim 35 \). See text for a discussion of typical gains averaged over many runs.

while PICASA achieved its results with 850 evaluations, a gain of a factor \( \simeq 108 \). For a fair comparison, it is also important to consider that the total number of likelihood evaluations in both PICASA and the standard MCMC can fluctuate from run to run. By repeating the standard MCMC sampling multiple times, we determined that the median and central 68\% region of the distribution of \( N_{\text{eval}} \) with the standard MCMC is \( (58.9^{+20.6}_{-19.7}) \times 10^3 \) for this data realisation of the \( P = 3 \) model. A similar exercise for PICASA with exactly the same setup as above but changing the random number seed for ASA sampling and subsequent steps gave \( N_{\text{eval}} = 1365^{+796}_{-515} \), with \( \sim 18\% \) of runs having optimization convergence failures leading to discarding \( N_{\text{samp}} \times 50 = 1500 \) samples each (see section 3.1). Thus, we see that gains of factors exceeding 50 are typical when comparing PICASA with standard MCMC, although PICASA is less robust to the choice of starting parameter range. Indeed, simply decreasing the initial parameter range to \(-2.5 \leq a_p \leq 2.5 \) for PICASA led to \( N_{\text{eval}} = 1080^{+721}_{-360} \) with a failure rate of only 3\%.

4.1.2 Quintic polynomial

We repeated a similar exercise for the \( P = 5 \) case from equation (8). For the example run displayed in the right panel of Fig. 2, we used \( N_{\text{samp}} = 100 \) with a starting range \(-3 \leq a_p \leq 3 \) and a relatively high sampling for the last iterations using \( f_{\text{last}} = 0.1 \), leading to a total sample size of \( N_{\text{eval}} = 5400 \), with GPR training convergence achieved using
will typically be achieved within a few thousand evaluations, with a smaller starting range and a larger.

When using with uniform priors in the range $1 \leq a_p \leq 5$, the overall gain compared to standard MCMC (even samples wasted. However, if the user then restarts the run succeeds. In practice, for example, starting with starting ranges lead to high rates of optimization failure in cases).

Thus, we see that with increasing dimensionality, broad starting ranges lead to high rates of optimization failure in PICASA, which fall as the starting range decreases in width, and PICASA very effectively recovers the minimum of $\chi^2$ whenever the run succeeds. In practice, for example, starting with $-5 \leq a_p \leq 5$ and $N_{\text{samp}} = 30$ in our $P = 5$ example above would almost certainly lead to a failed optimization, with $\approx 1500$ samples wasted. However, if the user then restarts PICASA with a smaller starting range and a larger $N_{\text{samp}}$, convergence will typically be achieved within a few thousand evaluations, so that the overall gain compared to standard MCMC (even accounting for the wasted samples), would still be factors of $\sim 10-50$.

### 4.2 Non-linear models

We next move on to models that are non-linear in the parameters $a$, considering the following two examples for the ‘truth’:

\begin{align}
  f_1(x) &= |x|^{3/2} \sin^2(x), \\
  f_2(x) &= |x - 1/2| \cos(\sqrt{|x|/2}),
\end{align}

which we sample in the range $-2 \leq x \leq 4$. For the errors, we assume a constant value $\sigma(x) = 0.4$ for $f_1(x)$ and $\sigma(x) = (1/2)[\tanh(|x| + 1)/\tanh(1)]^3$ for $f_2(x)$. The resulting data realisations for $f_1$ and $f_2$ are respectively shown in the left and right panel of Fig. 5.

We respectively fit these two data sets using the 3-dimensional models

\begin{align}
  M_1(x; a) &= a_0 |x|^{a_1} \sin^2(x + a_2), \\
  M_2(x; a) &= a_0 |x - a_1| \cos(\sqrt{a_2|x|}).
\end{align}

In equation (12), we have chosen to place the parameter $a_2$ under the square-root, along with a prior $a_2 \geq 0$, to avoid a perfectly degenerate maximum of the likelihood. Similarly, in equation (11), we impose a prior $-\pi \leq a_2 \leq \pi$ by ensuring that the model returns infinity outside this range (such values are discarded by the ASA sampling, see section 3.1). The current implementation of ASA would otherwise always sample only one of the multiple, perfectly degenerate minima of $\chi^2$. We leave a more careful treatment of (nearly) degenerate minima of the cost function to future work.

The left and right panels of Fig. 6 are formatted identically to Fig. 3 and respectively show the MCMC results for recovering $f_1$ and $f_2$. In both cases, the PICASA runs used starting ranges of $-1 \leq a_p \leq 3$ for all parameters, with the same ranges also being used to set uniform priors for the...
standard MCMC runs. The MCMC convergence criterion was set to $R - 1 < 0.002$ (0.001) for $f_1$ ($f_2$). While the PICASA run for recovering $f_1$ succeeded with $\delta_a = 5$ and $f_{last} = 0.05$ (i.e., the default values), the recovery of $f_2$ used $\delta_a = 18$ and $f_{last} = 0.1$, and we found that $\delta_a \lesssim 10$ for this case led to highly truncated results and unstable GPR training. This is easy to understand in retrospect from studying the shapes of the $2\sigma$ and $3\sigma$ contours in the right panel of Fig. 6, which shows a highly non-Gaussian posterior. We emphasize that this choice of last iterations can be made after the basic ASA optimization is complete, so it does not waste any evaluations. We also see that the $3\sigma$ contours (particularly in the $a_1$-$a_2$ plane) are now noisier than in the other examples, which explains our choice of a more stringent MCMC convergence criterion. For the specific runs shown, PICASA with $N_{samp} = 30$ (100) led to recovery using $N_{eval} = 880$ (5884) evaluations, a gain of a factor of $\sim 65$ ($\sim 35$) over the standard MCMC in recovering $f_1$ ($f_2$). For the GPR training for $f_1$ ($f_2$), we set the robust_convergence key to False (True), the cv_thresh key to 0.005 (0.05) and the kernel key to ‘rbf’ (‘rq’) (see section 3.2 for details). The GPR training then succeeded with 296 (2602) samples.

We also performed a statistical analysis of $N_{eval}$ for each of the models, comparing the standard MCMC with PICASA. For $f_1$, the median and central 68% range of the distribution of $N_{eval}$ for the standard MCMC, demanding $R - 1 < 0.002$, was $N_{eval} \approx (32.2^{+11.5}_{-8.8}) \times 10^3$, while for PICASA with the default $\delta_a$ and $f_{last}$ and $N_{samp} = 30$, it was $N_{eval} \approx 770^{+206}_{-120}$ with an optimization failure rate of 1%. Thus, PICASA leads to typical gains of factors of $\sim 40$ as compared to the standard MCMC in this case. For $f_2$, the standard MCMC demanding $R - 1 < 0.001$ used $N_{eval} \approx (104.4^{+66.6}_{-27.2}) \times 10^3$. With the $\delta_a$ and $f_{last}$ choices mentioned earlier, PICASA with $N_{samp} = 30$ led to $N_{eval} \approx 2460^{+984}_{-236}$ with an optimization failure rate of 5%, while increasing $N_{samp}$ to 100 (the displayed example) gave $N_{eval} \approx 6272^{+856}_{-645}$ and decreased the failure rate to 1%. Thus, in this case, gains of factors of 15-40 are typical.

### 4.3 Relative performance of ASA and GPR training

Although PICASA clearly outperforms a standard MCMC in terms of total number of cost function evaluations, in order for it to be a practical alternative to the latter, the computational expense of the GPR training step must also be understood. The main bottleneck in GPR training is the inversion of a $N_{train} \times N_{train}$ matrix, where $N_{train}$ is size of the training set. Our iterative training procedure slowly increases $N_{train}$ until convergence is achieved, making the total computational cost of GPR training highly problem-specific. In general, the GPR for likelihoods with features will of course be harder to train than for smooth likelihoods.

The examples discussed above give us some idea of possible variations. We saw that the GPR for all but the non-linear $f_2$ model was successfully trained with a few hundred training samples, while $f_2$ used $\sim 2600$ samples. In terms of total wall time used by the iterative GPR training and subsequent MCMC, using 6 processors on a laptop, all the examples except $f_2$ completed in $\lesssim 3$ minutes (in some cases, less than a minute), while $f_2$ required $\sim 27$ minutes. For these simple examples, the ASA optimization step was dramatically faster in terms of wall time, simply because the cost functions we used could be calculated extremely fast. In fact, in almost all the above cases, the standard MCMC outperformed PICASA in terms of wall time (especially for the non-linear $f_2$ example). The true utility of PICASA is expected to emerge in problems where the cost functions are much more expensive to calculate.
where each cost function evaluation takes, say, tens of seconds or more on a single processor.\textsuperscript{5}

We chose to display \(3\sigma\) contours in the examples above, so as to assess the performance of \textsc{picasa} under stringent requirements of accurately reproducing the outer tails of the posterior distribution. If one is interested in at most \(2\sigma\) contours, \textsc{picasa} can be run with smaller values of \(\delta_{\text{e}}\) (data pack key \texttt{dsig}), leading to even fewer evaluations. Of course, in this case the standard MCMC would also require fewer evaluations, but we have checked that the relative improvements seen above do not change significantly. Finally, all our examples used diagonal covariance matrices for the data errors, so it is also interesting to ask whether correlated data errors have any impact on the performance of \textsc{picasa}. Using some simple forms for positive-definite non-diagonal covariance matrices \(C_{\text{data}}\) (e.g., Musso & Sheth 2014) to define data errors in the examples above, we have checked that \(N_{\text{eval}}\) between \textsc{picasa} and standard MCMC are essentially unchanged.

These examples demonstrate the basic behaviour of \textsc{picasa} in a variety of situations with different levels of non-linearity and dimensionality of parameter space. We discuss a few caveats and avenues for improvement in section 5.

5 CONCLUSION

We have presented a new technique for efficient parameter exploration, dubbed Anisotropic Simulated Annealing (ASA), particularly aimed at computationally expensive likelihoods. ASA builds on the ideas underlying simulated annealing and iteratively samples smaller and smaller regions of parameter space using Latin hypercubes, zooming in on the minimum of a pre-defined cost function. In addition to efficient recovery of the location of this minimum (i.e., the best-fit parameters), the resulting sample can be interpolated using Gaussian Process Regression to subsequently perform a standard MCMC algorithm with inexpensive sampling. We have combined these features into the Python code \textsc{picasa} (Parameter Inference with Anisotropic Simulated Annealing), with the integration with MCMC implemented using the \textsc{cobaya} framework (Torrado & Lewis 2019, 2021).

Through a number of linear and non-linear examples, we demonstrated that \textsc{picasa} can easily achieve effective gains in the number of likelihood evaluations of factors of \(\gtrsim 10\) to \(\gtrsim 100\), depending on the specific problem, in comparison to a standard MCMC approach. The code \textsc{picasa} is being actively developed and currently has several aspects which can be improved further. These include:

(i) Handling situations where the best-fit lies close to a sharp prior (in other words, when the data provide a limit rather than a constraint on a parameter). Currently, the ASA optimization will find it troublesome to settle onto an acceptable fit in such cases, and the GPR training will also likely not converge well. It should be possible to address this issue by allowing for the likelihood to be evaluated (e.g., by analytic continuation) in undesirable regions of parameter space, and then separating the imposition of the prior from the likelihood evaluator, just like in standard MCMC implementations.

(ii) The efficiency and robustness of the GPR training can likely be enhanced by allowing for dynamic assessment of the best choice of kernel (akin to a low-level neural network), as well as automated and robust assessment of the need for additional \(\chi^2\) samples. At present, the GPR training described in section 3.2 can require some level of user intervention, which is also not desirable.

(iii) Handling multiple (nearly) degenerate minima of the cost function. At present, \textsc{picasa} is guaranteed to remain stuck in one such minimum. It should be possible to allow for exploration of degenerate minima using, e.g., known or suspected symmetries of the cost function.

(iv) Handling non-Gaussian errors defining the log-likelihood. At present, \textsc{picasa} interprets the value of the cost function \(\chi^2(a)\) assuming Gaussian errors on the observable data. This can be a problem when the likelihood is substantially non-Gaussian in the data (e.g., when analysing galaxy cluster counts for cosmological inference). Since the goodness-of-fit is only used to assess whether or not an acceptable minimum is being reached, at present we recommend shifting the definition of \(\chi^2\) in such cases by an additive constant, to make its expected value near the best fit close to the number of degrees of freedom. It should, however, also be possible to modify this condition by allowing for alternate (more Bayesian, say) criteria to assess the quality of the fit.

(v) Combining with approximate likelihood schemes. At present, \textsc{picasa} places the onus of accurate likelihood evaluation on the user. It will be interesting to explore whether approximation schemes such as \textsc{dali} (Sellentin et al. 2014) can be integrated into \textsc{picasa} for increased robustness and speed, especially for estimating the best fit, GPR training and extensions to non-Gaussian data errors.

These and other improvements will continue to be updated on the public repository for \textsc{picasa} listed under data availability below. We expect \textsc{picasa} to be useful in a number of situations requiring numerically expensive likelihood calls. A concrete example could be the reconstruction of the baryon acoustic oscillation (BAO) feature as discussed by Nikakhht et al. (2022). More generally, since \textsc{picasa} is agnostic to the nature of the observable, the list of these potential applications can span numerous topics in cosmology, extra-Galactic astrophysics, gravitational waves science, exoplanetary science, and so on. We will explore some of these applications in future work.

ACKNOWLEDGMENTS

\textit{It is a pleasure to thank Tirthankar Roy Choudhury for encouragement and invaluable input in developing \textsc{picasa} functionality, and both him and Barun Maity for stress-testing \textsc{picasa}. I am also grateful to Ravi Sheth and Shadlab Alam for collaboration on a project that used an early version of the ASA algorithm, as well as comments on an earlier draft.}

\textit{MNRAS 000, 1–14 (0000)}
Finally, I thank Dipankar Bhattacharya for discussions regarding simulated annealing which originally sparked the ideas underlying this work. This work made extensive use of the open source computing packages NumPy (Van Der Walt et al. 2011),6 Scipy (Virtanen et al. 2020),7 Scikit-Learn (Pedregosa et al. 2011),8 Matplotlib (Hunter 2007)9 and Jupyter Notebook.10

DATA AVAILABILITY

The code picasa is publicly available at https://bitbucket.org/aparanjape/picasa/. The COBAYA framework for parameter inference is publicly available at https://cobaya.readthedocs.io/.

REFERENCES

Abazajian K. N., et al., 2016, arXiv e-prints, p. arXiv:1610.02743
Abbott B. P., et al., 2019, Physical Review X, 9, 031040
Abbott T. M. C., et al., 2022, Physical Rev. D, 105, 023520
Agrawal S., Davé R., Bassett B. A., 2018, MNRAS, 478, 3410
Alam S., Peacock J. A., Farrow D. J., Loveday J., Hopkins A. M., 2021a, MNRAS, 503, 59
Alam S., et al., 2021b, MNRAS, 504, 4667
Baker J., et al., 2019, arXiv e-prints, p. arXiv:1907.06482
Bécsy B., Cornish N. J., Digman M. C., 2022, arXiv e-prints, p. arXiv:2204.07160
Borucki W. J., 2016, Reports on Progress in Physics, 79, 036901
Choudhury T. R., Paranjape A., 2018, MNRAS, 481, 3821
Cranmer S., Tomaso D., Rein H., Battaglia P., Hadden S., Armitage P. J., Ho S., Spergel D. N., 2021, Proceedings of the National Academy of Science, 118, 202032118
DESI Collaboration et al., 2016, arXiv e-prints, p. arXiv:1611.00036
Dai B., Seljak U., 2019, Cobaya: Bayesian analysis in cosmology, Astrophysics Source Code Library, record ascl:1910.019
Delgado A. M., Wadekar D., Hadzhiyska B., Bose S., Hernquist L., Ho S., 2021, arXiv e-prints, p. arXiv:2111.02422
Dueck G., Scheuer T., 1990, Journal of Computational Physics, 90, 161
Feng Y., Chu M.-Y., Seljak U., McDonald P., 2016, MNRAS, 463, 2273
Foreman-Mackey D., Hogg D. W., Lang D., Goodman J., 2013, PASP, 125, 306
Guo Q., et al., 2011, MNRAS, 413, 101
Hastings W. K., 1970, Biometrika, 57, 97
Heitmann K., Lawrence E., Kwan J., Habib S., Higdon D., 2014, ApJ, 780, 111
Heitmann K., et al., 2016, ApJ, 820, 108
Henriques B. M. B., White S. D. M., Thomas P. A., Angulo R., Tassev S., Zaldarriaga M., Eisenstein D. J., 2013, J. Cosmology Astropart. Phys., 2013, 036
Hobbs G., et al., 2010, Classical and Quantum Gravity, 27, 084013
Howlett C., Manera M., Percival W. J., 2015, Astronomy and Computing, 12, 109
Hunter J. D., 2007, Computing In Science & Engineering, 9, 90
Karamanis M., Beutler F., 2020, arXiv e-prints, p. arXiv:2002.06212
Kirkpatrick S., Gelatt C. D., Vecchi M. P., 1983, Science, 220, 671
Kitaura F.-S., Heß S., 2013, MNRAS, 435, L78
Kitaura F.-S., et al., 2016, MNRAS, 456, 4156
Knabe et al., 2019, MNRAS, 484, 5509
Konmatsu E., et al., 2011, ApJS, 192, 18
Koopmans L., et al., 2015, in Advancing Astrophysics with the Square Kilometre Array (AASKA14). p. 1 (arXiv:1505.07568), doi:10.23231/1.215.0001
LSST Science Collaboration et al., 2009, arXiv e-prints, p. arXiv:0912.0201
Laureijs R., et al., 2011, arXiv e-prints, p. arXiv:1110.3193
Lawrence E., et al., 2017, ApJ, 847, 50
Lewis A., 2013, Phys. Rev. D, 87, 103520
Lucie-Smith L., Peiris H. V., Pontzen A., Lochner M., 2018, MNRAS, 479, 3405
Merloni A., et al., 2012, arXiv e-prints, p. arXiv:1209.3114
Mesinger A., Furlanetto S., Cen R., 2011, MNRAS, 411, 955
Metropolis N., Rosenbluth A. W., Rosenbluth M. N., Teller A. H., Teller E., 1953, J. Chem. Phys., 21, 1087
Monaco P., Theuns T., Taffoni G., 2002, MNRAS, 331, 587
Monaco P., Sefusatti E., Borgani S., Crocce M., Fosalba P., Sheth R. K., Theuns T., 2013, MNRAS, 433, 2389
Moscati P., Fontanari J. F., 1990, Physics Letters A, 146, 204
Musso M., Sheth R. K., 2014, MNRAS, 443, 1601
Nikakhah F., Sheth R. K., Zehavi I., 2022, Phys. Rev. D, 105, 043536
Pedregosa F., et al., 2011, Journal of Machine Learning Research, 12, 2825
Planck Collaboration et al., 2018, arXiv e-prints, p. arXiv:1807.06209
Press W. H., Flannery B. P., Teukolsky S. A., 1986, Numerical recipes. The art of scientific computing. Cambridge University Press, Cambridge, UK
Rasmussen C. E., Williams C. K. I., 2005, Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning). The MIT Press, Boston, USA
Ricker G. R., et al., 2014, in Oschmann Jacobus M. J., Clampin M., Fazio G. G., MacEwen H. A., eds, Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series Vol. 9143, Space Telescopes and Instrumentation 2014: Optical, Infrared, and Millimeter Wave. p. 914320 (arXiv:1406.0151), doi:10.1117/12.2063489
Rizzato M., Sellentin E., 2022, arXiv e-prints, p. arXiv:2203.05009
Schauerder D., Li Y., Trinker J., Ho S., Refregier A., 2021, arXiv e-prints, p. arXiv:2111.06393
Schmidt S., et al., 2021, Phys. Rev. D, 103, 043020
Scoccimarro R., Sheth R. K., 2002, MNRAS, 329, 629
Sellentin E., Quartin M., Amendola L., 2014, MNRAS, 441, 1831
Tassev S., Zaldarriaga M., Eisenstein D. J., 2013, J. Cosmology Astropart. Phys., 2013, 036
Torrado J., Lewis A., 2019, Cobaya: Bayesian analysis in cosmology, Astrophysics Source Code Library, record ascl:1910.019 (ascl:1910.019)
Torrado J., Lewis A., 2021, J. Cosmology Astropart. Phys., 2021, 057
Van Der Walt S., Colbert S. C., Varoquaux G., 2011, preprint, (arXiv:1102.1523)
Villaescusa-Navarro F., et al., 2021, ApJ, 915, 71
Virtanen P., et al., 2020, Nature Methods, 17, 261
Wang Z., Liu C., Zhao J., Shao L., 2022, arXiv e-prints, p. arXiv:2203.02670
Yang X., et al., 2018, ApJ, 860, 30
Yuan S., Garrison L. H., Eisenstein D. J., Wechsler R. H., 2022a, arXiv e-prints, p. arXiv:2203.11963
Yuan S., Garrison L. H., Hadzhiyska B., Bose S., Eisenstein D. J., 2022b, MNRAS, 510, 3301
eBOSS Collaboration et al., 2020, arXiv e-prints, p. arXiv:2007.08991

6 http://www.numpy.org
7 http://www.scipy.org
8 https://scikit-learn.org/
9 https://matplotlib.org/
10 https://jupyter.org