Interval Methods For Improving The Efficiency Of Monte Carlo*

Richard G. Everitt
University of Warwick, Coventry, UK
richard.g.everitt@gmail.com

Abstract

The past thirty years have seen an explosion in the use of Monte Carlo methods for statistical inference, with an important advance being the use of population Monte Carlo (PMC) methods, which use of a population for exploring the search space. Despite their success in statistics, PMC methods have received less attention in the field of optimization. This paper begins by introducing population Markov chain Monte Carlo (MCMC) [10, 20]; and sequential Monte Carlo (SMC) [6, 12], then describes the use of interval arithmetic to aid these algorithms. In this paper, through showing how interval arithmetic may be used within PMC algorithms, we obtain methods that combine the useful properties of both PMC and internal arithmetic.

Keywords: interval analysis, Monte Carlo
AMS subject classifications: 65C05, 65G40

1 Introduction

This paper is concerned with *sampling* and *global optimization*. The former task is to simulate points from a distribution π on space $\mathcal{X}$; the latter is the task of finding the point in a domain $\mathcal{X}$ at which some *objective function* $f : \mathcal{X} \to \mathbb{R}$ obtains its maximum $\max_{x \in \mathcal{X}} f(x)$. These problems are widely encountered in numerous applications in science and engineering and many different methodologies have been proposed for solving them. Our focus in this paper is on combining two quite different types of technique: *population Monte Carlo* (PMC) and *interval methods*. The term PMC encompasses several different ideas, each of which uses a population of points to explore the distribution or objective function. The use of interval methods also refers to a variety of algorithms, their common characteristic being that the exploration of the distribution or objective function is guided by information obtained through the use of *interval arithmetic* [22, 23].

PMC has a number of desirable characteristics in common with the widely used techniques that can be termed *evolutionary algorithms*. The use of a population of

*Submitted: July 18, 2021; Revised: October 28, 2021; Accepted: October 28, 2021.

points often ensures that PMC can often avoid getting stuck in a local, rather than the desired global, optimum. However, there remain cases where PMC may not find the global optimum when only a fixed computational budget is available: in particular for objective functions where the global optimum is in a peak of the objective function that has small support and is well separated from other peaks of the function. It will be seen that this is precisely the type of function that is handled efficiently by interval methods.

Interval methods are usually motivated by a desire for guaranteed bounds or guaranteed computations, but this is not the motivation for their use in this paper. Instead, we aim to exploit some properties of interval arithmetic that have not previously been used in Monte Carlo methods. Namely, we aim to exploit the property that it is possible to find information about the range of a function on an interval solely through evaluating the interval extension of the function, which has a computational cost of the same order as evaluating the function itself. There is something of an analogy here with research that makes use of derivatives of a function to improve the efficiency of Monte Carlo methods. In this case the derivative function provided by automatic differentiation, analogous to the interval extension, provides information that is not directly available in the original function.

This paper illustrates Monte Carlo approaches that can use any method for approximating the range of a function over an interval. This is done through introducing the concept of a "dilation" of a function, and showing how a dilation may be used in Monte Carlo methods. The role of interval methods is to provide a computationally cheap method for approximating the dilation. The interval techniques used in this paper are the naive approaches described in Moore [23], however we will see that even these naive methods can yield improvements to the performance of Monte Carlo methods on some classes of functions.

Readers of this journal may not be familiar with the literature on Monte Carlo methods. Therefore, the remainder of this section gives the required background on each of the techniques used in the paper: beginning a description of Monte Carlo methods and *simulated annealing* (SA) in section 1.1.3, before describing two PMC methods in section 1.2. A reader familiar with this background material may wish to start reading from section 2 which describes the main contribution of the paper, showing how interval arithmetic may be used within PMC algorithms. Section 3 demonstrates the performance of these new techniques, before we discuss the implications of this work in section 5.

1.1 Simulation Algorithms And Simulated Annealing

Our starting point is a short description of the core Monte Carlo methods *importance sampling* (IS), Markov chain Monte Carlo (MCMC) and SA, since these ideas lie at the core of the paper. Suppose we wish to calculate the expectation $\mathbb{E}_\pi[X] = \int_x x\pi(dx)$ of a random vector X under a distribution $\pi(X)$ defined on space E. Often such an integral is intractable, so the Monte Carlo approximation

$$\widehat{\mathbb{E}}_\pi[X] = \frac{1}{N}\sum_{i=1}^{N} x^{(i)} \tag{1}$$

is used, where each $x^{(i)}$ is a point simulated from π. Now, π is often non-trivial to simulate from directly, so a Monte Carlo method is used as a substitute. The main reason for the popularity of Monte Carlo is that the error of the estimator scales

as $1/\sqrt{N}$, without the exponential dependence on the dimension d of X found in numerical integration.

We outline IS in section 1.1.1 and MCMC in section 1.1.2 below, in advance of a brief description of SA in section 1.1.3.

1.1.1 Importance Sampling

IS avoids the problem of simulating from π by instead simulating points from a *proposal* q. The Monte Carlo approximation analogous to (1) is then

$$\widehat{\mathbb{E}}_\pi[X]_{\text{imp}} = \frac{1}{N}\sum_{i=1}^{N} x^{(i)} w(x^{(i)}), \tag{2}$$

where the weight $w(x^{(i)}) := \pi(x^{(i)})/q(x^{(i)})$ (note that these weights must be normalised if π is only known up to a normalising constant). The behaviour of this estimator is closely tied to the choice of q which should be chosen to be as close to π as possible. Loosely speaking, it is crucial that q should allow an effective exploration of π (it should have mass in all of the regions where π has appreciable mass), but it is important for computational reasons that as few points as possible are drawn in regions where π has little mass. Clearly it is difficult to choose an appropriate q where π is a complicated distribution, especially in high dimensions. SMC, described in section 1.2.1, extends IS so that this problem is less pronounced.

1.1.2 Markov Chain Monte Carlo

MCMC takes a rather different approach to simulating from π: a Markov chain with transition kernel K is constructed such that it has invariant distribution π, then a realisation from this chain is used in a Monte Carlo estimator as if it is an independent sample from π. Although the sample is not independent, under weak and easy to verify conditions it can be shown that the law of large numbers holds in this setting, and that under some stronger conditions a Central Limit Theorem can hold (Roberts and Rosenthal [26] contains a thorough review of this material).

A simple, and widely used, method for constructing such a Markov chain for an arbitrary π is the Metropolis-Hastings (MH) algorithm, shown in algorithm 1. This algorithm was originally introduced using symmetric proposals in Metropolis et al [21] (this specific case is known as the Metropolis algorithm), then generalised to the given form in Hastings [13].

for $i = 1 : N$ **do**
  Simulate $x^* \sim q(.|x^{(i-1)})$;
  Simulate $u \sim \mathcal{U}[0,1]$;
  if $u < \min\left\{1, \frac{\pi(x^*)q(x^{(i-1)}|x^*)}{\pi(x^{(i-1)})q(x^*|x^{(i-1)})}\right\}$ **then**
    $x^{(i)} = x^*$;
  else
    $x^{(i)} = x^{(i-1)}$;
  end if
end for

Algorithm 1: The Metropolis-Hastings algorithm.

The proposal q in an MCMC algorithm plays a different role to that in importance sampling: it is the mechanism by which the chain moves around the space. However, again it is instrumental in determining the statistical properties of Monte Carlo estimators based on the method. For MCMC a good choice of proposal is usually one that results in a realisation from the chain that is as close to an independent sample as possible. A useful property of the MH algorithm is that it is possible to use proposals that do not move in every dimension of $\mathcal{X}$ at once. This has the consequence that the problem of constructing a proposal in a high dimensional space (as is encountered in IS) is avoided: it is common to move only one of the d dimensions at a time, known as a *single component move*.

1.1.3 Simulated Annealing

Our focus now changes from simulation to global optimization. Kirkpatrick et al [18] constructed an algorithm for global optimization simply by changing the target density at every iteration of the Metropolis algorithm. Specifically we choose the target density at iteration i of the algorithm to be $\pi_i = \pi^{\tau_i}(x) := \exp(f(x)/\tau_i)$ where $\{\tau_i\}$ is a sequence of "temperatures" that decrease to zero. If we begin with a high temperature, the peaks of π^{τ_i} are less accentuated and the function is easier for the Metropolis algorithm to explore; then as the temperature decreases to zero eventually all of the mass is concentrated at the global maximum. There is a large literature on the convergence of simulated annealing (an overview of which can be found in Henderson and Johnson [14], for example).

1.2 Population Monte Carlo

This section describes two methods that may often offer improvements in performance. Both are extensions of Monte Carlo simulation methods outlined in the previous section that, with minor alterations, may be used for global optimization. The first, SMC (section 1.2.1), is an extension of IS, with the second, population MCMC (section 1.2.2), being an extension of standard MCMC. The motivation for both methods lies in the idea that using a population of points to explore a target distribution can be more effective than using a single point (as is the case in standard MCMC): the population contains information about the target that can be used to construct better proposals, and should also safeguard against getting trapped in a local optimum. This is essentially the idea exploited in the very broad class of evolutionary algorithms, which have the common characteristic of using an evolving population to explore an objective function. PMC algorithms may be viewed as a type of EA that is based on the well understood theory of Monte Carlo algorithms.

1.2.1 Sequential Monte Carlo Samplers

We begin this section by noting that IS has the desired property of using a population of points to explore a target. Suppose we consider extending IS so that there is a means of moving the population of points in order to explore the target distribution and reduce the importance of the initial proposal distribution q. Suppose also that we generalise this idea further: in using this extension to IS to simulate from a sequence of target distributions $\pi_{1:T} := (\pi_1, ..., \pi_T)$, where each π_t is defined on a space E, and the final target in the sequence π_T is the distribution of interest. If we choose the sequence of targets such that π_1 is easy to sample from and there is not too much

difference between one of the targets in the sequence π_t and the next π_{t+1}, then the targets give a "route" to move the particles easily from π_1 to π_T. This idea is known as sequential IS, and more precisely proceeds as follows.

To begin, use IS with proposal K_1 to draw P weighted points from π_1. The i'th point, $x_1^{(i)}$, has weight

$$w_1^{(i)} = \frac{\pi_1(x_1^{(i)})}{K_1(x_1^{(i)})}. \tag{3}$$

To sample from π_2, we apply importance sampling sequentially. We apply the kernel $K_2(x_1^{(i)}, .)$ to particle i, labelling the new position as $x_2^{(i)}$. We then reweight the i'th particle:

$$w_2^{(i)} = w_1^{(i)} \frac{\pi_2(x_2^{(i)})K_1(x_1^{(i)})}{\pi_1(x_1^{(i)}) \int_x K_1(x)K_2(x, x_2^{(i)})dx} \tag{4}$$

However, the integral in the denominator cannot be calculated in general. Therefore we cannot use the algorithm in exactly this form. In order to use the methodology with a sequence of targets $\pi_{1:N}$, Del Moral et al [6] construct an alternative artificial sequence of targets that means that the ideas of particle filtering (introduced in Gordon et al [12] as a method for using importance sampling sequentially to simulate from a time-indexed state space) can be applied as follows.

Specifically, the sequence of targets $\tilde{\pi}_{1:N}$ is used, where each $\tilde{\pi}_t$ is a function defined on $(E)^t$, and where the marginal of $\tilde{\pi}_t$ on the final tth dimension is π_t. Del Moral et al [6] use the following sequence:

$$\tilde{\pi}_t(x_{1:t}) = \pi_t(x_t) \prod_{j=2}^{t} L_{j-1}(x_j, x_{j-1}) \tag{5}$$

where L_{j-1} is a *backwards* Markov kernel.

Applying the sequential importance sampling idea described above to this new sequence of targets, we obtain the weight update at time t:

$$w_t^{(i)} = w_{t-1}^{(i)} \frac{\pi_t(x_t^{(i)})L_{t-1}(x_t^{(i)}, x_{t-1}^{(i)})}{\pi_{t-1}(x_{t-1}^{(i)})K_t(x_{t-1}^{(i)}, x_t^{(i)})}. \tag{6}$$

This method gives an extremely flexible framework for sampling from a general distribution. To use this framework in practice, we first pick a sequence of distributions $\pi_{1:T}$ where π_T is the target we are interested in, and such that the sequence gives a gradual transition from π_1 to π_T. We then define the kernels $K_{1:T}$ that should allow us to move effectively between the distributions. Much flexibility is allowed here, including single component moves. Finally we must choose the backward kernels $L_{1:T}$ in order to minimise the variance of the importance weights. The optimal backwards kernel in this sense cannot be calculated (it corresponds to the original sequential importance sampling framework). Advice on choosing the L kernel is contained in Del Moral et al [6].

Now suppose that the kernel K_t is chosen to be an MCMC kernel. In this case, as long as π_{t-1} is not too different from π_t a good approximation to the optimal L kernel is the natural reversal of the MCMC kernel:

$$L_{t-1}(x_t, x_{t-1}) = \frac{\pi_t(x_{t-1})K_t(x_{t-1}, x_t)}{\pi_t(x_t)} \tag{7}$$

which leads to the weight update:

$$w_t^{(i)} = w_{t-1}^{(i)} \frac{\pi_t(x_{t-1}^{(i)})}{\pi_{t-1}(x_{t-1}^{(i)})}. \tag{8}$$

By restricting ourselves to using MCMC kernels (and, in particular, Metropolis-Hastings kernels, as we use in the algorithm below) we obtain an algorithm that is very easy to apply. All that is needed is to choose a sequence of target distributions and proposal distributions for the Metropolis-Hastings moves between the targets. Note that we can perform as many iterations of the MCMC kernel as we want when moving from target π_{t-1} to π_t and still use the same weight update.

As described thus far, the resultant algorithm is simply IS, constructed sequentially on a space of increasing dimension. However, the variance of estimators of the type in equation 2 can be shown to increase exponentially with t (see Doucet and Johansen [8], for example), so this method is not of practical use. To remedy this problem, Gordon et al [12] introduced the idea of *resampling* at each SMC iteration: to, at iteration t, sample N new particles independently from the empirical distribution given by $\left\{x_{t-1}^{(i)}\right\}_{i=1}^{N}$, the current particles and their normalised weights $\left\{\tilde{w}_t^{(i)}\right\}_{i=1}^{N}$ that is

$$\sum_{i=1}^{N} \tilde{w}_t^{(i)} \delta_{x_{t-1}^{(i)}},$$

with $\delta_{x_{t-1}^{(i)}}$ being the Dirac delta mass at $x_{t-1}^{(i)}$. This use of sequential importance sampling together with resampling is known as an *SMC sampler*.

Under weak integrability conditions, estimates from both IS and SMC obey a Central Limit Theorem. Del Moral et al [6], Doucet and Johansen [8] compare the asymptotic variances in the two cases, and observe that SMC does not suffer the exponential growth of the variance of IS. Empirically it is observed that the variance of estimators from SMC can be orders of magnitude smaller than those from IS.

The algorithm that results from making these choices is given in algorithm 2. In this algorithm we have also introduced the additional detail that resampling is not performed at every step: each time resampling is performed, additional noise is introduced into the particle system, thus often better performance is attained if it is only performed when required. Instead a criterion that measures the discrepancy between neighbouring targets is used to decide when to resample. Here we propose to resample when the commonly used *effective sample size* falls below some fraction c of the total number of particles.

A related technique is the *cross-entropy method* (e.g. Botev et al [5]), which Schafer [29] shows corresponds to a particular configuration of the general SMC sampler framework.

1.2.2 Population MCMC

The basic idea of population-based MCMC is to run several MCMC samplers in parallel and to exchange information between these samplers. This was first mentioned in Geyer [10] and subsequently developed by Liang and Wong [20] and Laskey and Myers [19], inspired by genetic algorithms. Subsequent developments of the idea are contained in, for example, Jasra et al [15], Strens [31], Strens et al [32]. Jasra et al [15] contains a particularly clear and simple explanation of population MCMC, and we take the algorithms in this section from that paper.

for $i = 1 : N$ **do**
  Simulate $x_1^{(i)} \sim q_1(.)$;
  Find the weight $w_1^{(i)} = \pi_1(x_1^{(i)})/q_1(x_1^{(i)})$;
end for
Normalise $\{w_1^{(i)}\}_{i=1}^N$ to give normalised weights $\{\tilde{w}_1^{(i)}\}_{i=1}^N$.
for $t = 2 : T$ **do**
  for $i = 1 : N$ **do**
    $w_t^{(i)} = \tilde{w}_{t-1}^{(i)} \pi_t(x_{t-1}^{(i)})/\pi_{t-1}(x_{t-1}^{(i)})$;
  end for
  Normalise $\{w_t^{(i)}\}_{i=1}^N$ to give normalised weights $\{\tilde{w}_t^{(i)}\}_{i=1}^N$.
  $\text{ESS} = \left(\sum_{i=1}^N [(\tilde{w}_{t-1}^{(i)})^2]\right)^{-1}$;
  if $\text{ESS} < cN$ **then**
    Resample $\{(x_{t-1}^{(i)}, \tilde{w}_t^{(i)})\}_{i=1}^N$;
  end if
  for $i = 1 : N$ **do**
    Simulate $x^* \sim q_t(.|x_{t-1}^{(i)})$;
    Simulate $u \sim \mathcal{U}[0, 1]$;
    if $u < \min\left\{1, \frac{\pi_t(x^*) q_t(x_{t-1}^{(i)}|x^*)}{\pi_t(x_{t-1}^{(i)}) q_t(x^*|x_{t-1}^{(i)})}\right\}$ **then**
      $x_t^{(i)} = x^*$;
    else
      $x_t^{(i)} = x_t^{(i-1)}$;
    end if
  end for
end for

Algorithm 2: An SMC sampler using Metropolis-Hastings kernel.

The fundamental framework is as follows. Suppose that our target density is π on the domain E. Then we define a new target on $(E)^M$:

$$\pi^*(X_{1:M}) = \prod_{j=1}^{M} \pi_j(X_j), \tag{9}$$

where at least one of the π_j is equal to the true target π. The π_j may then be chosen such that simulations from them may be useful in making moves on the true target. Geyer [10] suggests tempered versions of π: $\pi_j(x) = (\pi(x))^{1/\tau_j}$ where $\tau_j \geq 1$: this is also a common sequence of functions to use within SMC, and this sequence will be used extensively in the paper.

A population MCMC algorithm is simply an MCMC algorithm defined on π^*. Liang and Wong [20], Strens et al [32] and Jasra et al [15] give several examples of MCMC moves that may be useful on π^*. For simplicity, in this paper we use only two types of move. The first of these is a "mutation", where one of the M chains is chosen randomly, then updated with an MH move (as in algorithm 1). The other is to swap information between two chains chosen at random. In the most extreme case, this takes the form of an "exchange" move, where at iteration i the current value $x_a^{(i)}$ from one chain is exchanged with the value $x_b^{(i)}$ from another chain. The acceptance probability for this move $(x_a^*, x_b^*) = \left(x_b^{(i)}, x_a^{(i)}\right)$ is

$$A := \min\left\{1, \frac{\pi_a(x_a^*)\pi_b(x_b^*)}{\pi_a(x_a^{(i)})\pi_b(x_b^{(i)})}\right\}. \tag{10}$$

More generally, it is possible to propose a swap between a subset of the dimensions of $x_a^{(i)}$ and $x_b^{(i)}$, with the acceptance probability of such a move also being equal to A as long as the dimensions are chosen uniformly at random.

1.2.3 PMC And Optimization

As we have seen, there are numerous examples of using ideas from the optimization literature within simulation algorithms (mostly ideas from genetic algorithms). There has also been some work that uses simulation techniques for optimization (simulated annealing being the most prominent), but many of the most recent ideas in this field have not been used in optimization problems. In particular there is relatively little work on using PMC despite this only requiring simple modifications to the simulation algorithms described above (analogous to using SA compared to standard MCMC).

To use an SMC sampler for optimization, the only modification that is required is that instead of using a sequence of targets $\pi_{1:T}$ with π_T being the target of interest, we choose a sequence of targets such that they converge to the distribution where all of the mass is at the global maximum (e.g. use an annealing scheme that finishes with a temperature of zero). An obvious example of this is to use the same sequence of targets as is used in SA (as in Peters [25]). This choice will result in the population of points being concentrated at the global maximum, with the proof of convergence being exactly the same as those in the simulation case. There are a few examples of using SMC for optimization, including Johansen et al [17], Peters [25], Schafer [29].

Population MCMC is sometimes used for optimization in the form of *parallel tempering*. In this configuration, the M targets are chosen in the style of the sequence of targets chosen in SA, such that some of them are at a very low temperature where the global maximum obtains most of the mass, and so that there is a sequence of targets

at higher temperatures that will help the MCMC algorithm to explore the objective function to find this mass. In this paper, we will choose the M targets to take a different form that will help the exploration of the objective function, and instead choose to use the annealing idea on the composite target π^*. This algorithm is simply SA on this extended space, and thus inherits the convergence results from this literature.

We will use both SMC and population MCMC later in the paper, where the precise configuration of the these algorithms we use will be described. The next section focuses on interval methods: the means by which we will help both SMC and population MCMC to perform more efficiently.

2 Interval Arithmetic Within Population Monte Carlo

2.1 Combining Monte Carlo And Interval Methods

This paper looks to use information contained in the interval extension of a function to aid Monte Carlo algorithms. The key requirement of such an approach is that this is done without subdividing the domain in the manner of numerical integration (as is performed in branch-and-bound optimization for example), to avoid the curse of dimensionality. This distinguishes the methods in this paper from previous work [9, 27, 28] that made use of interval methods to construct proposals for rejection and importance samplers. which do rely on subdividing a domain and hence suffer a curse of dimensionality. Instead, our new approaches inherit their dependence on dimension from the Monte Carlo methods in which the interval techniques are embedded.

The following section outlines: in section 2.2, the idea of *dilated* objective functions, the way in which we make use of interval methods; in section 2.3, situations in which dilated objective functions might provide advantages over standard annealing approaches; and in section 2.4 the use of dilated objective functions in PMC. A short review of the fundamentals of interval methods is included in appendix A.

2.2 Dilated Objective Functions

This section outlines the concept of *dilating* an objective function. Dilation is a concept we borrow from mathematical morphology (e.g. Glasbey and Horgan [11]) that has the effect of widening the peaks in an objective function, regardless of their original width. When the idea of dilation is used in mathematical morphology it is usually applied to functions that have a discrete domain. In this case computation of the dilation of the function at any given point is tractable. However, our target density is usually defined on a complete space, which makes direct approaches to computing the dilation intractable. This is where interval methods can be used: when our target density is an elementary function, we can use an interval or affine extension of f to find an approximation to the dilation. The quality of this approximation depends on the over-enclosure of f by its interval extension.

In section 2.2.1 and 2.2.2 we describe dilation, and how to use interval arithmetic to use the idea in practice; in section 2.4 we describe how it may be used in PMC.

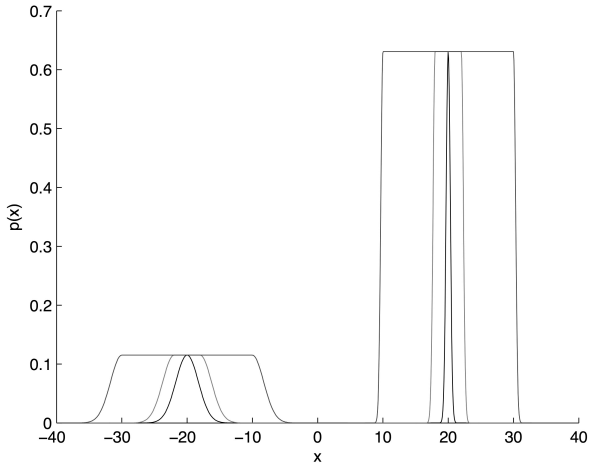


Figure 1: A mixture of normal densities (narrowest), and examples of dilations by radius $\epsilon = 2$ and $\epsilon = 10$ (widest).

2.2.1 Dilation Of A Function By A Set

We define the dilation of a function $f : D \to \mathbb{R}$ for a domain $D \subseteq \mathbb{R}^d$ by a set $B \subseteq \mathbb{R}^d$ as follows. The dilated function $(f \oplus B) : D \to \mathbb{R}$ is

$$(f \oplus B)(x) := \sup_{y \in B_x} f(y), \tag{11}$$

where $B_x = \{b + x | b \in B\} \cap D$. We restrict our attention to dilation of a function by a box B_ϵ in the set

$$\mathcal{B} = \{B_\epsilon = ([-\epsilon, \epsilon] \times ... \times [-\epsilon, \epsilon]) \in \mathbb{IR}^d | \epsilon \in \mathbb{R}_{\geq 0}\} \tag{12}$$

We use f_ϵ to denote the dilation of the function $f : \mathbb{R}^d \to \mathbb{R}$ by the box $B_\epsilon \in \mathbb{B}$ and say that f_ϵ is the dilation of f by radius ϵ. To see the effect of such a dilation consider the mixture of normal densities $f = 0.5\mathcal{N}(-20, 3) + 0.5\mathcal{N}(20, 0.1)$. Figure 1 shows f, along with its dilations f_2 and f_{10}. The effect of dilation is to widen peaks by 2ϵ, regardless of their original width. For multi-dimensional targets, the dilation widens peaks in the directions parallel to the axes.

We propose to use dilated functions as an alternative set of auxiliary targets in PMC: we use a set of dilations of the true target, with radii that begin at some maximum $\epsilon_{\max}$ and go through to 0 (to give the true target). Analogous to the use of tempered targets, the most dilated target will be easier to sample from than the true target (in particular, it is easier to locate peaks in these targets), and this set of functions will provide a route for moving points drawn from the most dilated target to the true target.

2.2.2 Interval Dilation

Although the definition of the dilation of an arbitrary function f is simple, it is difficult to compute at any given point in the domain of f, except in some special cases (e.g.

when the set by which we dilate is finite). In order to use dilation on continuous domains we need an alternative way of calculating it. Suppose we restrict f to being an elementary function with $f : \mathbb{R}^n \to \mathbb{R}$ and that we only consider dilations by boxes of radius ϵ. In this case we may find the dilation approximately using interval analysis. Let F be the natural interval extension of a representation of f (defined in the appendix). We define the *interval dilation* of f (with respect to the representation for f) to be

$$f_\epsilon(x) = \max\{F(B_{x+\epsilon})\} \tag{13}$$

where $x \in \mathbb{R}^n$ and $B_{x+\epsilon} = \{b + x | b \in B_\epsilon\}$ with $B_\epsilon \in \mathcal{B}$. An evaluation of an interval dilation of f has at most four times the computational cost of an evaluation of f, which makes it feasible to use interval dilations as auxiliary functions in PMC. From hereon we use the notation f_ϵ to represent an interval dilation by radius ϵ, rather than the true dilation as in expression (11). When the natural interval extension of f satisfies range equivalence, the interval dilation is exactly the same as the true dilation. This can be achieved if we can find a representation of f in which each input variable only appears once. However, in many cases such a representation does not exist and we can only find natural interval extensions that satisfy range enclosure. In this case, for small dilations we can have some confidence that the interval dilation will be similar to the true dilation, but the error increases as the size of the dilation grows.

2.3 Comparing Dilation And Tempering

We intend to make use of both tempering (with high temperatures, $t_i > 1$) and dilation for exploring multi-modal objective functions, but note that both ideas are designed to make peaks wider, thus easier to discover. In this section we provide an illustration of situations in which dilation is particularly useful: when the global maximum lies at the top of a very narrow peak.

As an example we consider a target density (for our Monte Carlo optimization algorithms) that is a mixture of k univariate Gaussians:

$$\pi(x) = \sum_{i=1}^{k} \frac{w_i}{\sigma_i\sqrt{2\pi}} \exp\left(\frac{-(x-\mu_i)^2}{2\sigma_i^2}\right), \tag{14}$$

where the parameters μ_i and σ_i are such that we can consider the overlap between the components as being negligible (in everything that follows we will assume that there is no overlap between the components). We look at the effects of tempering and dilation on the effective heights, weights and widths (standard deviations) of the components: the weight of component i is w_i, the height, $w_i/(\sigma_i\sqrt{2\pi})$ and the width, σ_i. We choose this example since the analysis is relatively simple and we can have some confidence that the results are more widely relevant.

2.3.1 Effect Of Tempering

As previously, we define a tempered version π^τ of the target π, for some real "temperature" $\tau > 0$, as $\pi^\tau(x) = (1/Z_\tau)\pi(x)^{1/\tau}$ where Z_τ is a normalising constant of the density.

Consider the tempered version of a single weighted component p_i^τ (we can consider the effect component-wise, since the overlap between the components is negligible):

$$p_i^\tau(x) = \frac{1}{Z_\tau}\left(\frac{w_i}{\sigma_i\sqrt{2\pi}}\right)^{(1/\tau)} \exp\left(\frac{-(x-\mu_i)^2}{2\tau\sigma_i^2}\right). \tag{15}$$

The effective height is $\frac{1}{Z_\tau}(w_i/(\sigma_i\sqrt{2\pi}))^{1/\tau}$ and the effective width $\sigma_i\sqrt{\tau}$. Thus we see the expected behaviour that as the temperature increases, each component gets wider and the heights of the components tend towards having the same value.

The unnormalised effective weight $w_i^{(\tau)}$ can be easily computed using

$$\begin{aligned} w_i^{(\tau)} &= \frac{1}{\sqrt{(2\pi)}^{1/\tau}} \frac{w_i^{1/\tau}}{\sigma_i^{1/\tau}} \int \exp\left(\frac{-(x-\mu_i)^2}{2\tau\sigma_i^2}\right) dx & (16) \\ &= \frac{1}{\sqrt{(2\pi)}^{1/\tau}} \frac{w_i^{1/\tau}}{\sigma_i^{1/\tau}} \sqrt{2\pi\sigma_i^2\tau} & (17) \\ &= \frac{\sqrt{\tau}}{\sqrt{(2\pi)}^{(1/\tau)-1}} \frac{w_i^{1/\tau}}{\sigma_i^{(1/\tau)-1}}. & (18) \end{aligned}$$

Thus the normalised weight for component i, $\tilde{w}_i^{(\tau)}$, is

$$\tilde{w}_i^{(\tau)} = \frac{w_i^{1/\tau}/\sigma_i^{(1/\tau)-1}}{\sum_i w_i^{1/\tau}/\sigma_i^{(1/\tau)-1}}. \quad (19)$$

So, as $\tau \to \infty$, the weight of each of the components becomes equal. As $\tau \to 0$, we observe the expected behaviour that the highest component (the component for which w_i/σ_i is largest) gets all of the weight

$$\widehat{w_i^{(\tau)}} \to \begin{cases} 1 & \text{if } i = \arg\max \frac{w_i}{\sigma_i} \\ 0 & \text{otherwise,} \end{cases} \quad (20)$$

(assuming there is a unique highest component).

2.3.2 Effect Of Dilation

The effect of a dilation of radius ϵ on a single component is to insert a rectangle of height $\frac{w_i}{\sqrt{2\pi}\sigma_i}$ and width 2ϵ into the middle of the density:

$$(\pi_i)_\epsilon(x) = \begin{cases} \frac{1}{Z_\epsilon}\frac{w_i}{\sigma_i\sqrt{2\pi}} \exp\left(\frac{-(x+\epsilon-\mu_i)^2}{2\sigma_i^2}\right), & x < \mu_i - \epsilon \\ \frac{1}{Z_\epsilon}\frac{w_i}{\sigma_i\sqrt{2\pi}} \Phi\left(-\frac{\epsilon}{\sigma_i}\right), & \mu_i - \epsilon \le x \le \mu_i + \epsilon \\ \frac{1}{Z_\epsilon}\frac{w_i}{\sigma_i\sqrt{2\pi}} \exp\left(\frac{-(x-\epsilon-\mu_i)^2}{2\sigma_i^2}\right), & x > \mu_i + \epsilon, \end{cases} \quad (21)$$

where Z_ϵ is the normalising constant of the density and Φ is the distribution of the standard normal.

Thus the effective height of the component is $\frac{1}{Z_\epsilon} w_i/(\sigma_i\sqrt{2\pi})$, so the relative heights of the components does not change. When ϵ grows large, the dilated component can be approximated by a uniform distribution, so the effective width is approximately $\epsilon/\sqrt{3}$ (using the standard deviation of the uniform distribution). The unnormalised effective weight is $w_i + 2\epsilon w_i/(\sigma_i\sqrt{2\pi})$.

Consider the ratio of the weights of components 1 and 2. When ϵ is zero, the ratio is w_1/w_2. When ϵ is large, the term $2\epsilon w_i/(\sigma_i\sqrt{2\pi})$ dominates w_i, so the ratio tends towards the ratio of the original heights of the components, $(w_1/w_2)(\sigma_2/\sigma_1)$.

2.3.3 Comparison Of The Effects

As the number of points simulated by a Monte Carlo algorithm tends to infinity, the number of points from each component will be drawn in proportion to the weights of the components. However, for a finite sample, the widths of the components are significant since they influence how easy it is for the Monte Carlo algorithm to find the component. The width of a dilated component, $\epsilon/\sqrt{3}$, is independent of the original width, whereas the width of the tempered component, $\sigma_i\sqrt{\tau}$, is not. This indicates the main advantage of dilations over tempered targets – the effect of the dilation is large however narrow the original mode is. When tempering we may have to use an extremely high temperature to ensure that the width of a narrow component becomes significant. The disadvantage of doing this is that the temperature may have to be so high that the target function is approximately 1 everywhere, which makes it of little use.

Thie example also illustrates a known weakness of tempering when used in simulation. For a high temperature, a method will simulate many points in regions corresponding to components that have little weight in the untempered target. Dilation suffers a similar weakness: although it may be used to find narrow components, it may simulate many points in components that have little weight in the undilated target.

2.4 Dilation In PMC

Alongside devising targets π_i for use in PMC algorithms using annealing, we use instead *dilation* to construct alternative π_i. Interval dilation is the mechanism through which we allow PMC to make use of interval methods. When simulating from the target f_ϵ, each Monte Carlo point gains a view of the range of the distribution or objective function in the box of radius ϵ around the point.

Here we describe precisely how we will use interval dilations in PMC algorithms, first in SMC, then in population MCMC. In both cases we use auxiliary targets as the mechanism by which the Monte Carlo algorithm makes use of the information provided by interval methods, specifically

- **Sequential Monte Carlo.** To use interval dilated targets in SMC we simply modify the SMC optimization method outlined in section 1.2.3 to use a sequence of targets that is both annealed and dilated. Specifically, we choose a sequence of functions $\pi_{1:T}$ where $\pi_t = \pi_{\epsilon_t}^{1/\tau_t}$, with τ_t being some sequence of temperatures that decreases to zero (using one of the annealing schemes described in section 1.1.3) and ϵ_t some sequence of dilation radii decreasing to zero.

- **Population MCMC.** The most straightforward use of dilated targets in population MCMC is to chose the targets π_j in equation 9 to be $\pi_j = \pi_{\epsilon_j}$ with ϵ_j some sequence of dilation radii decreasing to zero. We then perform MCMC or SA on the composite target π^* to obtain, respectively, a simulation or global optimization algorithm.

Since the methods we describe do not deviate from the frameworks described in section 1.2 - they are purely Monte Carlo methods - all of the properties of these methods are exactly as described in that section.

3 Optimisation Example

In order to illustrate precisely the benefits of our proposed approach, we focus on a concrete example. The problem we tackle is the maximisation of a (unnormalised) Levy density $l : \mathbb{R}^2 \to \mathbb{R}$ given by the following equation:

$$\begin{aligned} l(X_1, X_2) \quad = \quad & \exp\left\{ \left[\left(\sum_{i=1}^{5} i \cos((i-1)X_1 + i) \right) \right. \right. \\ & \left(\sum_{j=1}^{5} j \cos((j-1)X_2 + j) \right) + \\ & (X_1 + 1.42513)^2 + \\ & \left. \left. (X_2 + 0.80032)^2 \right] / 40 \right\}. \end{aligned} \tag{22}$$

This is a test problem from the optimization literature with a large number of unequal peaks known as the "Levy Number 5". The true maximum is found at $(-1.3068, -1.4248)$ (to 5 s.f.). This function (shown in figure 2a) is challenging for most methods, but is exactly the type of problem for which we anticipate interval methods being useful because:

- **The modes are narrow and well separated.** Using an annealed version of the function with a high temperature only partially helps in this extreme case: a very high temperature is required in order the make the modes wide enough to be easily found, but this results in the entire function being too flat for this annealing to be useful.
- **It is an elementary function.**
- **The variables X_1 and X_2 do not occur many times in the expression, so over-enclosure will not be large.**

In the following sections we describe the results of applying the methods described in this paper to this example problem, using approximately the same computational time for each approach (with the exception of one of the runs of SMC, where the performance is sufficiently good for us to use a tenth of the effort). We ran each algorithm 25 times: table 1 gives the mean absolute distance from the true maximum (to 4 d.p.). To give a balance between moving within the modes and having the chance to move between modes, all algorithms use the following proposal in their Metropolis-Hastings moves

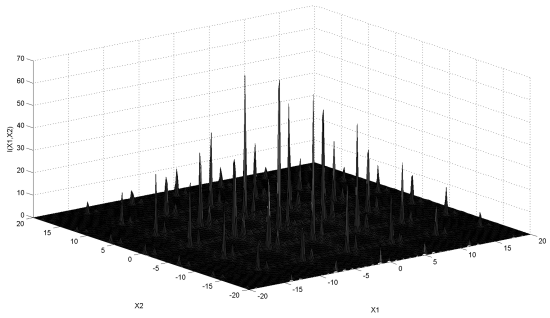
$$q(\cdot|x^{(i)}) = 0.5\mathcal{N}(\cdot|x^{(i)}, 0.1I) + 0.5\mathcal{N}(\cdot|x^{(i)}, 9I), \tag{23}$$

where $\mathcal{N}(\cdot|\mu, \Sigma)$ is the normal distribution with mean μ and variance Σ. All initial values are drawn uniformly from $[-20, 20] \times [-20, 20]$.

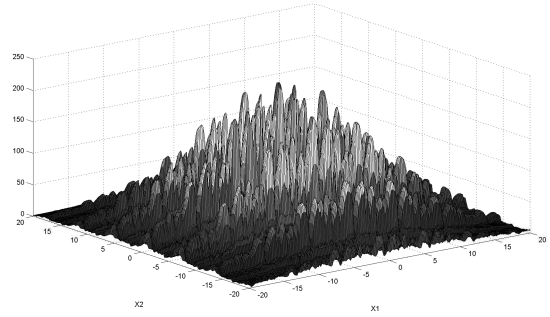
3.1 Simulated Annealing

3.1.1 Standard Simulated Annealing

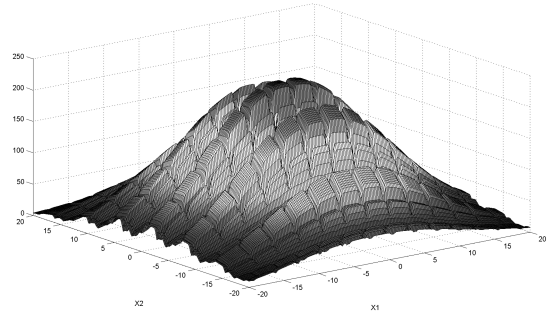
We used SA as described in section 1.1.3, with $\pi^{1/\tau_i} = l(X_1, X_2)^{1/\tau_i}$. We used 20000 iterations, with $\tau_i = 1$ for $1 \leq i \leq 10000$ and $\tau_i = (20001 - i)/10000$ for $10000 < i \leq 20000$. Starting with the $\tau_i > 1$ made little impact on the results in this case: the use of targets with $\tau_i > 1$ to aid exploration of the space is investigated below within the



(a) The Levy Number 5 test function.



(b) The Levy Number 5 test function dilated by $\epsilon = 0.5$.



(c) The Levy Number 5 test function dilated by $\epsilon = 1$.

Figure 2: The Levy Number 5 test function, together with its dilations by $\epsilon = 0.5$ and $\epsilon = 1$.

Table 1: The performance of several optimization algorithms on the Levy density in equation 22.

Algorithm	Mean abs error
Simulated annealing	9.8190
Simulated annealing with dilation	3.2727
Population version of simulated annealing, all functions the same	13.5991
Tempered population (MCMC), max temp 10	6.0085
Tempered population (MCMC), max temp 100	5.0469
Tempered population (MCMC), max temp 1000	5.4440
Dilated population (MCMC), max radius 0.5	15.2719
Dilated population (MCMC), max radius 1	0.0162
Dilated population (MCMC), max radius 2	0.0158
Tempered SMC, 200 particles	0.5099
Tempered and dilated SMC, 20 particles	0.0408

population MCMC algorithm. We observe that SA exhibits very poor performance, with a mean error of 9.8190.

3.1.2 Simulated Annealing With Dilation

Here we explore the use of SA, where the sequence of distributions uses dilation as well as tempering. Dilated versions of the objective function can be found in figures 2b and 2c. We observed that the effect of dilation is to significantly smooth out the objective function, making it easier to explore. At iteration i of our dilated SA algorithm our target distribution is $\pi_{\epsilon_i}^{1/\tau_i}$, where the τ_i sequence is the same as in the previous section, and ϵ_i is a sequence of dilation radii that decreases to zero. The sequence we use is to space ϵ_i equally from $\epsilon_{\max} = 1$ at $i = 1$ to 0 at $i = 19000$. For $i > 19000$, we take $\epsilon_i = 0$. The performance of dilated SA is much better than standard SA, with a mean error of 3.2727, although the true maximum is rarely found.

3.2 Population MCMC

We now examine the use of population MCMC, as described in section 1.2.2. We use the target distribution in equation 9, where one of π_1 is the true target distribution $l(X_1, X_2)$, and the number of targets is $M = 100$. To use this algorithm for optimization we simply temper the target at every iteration: we use $(\pi^*)^{1/\tau_i}$. As for SA, we perform $N = 20000$ iterations, with $t_i = 1$ for $1 \leq i \leq 10000$ and $\tau_i = (20001-i)/10000$ for $10000 < i \leq 20000$. At each iteration of the algorithm we uniformly randomly select a chain to "Mutate"', then uniformly randomly pick a chain to "Exchange" with its neighbour. The estimated maximum from the algorithm is taken to be the value $x_1^{(N)}$: the final point of the chain for the true target distribution.

3.2.1 Population MCMC With Tempering

This algorithm uses a population of tempered targets: we choose the π_j to be tempered with temperatures $\tau_j \geq 1$ to help exploration of the target. Specifically we choose $\tau_j = 1$ at $j = 1$, $\tau_j = \tau_{\max}$ at $j = N$ and the other temperatures equally spaced between these two extremes. The errors for $\tau\max = 10, 100$ and 1000 are $6.0085, 5.0469$ and 5.4440 respectively. We observe a relatively small improvement over SA in all cases: the use of tempered targets does help the exploration of the space, but the effect is not dramatic. Note that it is difficult to tune the method in terms of deciding the number of members of population and the maximum temperature that should be used. It needs to be chosen large enough in order to allow moves between the modes, but not so large that no useful path is created between the distributions in the population.

3.2.2 Population MCMC With Dilation

We now use a population of dilated targets: we choose the π_j to be dilated with radii $\epsilon_j \geq 0$ to help exploration of the target. We choose $\epsilon_j = 0$ at $j = 1$ (to give the true target), $\epsilon_j = \epsilon_{\max}$ at $j = N$ and the other radii equally spaced between these two extremes. The errors for $\epsilon\max = 0.5, 1$ and 2 are $15.2719, 0.0162$ and 0.0158 respectively. Here the improvement over SA and population MCMC with tempering is dramatic. As long as $\epsilon_{\max}$ is large enough to span the distance between peaks, the peak containing the true maximum was located at every run of the algorithm. This provides a clear demonstration of when dilated targets may be useful and also, when comparing to dilated SA, of the advantages of population MCMC over the use of a single chain. Similarly to the tempering approach, it is not obvious in advance how to choose the most dilated target in order to achieve a good performance. Here, when $\epsilon\max = 0.5$, no useful paths are created between the modes, leading to poor results.

3.3 Sequential Monte Carlo

Finally we apply the SMC algorithm in algorithm 2 to this problem. We use a sequence of $T = 200$ targets, and consider two schemes:

- **Annealing only:** we choose a sequence of targets $\pi_t = \pi^{1/\tau_t}$, with $\tau_t = 1$ for $1 \leq t \leq 100$ and $\tau_t = (201 - t)/100$ for $100 < t \leq 200$. This choice yields an algorithm that is essentially similar to running a population of simulated annealing algorithms, with a selection step (via resampling) to keep only the best performing members of the population. We do not make use of tempered targets with $\tau > 1$ in order to give a clearer comparison with the SA in section 3.1.1.
- **Annealing and dilation:** we choose a sequence of targets $\pi_t = \pi_{\epsilon_t}^{1/\tau_t}$, where τ_t is the same sequence as for the annealing only sampler, and ϵ_t is a sequence of dilation radii that decreases to zero. Specifically, the ϵ_t are equally spaced from $\epsilon_{\max} = 1$ at $t = 1$ to 0 at $t = 190$, and then for $t > 190$, we take $\epsilon_i = 0$.

Using $N = 200$ particles, the annealing only algorithm achieved an error of 0.5099. This is favourable compared to most of the other algorithms used, except for those that use dilation, providing a clear demonstration of how effective SMC can be in this setting. Also using 200 particles, the SMC sampler that uses annealing and dilation found the peak containing the true maximum on every run. In this case, we experimented with reducing the number of particles and found that using as few as 20 particles resulted in the correct peak being found on every run, yielding an

error of 0.0408 (similar to the error when using 200 particles, which is not reported in the table). None of the other algorithms were able to give a comparable level of performance for similar reduction in computational effort, again illustrating how the use of dilated targets may be beneficial.

4 Simulation Example

In this section we use interval dilated versions of a target π as auxiliary targets within the population MCMC algorithm. We apply the algorithm to inferring the means of a mixture of Gaussians, an example that is also used in Jasra et al [15] for illustrating the properties of population-based simulation algorithms. Similar to that work, we do not consider the best statistical approach to the problem; rather we use the fact that there are known multiple modes in the posterior distribution due to the lack of identifiability of the mixture, and examine the mixing of the method over these modes. We begin by giving the specifics of the target under consideration and the algorithm we apply to it, then show the effect of interval dilation on the target. We then examine the effect of dilation on the likelihood in the two component model, followed by examining how the performance of population MCMC algorithms is affected by the number of mixture components (which affects the dimensionality of the parameter space). This section aims to illustrate limitations of the proposed approach, as well as its strengths.

4.1 Model And Algorithm Details

Let $\{y_i\}_{i=1}^N$ be a data set with $y_i \in \mathbb{R}$. We assume that the y_i are i.i.d., each with density:

$$p(y_i|\{\mu_j, \Sigma_j, w_j\}_{j=1}^k) = \sum_{j=1}^k w_j \mathcal{N}(y_i; \mu_j, \Sigma_j), \tag{24}$$

where $\mathcal{N}(y_i; \mu_j, \Sigma_j)$ denotes a normal density on y with mean μ_j and variance Σ_j. We apply the population MCMC algorithm to the posterior for the means of a mixture of Gaussian densities

$$p(\{\mu_j\}_{j=1}^k|y) = p(y|\{\mu_j, \Sigma_j, w_j\}_{j=1}^k)p(\{\mu_j\}_{j=1}^k). \tag{25}$$

Our results are based on 5 data points from each mode (similar results in terms of the relative performance of the algorithms were observed for when using 100 or 1000 data points per mode) and we take the mean of the j'th component to be $3(j-1)$. We use the same variance and use an equal weight for each component.

For the population MCMC algorithm, at each iteration we uniformly at random select a chain to "Mutate", then uniformly at random select a chain to "Exchange" with its neighbour. There are two accept-reject stages in this algorithm, thus one iteration of the algorithm costs roughly twice as much as a standard Metropolis-Hastings algorithm. We compare population MCMC algorithms that use tempered auxiliary targets with population MCMC algorithms that use interval dilated auxiliary targets. We always use $N = 10$ targets, including the true target. Our tempering choice is similar to that in Jasra et al [15] – we choose the temperatures to be equally spaced between 1 and some maximum $T_{\max}$ (inclusive of these limits).

The dilation scheme is influenced by what we know about the posterior. We pick a maximum dilation radius, $\epsilon_{\max} = 2$, that is about the same as half of the distance

between the nearest modes in the posterior. We choose interval dilations with ϵ equally spaced between 0 and 2 (including these two limits).

The choice of proposals for the "Mutate" move in the algorithms is not trivial, since it is likely that better performance will be obtained by using different proposals in different auxiliary chains. We ignore this issue here and choose the same proposals for each chain. We choose a random walk proposal – where x_t is the current state of the chain:

$$q(x_{t+1}|x_t) = 0.5\mathcal{N}(x_{t+1}; x_t, 0.1I) + 0.5\mathcal{N}(x_{t+1}; x_t, 9I). \tag{26}$$

This proposal is chosen in order to give a balance between moving within the modes and having the chance to move between modes.

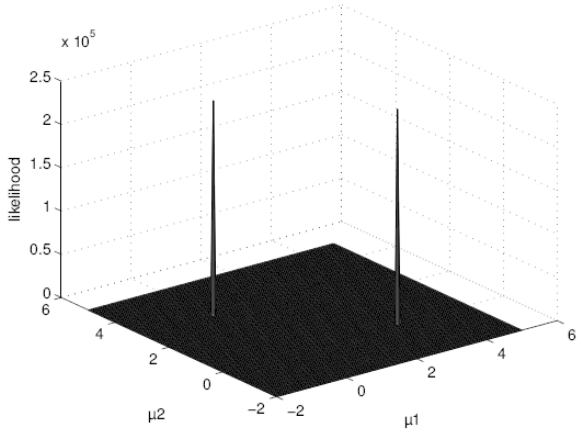
4.2 Illustrations Of Interval Dilations

The idea of using interval dilations is that they provide a path in the product space between the two well separated modes in the true target. It is easy to see this by looking at plots of interval dilations of the likelihood. The likelihood we consider is for the 2 dimensional ($k = 2$) case, with each component in the mixture having a variance of $\Sigma_i = 0.01$. In figure 3a we show the true likelihood, with its interval dilations by 0.5 and 2 in figures 3b and 3c respectively. In the figure 3b we observe the desired effect of interval dilation; that it makes the peaks in the likelihood wider, despite being very narrow in the original function. However we also observe, noting the change in scale between the two figures, a potential drawback of the approach: the over-enclosure of the likelihood is very large, even with only 5 data points. This over-enclosure arises due to the likelihood being a product of terms. The parameter is present in the representation for the likelihood the same number of times as there are data points, thus the dependency problem is severe and a very large over-enclosure results. In this case, we do not anticipate the over-enclosure to adversely effect the performance of the method, since the normalising constant of the function used in the MCMC does not matter. However, we highlight that over-enclosure will always result from using a likelihood that takes the form of a product. This has the potential to give interval dilations that have shapes that are difficult to predict. For example, figure 3c shows that for the interval dilation by radius 2, the mass is dominated by over-enclosure in the region where the dilations of the two modes overlap. This function is very different, and much less useful that, the true dilation of the likelihood by the same radius.

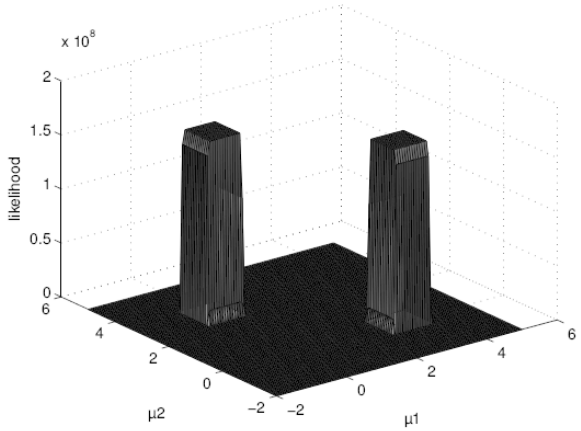
4.3 The Effect Of Dimension

In this section we consider three separate algorithms: the standard random-walk Metropolis-Hastings algorithm; the population MCMC algorithm with tempered targets (using the tempering scheme above); and the population MCMC algorithm with dilated targets (using the interval dilation scheme above). We generate 10,000 points using each algorithm. We compare the performance of each method as the dimension increases. The variance of each component was $\Sigma_i = 0.06^2$ in all of the experiments. We use tempering and dilation schemes that should be well suited to the problem under consideration: for the dilation scheme $\epsilon_{\max} = 2$ and for the tempering scheme $T_{\max} = 100$.

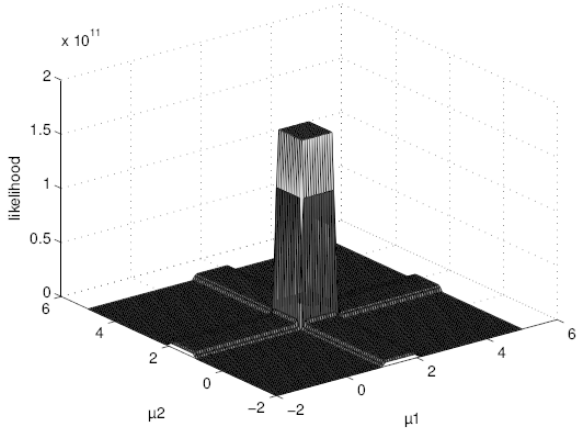
We initialise every chain in every algorithm at the mode $(0, 3, ..., 3(k-1))'$ (for the k dimensional target in equation (25)) and generate 10^6 points using each algorithm.



(a) The likelihood.



(b) Interval dilation by $\epsilon = 0.5$.



(c) Interval dilation by $\epsilon = 2$.

Figure 3: The interval dilation by different values of ϵ of the likelihood of the means of a Gaussian mixture with two components, where each component in the mixture has a variance of 0.01.

The first 40% of the points were discarded as a burn in period. We find the trace of the autocorrelation time (this is infinite if there is no movement in one or more of the dimensions), the sample mean and the number of jumps between modes for the points generated by each of the algorithms. The sample mean should give an idea as to how many of the modes the algorithm has found – the mean should be the same in each dimension, taking a value of $3(k-1)/2$ in k dimensions. The integrated autocorrelation time (ACT) was estimated for each method. The results are displayed in table 2 ("MH" is standard Metropolis-Hastings, "Temp" refers to the population MCMC method with a tempered population and "Dil" refers to the population method with a dilated population).

Table 2: Sampling results of the population MCMC algorithms.

Dim.	Method	Mean	ACT	No. jumps
2	MH	(1.9181, 1.0835)	196.5	3
2	Temp	(1.5014, 1.5006)	142.8	612
2	Dil	(1.5256, 1.4732)	89.0	2030
3	MH	(-0.0048, 3.0086, 5.9950)	171.8	0
3	Temp	(2.0958, 3.7941, 3.1213)	177.0	354
3	Dil	(3.9928, 2.2137, 2.7960)	147.4	2023
4	MH	(-0.0167, 3.0189, 6.0239, 9.0166)	194.5	0
4	Temp	(0.0460, 3.0150, 6.1487, 8.8768)	71.4	216
4	Dil	(5.0836, 2.3675, 5.3397, 5.2494)	103.9	3155
5	MH	(0.0251, 3.0267, 5.9678, 9.0289, 11.9773)	∞	0
5	Temp	(0.0071, 2.9286, 5.9391, 8.9760, 11.9646)	157.7	0
5	Dil	(-0.0259 3.0046 6.0150 9.1281 11.9708)	66.8	301
6	MH	(-0.0216, 2.9550, 5.9537, 9.0060, 12.0820, 15.0631)	197.4	0
6	Temp	(0.0083, 2.9775, 6.0146, 8.9906, 11.9232, 15.0070)	162.4	0
6	Dil	(0.0081, 3.0287, 5.9113, 9.0760, 12.1069, 15.0650)	23.1	0

We see that the interval dilations have helped us to jump between the modes in cases where they are too narrow for the tempered population MCMC algorithm to work. These results also suggest the advantage of interval dilation over tempering is maintained in higher dimensions. However, this advantage is only present when the support of the peaks in the target is small – in this case the interval dilation scheme is more effective at finding the peaks than the tempering scheme. When the support of the peaks is larger (by the components from which we simulated the data having a higher variance) we found that this advantage is lost (results not shown). In this case the tempering scheme has no problem finding the modes and there is no longer an advantage in interval dilation. In fact, in this case the interval dilated targets are less useful than the tempered targets since they are a different shape to the true target.

5 Discussion

In this paper we study the problem of global optimization for objective functions with narrow, well separated peaks. PMC methods are natural extensions of simulated annealing that use a population of points to tackle this type of problem. However, PMC methods are usually based on annealed targets, which are not effective if the modes in the objective function are too narrow. This paper describes a method for incorporating the information provided by interval methods into PMC algorithms, through their use in finding dilated target functions, in order to more effectively explore this type of target function.

Empirically we find that SMC samplers in particular can perform well on this type of problem. One further appealing property of these methods is that many of the choices we may make in the algorithmic design can be partially, or sometimes fully, automated. To simplify the presentation of the algorithm, these ideas were not included in the paper: the sequence of distributions may be chosen automatically as the algorithm progresses, in order to ensure the distance between successive distributions is not too large; and the proposal distribution for the Metropolis-Hastings step may be chosen adaptively, to match the properties of the distribution being explored at each iteration [3, 4, 7, 29]. This can make SMC easier to tune than population MCMC; designing adaptive schemes is more difficult for MCMC algorithms (see Andrieu and Thoms [1] for a review). SMC samplers may also be improved by using alternative resampling schemes (see Doucet and Johansen [8]; stratified resampling is often a good choice) or using proposal distributions that exploit the gradient or curvature of the target distribution (e.g. Septier and Peters [30]).

We have observed that interval methods, through the use of dilated target distributions, can significantly improve the performance of Monte Carlo methods. Similar improvements over annealing-only schemes should be expected when three conditions are satisfied:

1. the objective function is an elementary function;
2. there are narrow, well separated peaks;
3. the over-enclosure of the objective function is small enough that the interval dilation is similar in shape to the true dilation.

The last of these conditions is probably the most important. It is satisfied when each variable only occurs a small number of times in the function expression - recall that if each variable only occurs once, the dependency problem does not occur, and there is no over-enclosure. This issue means that we advise caution in using the approach with posterior distributions or likelihoods that involve the product over a large number of terms. In some cases this can be avoided through the use of summary statistics.

Several ways of countering the dependency problem have been devised (see e.g. Moore et al [24]), but are out of the scope of this paper. Any of these alternative methods can be used as a substitute for standard interval methods in most of the applications in this paper, and if they yield a smaller over-enclosure with a small additional computational cost, it is likely that an improvement over the results obtained in the paper will be obtained.

Acknowledgements

I wish to thank the anonymous referees who helped improve the manuscript.

References

[1] Andrieu C, Thoms J (2008) A tutorial on adaptive MCMC. Statistics and Computing 18(4):343–373

[2] Kearfott RB (1996) Rigorous Global Search: Continuous Problems. Springer

[3] Beaumont MA, Cornuet JM, Marin JM, Robert CP (2009) Adaptive approximate Bayesian computation. Biometrika 96(4):983–990

[4] Beskos A, Jasra A, Muzaffer E, Stuart A (2015) Sequential Monte Carlo Methods for Bayesian Elliptic Inverse Problems. Statistics and Computing 25:727–737

[5] Botev ZI, Kroese DP, Rubinstein RY, L'Ecuyer P (2013) The Cross-Entropy Method for Optimization. In: Rao CR, Govindaraju V (eds) Handbook of Statistics, Volume 31: Machine Learning Theory and Applications, North Holland & IFIP, pp 35–59

[6] Del Moral P, Doucet A, Jasra A (2006) Sequential monte carlo samplers. Journal of the Royal Statistical Society: Series B(Statistical Methodology) 68(3):411–436

[7] Del Moral P, Doucet A, Jasra A (2012) An adaptive sequential Monte Carlo method for approximate Bayesian computation. Statistics and Computing 22(5):1009–1020

[8] Doucet A, Johansen AM (2009) Particle Filtering and Smoothing: Fifteen years later. Handbook of Nonlinear Filtering 12:656–704

[9] Everitt RG (2017) Efficient importance sampling in low dimensions using affine arithmetic. Computational Statistics 33(1):1–29

[10] Geyer C (1991) Markov chain Monte Carlo maximum likelihood. In: Keramidas EM (ed) Computing Science and Statistics: Proceedings of the 23rd Symposium on the Interface, Interface Foundation of North America, Fairfax Station, VA, pp 156–163

[11] Glasbey CA, Horgan GW (1995) Mathematical Morphology. In: Image Analysis for the Biological Sciences, John Wiley & Sons, Inc., New York, NY, USA

[12] Gordon NJ, Salmond DJ, Smith AFM (1993) Novel approach to nonlinear/non-Gaussian Bayesian state estimation. In: Radar and Signal Processing, IEE Proceedings F, IET, vol 140, pp 107–113

[13] Hastings WK (1970) Monte Carlo sampling methods using Markov chains and their applications. Biometrika 57(1):97–109

[14] Henderson D, Johnson AW (2003) The Theory and Practice of Simulated Annealing. In: Handbook of Metaheuristics, Springer US, pp 287–319

[15] Jasra A, Stephens DA, Holmes CC (2007) On population-based simulation for static inference. Statistics and Computing 17(3):263–279

[16] Jaulin L, Kieffer M, Didrit O, Walter E (2001) Applied interval analysis: with examples in parameter and state estimation, robust control and robotics. Springer Verlag

[17] Johansen AM, Doucet A, Davy M (2007) Particle methods for maximum likelihood estimation in latent variable models. Statistics and Computing 18(1):47–57

[18] Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by Simulated Annealing S. Science 220(4598):671–680

[19] Laskey K, Myers J (2003) Population markov chain monte carlo. Machine Learning 50(1):175–196

[20] Liang F, Wong W (2000) Evolutionary Monte Carlo: Applications to Cp Model Sampling and Change Point Problem. Statistica Sinica 10:317–342

[21] Metropolis N, Rosenbluth AW, Rosenbluth MN, Teller AH, Teller E (1953) Equations of State Calculations by Fast Computing Machines. Journal of Chemical Physics 21:1087–1092

[22] Moore RE (1962) Interval Arithmetic and Automatic Error Analysis in Digital Computing. PhD thesis, Stanford University

[23] Moore RE (1966) Interval Analysis. Prentice-Hall, New York

[24] Moore RE, Kearfott RB, Cloud MJ (2009) Introduction to Interval Analysis. Society for Industrial and Applied Mathematics, Philadelphia

[25] Peters GW (2005) Topics in Sequential Monte Carlo Samplers. PhD thesis, University of Cambridge

[26] Roberts GO, Rosenthal JS (2004) General state space Markov chains and MCMC algorithms. Probability Surveys 1:20–71

[27] Sainudiin R (2005) Machine interval experiments. PhD thesis, Cornell University

[28] Sainudiin R, York T (2009) Auto-validating von Neumann rejection sampling from small phylogenetic tree spaces. Algorithms for Molecular Biology 4(1)

[29] Schafer C (2013) Particle algorithms for optimization on binary spaces. ACM Transactions on Modeling and Computer Simulation 23(1)

[30] Septier F, Peters GW (2016) Langevin and Hamiltonian Based Sequential MCMC for Efficient Bayesian Filtering in High-Dimensional Spaces. IEEE Journal on Selected Topics in Signal Processing 10(2):312–327

[31] Strens MJA (2003) Evolutionary MCMC sampling and optimization in discrete spaces. Proceedings of the 20th International Conference on Machine Learning

[32] Strens MJA, Bernhardt M, Everett NO (2002) Markov Chain Monte Carlo Sampling using Direct Search Optimization. Proceedings of the 19th International Conference on Machine Learning

A Interval Methods

This appendix contains a summary of the fundamentals of interval analysis that are required for this paper, for researchers who are not familiar with this field.

Interval methods have their roots in numerical analysis, with the original aim being to describe errors due to performing calculations using the finite representation of real numbers in a computer. The methods keep track of the errors introduced by the calculation, so it is possible to automatically decide if it is necessary to refine the calculation in order to reduce the error. The first work in this field was the introduction of *interval analysis* by Moore [23]. This idea lets us achieve the following. Suppose we have a real valued function $f : D \to \mathbb{R}$ with $D \subseteq \mathbb{R}$, with the range of f over a subset $E \subseteq D$ defined to be $f(E) = \{f(x) \mid x \in E\}$. For any interval $I_D \subseteq D$, interval analysis gives a computationally cheap way of finding an interval $I_R \subset \mathbb{R}$ such that $f(I_D) \subseteq I_R$, for a large class of functions f. This property is known as *range enclosure*. In some cases $f(I_D)$ is close enough to I_R for us to be able to use I_R as an approximation for $f(I_D)$ (in fact, in some cases we have $I_R = f(I_D)$). However, sometimes it is the case that I_R is significantly larger than $f(I_D)$, in which case we say we have a large *over-enclosure*. It is this computationally cheap manner of finding a "range approximation", an interval I_R such that $I_R \approx f(I_D)$, that we make use of in the paper. The major limitation on our use of interval analysis is in the case where we have a large over-enclosure.

In this section (based mainly on the descriptions in Jaulin et al [16] and Sainudiin [27]), we introduce interval extensions of functions (section A.1). The discussion of their use for optimization leads us on to the main contribution of the paper in section 2, where these methods are used within PMC.

A.1 Interval Extensions

We begin by making some definitions. The *interval extension* of the real numbers, is the set of compact intervals in $\mathbb{R}$, $\mathbb{IR} := \{[a, b] \mid a, b \in \mathbb{R}, a < b\}$. We denote the minimum value of an interval $X \in \mathbb{IR}$ by $\underline{x}$ and its maximum by $\overline{x}$. Likewise, the interval extension $\mathbb{I}D$, of any subset $D \subset \mathbb{R}$ is: $\mathbb{I}D := \{[a, b] \mid a, b \in D, a < b\}$. We say that a function $F : \mathbb{I}D \to \mathbb{IR}$ satisfies range enclosure on $f : D \subset \mathbb{R} \to \mathbb{R}$ if $\forall X \in \mathbb{I}D$, $f(X) \subseteq F(X)$, with range equivalence if $f(X) = F(X)$. In this section, for any f that is a composition of commonly used functions, we give a recipe for building a function $F : \mathbb{I}D \to \mathbb{IR}$ that satisfies range enclosure with respect to an arbitrary f (where f is an *elementary* function) and also *inclusion isotony*: $\forall X, X' \in \mathbb{I}D$ with $X \subseteq X'$, $F(X) \subseteq F(X')$. Such a function is called an *interval extension* of f.

The interval extension that we define is a "natural" interval extension. To define a natural interval extension of f, we need to consider a "representation" of f: that is, we need to express it as a composition of "simple" functions. We then define the natural interval extension to be the composition of natural interval extensions of each of these simple functions. Thus, to begin we define interval extensions of arithmetic operations, then move on to interval extensions of standard functions.

Arithmetic on $\mathbb{IR}$ is defined as

$$X * Y := \{x * y \mid x \in X, y \in Y\}, \tag{27}$$

where $X, Y \in \mathbb{IR}$ and $* \in \{+, -, \times, /\}$, except for when $*$ is $/$ and $0 \in Y$, in which case we leave $X * Y$ undefined (note that when used in practice, we use the convention that dividing by zero results in infinity). Because of the monotonicity of each operator

on the first argument, it is easy to see that the minimum and maximum of $X * Y$ are attained on the boundary values of X and Y, thus we can write expressions for $X * Y$ for each value of $*$ using these boundary values: e.g. $X + Y = [\underline{x} + \underline{y}, \overline{x} + \overline{y}]$.

We now define the set of standard functions $\mathcal{G}$ to be

$$\begin{aligned}\mathcal{G} \quad := \quad \{&a^x, \log_b(x), x^a, ax, x + a, |x|, \sin(x), \cos(x), \tan(x),\\ &\sinh(x), \cosh(x), \tanh(x), \arcsin(x), \arccos(x),\\ &\arctan(x)\}\,.\end{aligned}$$

where $a \in \mathbb{R}$ and $b \in \mathbb{R}_{\geq 0, \neq 1}$ are constants. Other functions may be included in this set, but here we keep close to the conventions established in the literature on interval arithmetic. For each of the functions in $\mathcal{G}$ that is monotone, again, the interval extension can be defined in terms of the boundary points of an input interval. For example, for $X \in \mathbb{IR}$:

$$\exp(X) = [\exp(\underline{x}), \exp(\overline{x})]. \tag{28}$$

It is also simple to define interval extensions of the members of $\mathcal{G}$ that are not monotone but piecewise monotone. For example, if an input interval is such that $\cos(x)$ is monotone on this interval, the interval extension is given simply by evaluating cos on the endpoints of this interval. However, if (for example) the interval contains any point in the set $\{2k/\pi\}$ (for integer k) then we know that the upper bound of the output interval should be 1, and that the lower bound is given by evaluating cos on the endpoints of the interval. It is easy to see that functions on $\mathbb{IR}$ defined in this way satisfy range equivalence.

Finally we say that an elementary function $f : D \subseteq \mathbb{R} \to \mathbb{R}$ is a function that may be expressed as a finite combination of constants, variables, arithmetic operations, standard functions and compositions, and denote the set of elementary functions by $\mathcal{E}$. We refer to a particular expression of $f \in \mathcal{E}$ as a finite combination of constants, variables, arithmetic operations, standard functions and compositions as a *representation* for f. Every member of $\mathcal{E}$ has an infinite number of representations, for example $f(x) = 1 - \frac{1}{x+1}$ can be rewritten as $f(x) = \frac{x}{x+1} + 5 - 5$. We then define the natural *interval extension* of a representation of $f \in \mathcal{E}$ as the function $F : \mathbb{I}D \to \mathbb{IR}$ where every constant, variable, operation and standard function in the representation of f is replaced by its interval counterpart. The *Fundamental Theorem of Interval Analysis* says the natural interval extensions of elementary functions satisfy inclusion isotony and range enclosure.

Importantly for our use of interval methods, it is trivial to extend these results to functions defined on multi-dimensional domains. Suppose we define $\mathbb{IR}^n := \{(X_1, ..., X_n) \mid X_i = [a_i, b_i] \text{ with } a_i, b_i \in \mathbb{R}, a_i < b_i\}$ to be the set of compact *boxes* (Cartesian products of intervals) in $\mathbb{R}^n$, and consider the natural interval extension $F : \mathbb{IR}^d \to \mathbb{IR}$ of an objective function $f \in \mathcal{E}$. We have the following result on the *over-enclosure* of f by F (from Kearfott [2]: there exists a constant K, independent of the box X where $w(F(X)) - w(f(X)) \leq Kw(X)$, with $w(X)$ denoting the maximum of each the widths of each of the dimensions of the box X.

In practice, the extent of the over-enclosure of a function $f \in \mathcal{E}$ by a natural interval extension F depends on the representation used for f. A simple example where the over-enclosure depends on the representation is $f(x) = 1 - \frac{1}{x+1} = \frac{x}{x+1}$. Using the definitions earlier in this section, evaluating the first and second expressions

on $[0, 1]$ gives

$$\begin{aligned} f(x) &= 1 - \frac{1}{[0,1]+1} && (29)\\ &= 1 - \frac{1}{[1,2]} && (30)\\ &= 1 - [1/2, 1] && (31)\\ &= [0, 1/2] && (32) \end{aligned}$$

(the true range) and

$$\begin{aligned} f(x) &= \frac{[0,1]}{[0,1]+1} && (33)\\ &= \frac{[0,1]}{[1,2]} && (34)\\ &= [0,1], && (35) \end{aligned}$$

an over-enclosure, respectively.

Over-enclosure does not occur if each variable only occurs once in a representation for a function. When the same variable occurs more than once, an over-enclosure because of the "dependency problem" is often the result. A simple illustrative example is evaluating the interval extension of the function $f(x) = x \times x$ on the interval $[-1, 1]$. We obtain a bound on the range of $[-1, 1]$. We do not obtain the true range of $[0, 1]$, since we do not acknowledge that the arguments of the $\times$ operation are the same (if we used the function x^2 we would not have this problem). To avoid the dependency problem and thus minimise over-enclosure, a good rule of thumb is to use a representation that minimises the number of occurrences of each variable.

In conclusion, for f in the set of elementary functions, interval arithmetic yields a computationally cheap (a cost of twice that of evaluating f) method for bounding the range of f on an interval. In the following section we outline a way in which this has been exploited to construct a global approximation algorithm.

A.2 Implementation

Interval arithmetic is a rather different idea to most others used in computational statistics. To be used easily in general it requires the used of a software library that converts user-inputted functions into their interval extensions. Fortunately, there are a number of libraries available for interval analysis. In C++, the `C-XSC` and `boost` libraries contain interval classes. Libraries are also available for Matlab (`Intlab`) and python (within `mpmath`). In these libraries each operation is overloaded with an interval counterpart, therefore to compute interval extensions, one needs only to evaluate the desired function on an interval rather than a floating point number.