Sherlock C, Golightly A, Henderson DA.

Adaptive, delayed-acceptance MCMC for targets with expensive likelihoods.

Journal of Computational and Graphical Statistics **(2016)**

DOI: 10.1080/10618600.2016.1231064

Adaptive, delayed-acceptance MCMC for targets with expensive likelihoods

Chris Sherlock[1]*, Andrew Golightly[2] and Daniel A. Henderson[2]

[1]Department of Mathematics and Statistics, Lancaster University, UK
[2]School of Mathematics & Statistics, Newcastle University, UK

Abstract

When conducting Bayesian inference, delayed acceptance (DA) Metropolis-Hastings (MH) algorithms and DA pseudo-marginal MH algorithms can be applied when it is computationally expensive to calculate the true posterior or an unbiased estimate thereof, but a computationally cheap approximation is available. A first accept-reject stage is applied, with the cheap approximation substituted for the true posterior in the MH acceptance ratio. Only for those proposals which pass through the first stage is the computationally expensive true posterior (or unbiased estimate thereof) evaluated, with a second accept-reject stage ensuring that detailed balance is satisfied with respect to the intended true posterior. In some scenarios there is no obvious computationally cheap approximation. A weighted average of previous evaluations of the computationally expensive posterior provides a generic approximation to the posterior. If only the k-nearest neighbours have non-zero weights then evaluation of the approximate posterior can be made computationally cheap provided that the points at which the posterior has been evaluated are stored in a multi-dimensional binary tree, known as a KD-tree. The contents of the KD-tree are potentially updated after every computationally intensive evaluation. The resulting adaptive, delayed-acceptance [pseudo-marginal] Metropolis-Hastings algorithm is justified both theoretically and empirically. Guidance on tuning parameters is provided and the methodology is applied to a discretely observed Markov jump process characterising predator-prey interactions and an ODE system describing the dynamics of an autoregulatory gene network. Supplementary material for this article is available online.

Keywords: Delayed-acceptance; surrogate; adaptive MCMC; pseudo-marginal MCMC; KD-tree.

*c.sherlock@lancaster.ac.uk

1 Introduction

A major challenge for Bayesian inference in complex statistical models is that evaluation of the likelihood or, for pseudo-marginal MCMC (Andrieu and Roberts, 2009), obtaining a realisation from an unbiased estimator of the likelihood, can be computationally expensive. The use of a *surrogate* model with a computationally inexpensive likelihood in such circumstances has a long history; see, for example, Sacks et al. (1989), Kennedy and O'Hagan (2001), Rasmussen (2003), Bliznyuk et al. (2008), Fielding et al. (2011), Joseph (2012, 2013), Overstall and Woods (2013) and Conrad et al. (2014). The use of inexpensive surrogates has also been explored in the related context of likelihood free inference or approximate Bayesian computation (ABC) (Wilkinson, 2014; Meeds and Welling, 2014).

In this paper we propose to use a relatively generic surrogate for models with expensive likelihoods and we justify its use in adaptive, delayed-acceptance (pseudo-marginal) MCMC schemes. The delayed acceptance MCMC algorithm of Christen and Fox (2005) is a two-stage Metropolis-Hastings algorithm in which, typically, proposed parameter values are accepted or rejected at the first stage based on a computationally cheap surrogate for the likelihood. Detailed balance with respect to the true posterior is ensured by a second accept-reject step, based on the computationally expensive likelihood, for those parameter values which are accepted in the first stage. Delayed acceptance algorithms thus provide draws from the posterior distribution of interest whilst potentially limiting the number of evaluations of the expensive likelihood. Recent examples of the use of surrogates in delayed acceptance algorithms can be found in Cui et al. (2011), Higdon et al. (2011), Golightly et al. (2015) and Sherlock et al. (2015), amongst others. Delayed acceptance algorithms which use data subsampling and partitioning for tackling large datasets have also been proposed; see Payne and Mallick (2014), Quiroz (2015) and Banterle et al. (2015).

For some models there may be an obvious cheap surrogate. For example, Golightly et al. (2015) use both the diffusion approximation and the linear noise approximation as surrogates for a Markov jump process in the context of analysing stochastic kinetic models. For many models, however, there are no obvious model-based candidates for the surrogate. It is natural in such scenarios to use regression-based methods which utilise previous evaluations of the computationally expensive likelihood to approximate the likelihood or the unnormalised

posterior density at new parameter values. For example, Bliznyuk et al. (2008) use radial basis functions whereas Rasmussen (2003) and Fielding et al. (2011) use Gaussian processes (GPs). In this paper, we focus on a generic surrogate based upon likelihood values at the k-nearest neighbours (e.g. Hastie et al., 2009). It is easy to implement, computationally cheap, and it adapts as new evaluations of the computationally-expensive likelihood become available. In Section 5.1 we consider the merits and disadvantages of an alternative, GP-based solution.

We approximate the likelihood at proposed parameter values by an inverse-distance-weighted average of the likelihoods of its k-nearest neighbours in the training data; the training data here consist of pairs of parameter vectors and their corresponding likelihoods. This k-NN approach has the advantage of being simple, local, flexible to the shape of the likelihood, and trivially adaptive as new training data become available. Our focus on adapting a local approximation is thus similar to that in Conrad et al. (2014) and we compare and contrast the two approaches in Section 5.1.

Although it is trivial to update our simple k-NN approximation as new training data become available, a naive implementation of adaptive MCMC algorithms may not converge to the intended target (e.g. Roberts and Rosenthal, 2007; Andrieu and Thoms, 2008). Careful control of the adaptation is therefore essential and we prove that, subject to conditions, the strategy we propose is theoretically valid. As with a number of previous adaptive MCMC algorithms (e.g. Roberts and Rosenthal, 2009; Sherlock et al., 2010), our MCMC kernel is a mixture of a fixed kernel and an adaptive kernel. Unlike previous such algorithms, however, the adaptive kernel can be most efficient when the fraction of applications that involve a computationally expensive evaluation is very low; this impacts on the rate of adaptation and on the rate of convergence. We therefore also provide theoretically-justified guidance on choosing both the mixture probability and the total number of iterations of the algorithm.

The main computational expense of the k-NN approximation is searching for the nearest neighbours. In a naive implementation this search takes $O(n)$ operations, where n is the number of training datapoints, so the computational expense of an adaptive, nearest-neighbour-based approach would grow linearly with the length of the MCMC run. Fortunately, there are more efficient algorithms. We use an approach based on storing the training data in a multi-dimensional binary tree, known as a KD-tree (Bentley, 1975; Fried-

man et al., 1977). The 'K' in 'KD-tree' indicates the dimension of the space, but to avoid confusion with the number of nearest neighbours k, we denote the dimension of the space by d. Because our tree grows on-line as new training data become available we make two major changes to the standard KD-tree algorithm, described at the start of the next section. Our adapted KD-tree algorithm allows us to efficiently search for the nearest neighbours in approximately $O(d \log n)$ operations and add an additional value to the training set also in $O(d \log n)$ operations. This yields a highly efficient, adaptive surrogate.

The remainder of the paper is structured as follows. Our KD-tree k-NN algorithm is described in Section 2 and its use in adaptive, delayed-acceptance (pseudo-marginal) MCMC is discussed in Section 3. This section also provides the theoretically-justified guidance on the choice of kernel mixture probability and number of iterations. Section 4 applies the methodology to a discretely observed Markov jump process characterising predator-prey interactions and an ODE system describing the dynamics of an autoregulatory gene network. The paper concludes in Section 5 with a discussion.

2 The KD-tree k-nearest neighbour algorithm

Suppose the true parameter is $\theta \in \mathbb{R}^d$. We wish to find a cheap approximation $\hat{\pi}_c(\theta)$ to $\pi(\theta)$ using a weighted average of the expensive values that have already been calculated. These expensive values might be of the true posterior $\pi(\theta_1), \ldots, \pi(\theta_m)$ (involving, for example, the numerical solution of a number of differential equations as in Section 4.2) or the expensive values might be $\hat{\pi}_s(\theta_1), \ldots, \hat{\pi}_s(\theta_m)$, unbiased stochastic estimates obtained as part of a pseudo-marginal MCMC algorithm (Section 4.1).

We will average the k-nearest neighbours. Naive use of a vector of n θ values and associated log-likelihoods, v, is expensive. While adding to the list takes $O(1)$ operations, searching the list for the k nearest neighbours to a particular point, θ^*, takes $O(n)$ operations.

For our applications, typically the dimension, d, of the problem is moderate: between 3 and 12. For very low dimensional problems an analogue of the quadtree (Finkel and Bentley, 1974) would be the most efficient approach, but the number of pointers from each node grows exponentially with dimension.

We therefore use a variation on the KD-tree (Bentley, 1975; Friedman et al., 1977).

Creation of the tree from n values takes $O(dn \log n)$ operations and requires storage of $O(n)$. For a balanced (see Section 2.7) tree the computation required to add an additional value is $O(d \log n)$, and to search for a nearest neighbour is also $O(d \log n)$.

The standard KD-tree has a single item of data (θ value and associated information) at each node; all items further down the tree from this node have been split at this node as described in detail in Section 2.4. The standard structure assumes that all of the θ values to be used are available before the tree is constructed, whereas our tree potentially grows with each new position at which the log-likelihood (or an unbiased estimate thereof) is evaluated. The most efficient 'splitting point' of any node of a tree is the median of all relevant values. Use of the median leads to a balanced tree where all end nodes would be at approximately the same depth. However we do not know the true median. We therefore separate our tree in to *leaf nodes* and *branch nodes*. Branch nodes provide the splitting information and leaf nodes, which occur at the base of the tree, store multiple data values. When a leaf node becomes full, it splits according to the median of the relevant data values it contains, rather than a true median, and becomes a branch node, creating two leaf nodes beneath it. The maximum size of a leaf node is defined so that the leaf median is unlikely to be 'too far' from the true median; the choice of this tuning parameter is investigated in Section 2.7 and in the simulation study in Section 4.1.

2.1 Preliminary run

To estimate the likelihood at some new point, we will take a weighted average of likelihoods, or estimated likelihoods at the k nearest points. If the likelihood varies more quickly with distance along some axes than along others then 'nearest' should be according to some alternative metric such as a Mahalanobis distance. However we also divide the KD-tree along hyperplanes which are perpendicular to one of the Cartesian axes. This leads to gross inefficiencies when there is a strong correlation between components of θ. For example if θ has a bivariate Gaussian distribution with marginal variances of 1 and correlation 0.999 then the hyperplane splits in a tree of depth 5, say, effectively partition the first principal component but fail to partition the second, so that two θ values which are reasonably close according to the Mahalanobis distance can be in different portions of the tree. This problem is exacerbated in higher dimensions.

The algorithm should, therefore, be more efficient if the the parameters are relatively uncorrelated and if the length scales in the direction of each axis are similar. A preliminary run of the MCMC algorithm allows us to find an approximate center $\hat{\mu}$, and variance matrix $\hat{\Sigma}$. For all θ we then define $\psi := \sqrt{\hat{\Sigma}}^{-1}(\theta - \hat{\mu})$ to normalise. Since this transformation gives a one-to-one mapping from θ to ψ, in what follows, for notational simplicity, we refer to the parameter values to be stored in the KD-tree as θ and implicitly assume that, in practice, the transformation has already been applied. The preliminary run should be of sufficient length that the sample approximately represents the gross relationships in the main posterior and hence that Euclidean distance is a reasonable metric for the transformed parameters; the preliminary chain does not need to have mixed thoroughly.

2.2 Tree preliminaries

The KD-tree stores a large number of vector parameter values, $\theta \in \Theta$, each with an associated vector of interest, $v_\theta \in \mathbb{V}$, in such a way that the time taken to either update the tree or to retrieve the information we require is logarithmic in the number of (θ, v_θ) pairs that are stored in the tree. In our case $v_\theta = (l_\theta, n_\theta) \in \mathbb{R} \times \mathbb{N}$ is the logarithm of the average of all estimates of the posterior at parameter values close to θ and the number of such estimates.

Associated with the vector of interest is a **merge** function $M : \Theta \times \mathbb{V} \times \Theta \times \mathbb{V} \to \mathbb{V}$, which combines the current vector of interest with the vector at a new value, θ^*.

2.3 Tree structure

The tree consists of **branch nodes** and **leaf nodes**. Each branch node has two children, each of which may either be a branch node or a leaf node.

Leaf nodes. Each leaf node stores up to $2b - 1$ (θ, v_θ) pairs, and the dimension, $d_{split} \in \{1, \ldots, d\}$, on which the node will split.

Branch nodes. Each branch node stores a component, $d_{split} \in \{1, \ldots, d\}$ on which it was split, and a corresponding scalar value θ_{split}, the split point. The left-hand child of the node contains (if it is a leaf node) or points to (if it is a branch) all (θ, v) pairs that have passed through this branch node and that have $\theta_{d_{split}} < \theta_{split}$. The right-hand child contains or points to all (θ, v) pairs with $\theta_{d_{split}} > \theta_{split}$. If $\theta_{d_{split}} = \theta_{split}$ it may be contained by the

right hand node or the left hand node.

Root node. The node at the very top of the tree is called the root node. By default the root node has $d_{split} = 1$. If there are fewer than $2b$ leaves in the tree then the root node is a leaf node, otherwise it is a branch node.

2.4 Adding data to a tree

When a leaf has $2b$ entries it immediately spawns two leaf node children by splitting all $2b$ entries along the component, d_{split}. The splitting point, θ_{split}, is the median of the $2b$ values for θ_{dsplit}. The parent node then becomes a branch, and the leaf nodes inherit $d_{split} = d_{split}^{(p)} \oplus 1$, where $d_{split}^{(p)}$ is the component on which the parent has just been split, and $\oplus$ represents regular addition except that $d \oplus 1 = 1$.

Some initial number, n_0 of data points are used to create a balanced tree using the standard recursive procedure given in the supplementary material (Appendix A.1). After this, a single new entry (θ, l_θ) is added to the tree by descending from the root node and, at each branch, comparing component $\theta_{d_{split}}$ with θ_{split} to choose the relevant child (if $\theta_{d_{split}} = \theta_{split}$ then the left child is chosen with probability 0.5). The new entry is added to the leaf node that is reached. If this leaf node still has fewer than $2b$ entries then the algorithm stops, otherwise the leaf-node becomes a branch and spawns two child leaf-nodes as described above.

2.5 Searching the tree

We apply a standard two-stage recursive algorithm to find the r nearest neighbours of a given point, θ^*, together with their distances from θ^*. We provide an overview of the algorithm below; the procedure is detailed in full in the supplementary material (Appendix A.2).

The algorithm first descends the tree to find the leaf node to which θ^* would belong, as described in Section 2.4. It then gradually ascends the tree from this node to the root node; after each ascent from a node to its parent a test is conducted to see whether the other child, that is the child of the current node from which the algorithm has *not* just ascended, or any of its offspring might hold a closer neighbour than the current k nearest. If this is the case then, before any further ascent can take place, the search descends down the tree via this

other child.

2.6 Restricting the growth of the tree

The tree will be used to obtain computationally cheap estimates of the posterior density. We would like to ensure that the accuracy of the approximations increases with the amount of information stored in the tree, whether the information arises from exact evaluations of the posterior or from stochastic estimates. We would also prefer the cost of obtaining this information to increase slowly, if at all, with the amount of information.

To reduce the total number of leaves we set a minimum distance between leaves, ϵ. For any new information, (θ^*, v_{θ^*}), to be added to the tree we first ascertain whether or not any pairs (θ, v_θ) exist with $||\theta^* - \theta|| < \epsilon$. If one or more such pair exists then the merge function, M, described in Section 2.2 is used to combine (θ^*, v_{θ^*}) with the nearest pair, providing a replacement value for v_θ; otherwise (θ^*, v_{θ^*}) is added to the tree as described in Section 2.4.

When each v_{θ^*} contains an exact evaluation of the posterior then the merge simply ignores the new information. However, when v_{θ^*} contains a stochastic estimate of the posterior some weighted average of the new estimate and of the current average will be more appropriate since, by the continuity of the posterior, for sufficiently small ϵ, $\pi(\theta^*) \approx \pi(\theta)$ for all θ^* such that $||\theta^* - \theta|| < \epsilon$. In particular therefore, for pseudo-marginal algorithms, we define

$$M_{PM}(\theta, [l_\theta, n_\theta], \theta^*, [l_{\theta^*}, 1]) := \left[\log\left[n_\theta e^{l_\theta} + e^{l_{\theta^*}}\right] - \log(n_\theta + 1),\ n_\theta + 1\right].$$

This vector replaces the previous $[l_\theta, n_\theta]$ vector and so is associated with the position θ.

Choice of merge distance

Consider a tree with n existing points, $\theta_1, \ldots \theta_n$ and to which it is proposed that a new point θ^*, chosen at random, will be added. We relate the merge distance, ϵ, to the probability, p_{keep}, that none of the existing points is within the ϵ ball of θ^*, so that θ^* will be added to the tree. This then provides a guide to setting ϵ itself.

Let B^*_ϵ be the ϵ ball around θ^*, define $N_{n,\epsilon}$ to be the number of the n existing points that are inside B^*_ϵ and consider $E_{n,\epsilon} = \mathbb{E}[N_{n,\epsilon}]$. The following is proved in the supplementary material (Appendix B).

Proposition 1. *If* $0 < E_{n,\epsilon} < 1$ *then* $1 - E_{n,\epsilon} < p_{keep} < e^{-E_{n,\epsilon}}$.

To use Proposition 1 we require an expression for $E_{n,\epsilon}$, and this depends on the distribution of $(\theta_1, \ldots, \theta_n, \theta^*)$. For tractability, and because it will often hold approximately with reasonably sized data sets, we suppose that the target is Gaussian, so that $\theta_i \sim N(\mu, \Sigma)$, marginally. This is then normalised (see Section 2.1) so that the following result (see again Appendix B for a proof) can be applied.

Proposition 2. *Let jointly distributed* $\theta_1 \sim N(0, I_d), \ldots, \theta_n \sim N(0, I_d)$*, be independent of* $\theta^* \sim N(0, I_d)$*. Then* $E_{n,\epsilon} = nF_{\chi^2_d}(\epsilon^2/2)$*, where* $F_{\chi^2_d}$ *is the cumulative distribution function of a* χ^2_d *random variable.*

In the simulation studies of Section 4 we choose ϵ such that $E_{n,\epsilon} = 0.5$, giving $0.5 < p_{keep} < 0.61$ and $\epsilon \approx \sqrt{2q_{\chi^2_d}(1/2n)}$, where $q_{\chi^2_d}$ is the quantile function of a χ^2_d random variable.

2.7 Ensuring that the tree remains balanced

We consider two mechanisms through which a KD-tree that is constructed on-line using an MCMC algorithm may become unbalanced, and for each problem we provide a solution.

When $d = 1$, there are two binary tree structures that allow for online rebalancing: the red-black tree and the AVL tree (e.g. Storer, 2002). Unfortunately there are no known algorithms for rebalancing a KD-tree online and so we consider an alternative which ensures that the KD-tree remains approximately balanced as at grows.

The splitting hyper-planes define a partition of Θ. Let the *node box* corresponding to a particular node be the (possibly unbounded) subset of Θ defined through the constraint at each splitting hyper-plane on the journey from the root node to the node in question, as described in Section 2.4.

Consider, informally, for any node box, an 'effective width' along a particular co-ordinate axis to be some representative width such as the standard deviation of the posterior restricted to the node box. Suppose that the MCMC chain is currently in a node box where an effective width along its splitting co-ordinate is δ. Now, imagine that at each iteration the chain barely moves compared to δ, and each jump proposal - which can give a new evaluation of the log-likelihood even if the chain does not move - is also small compared to δ. In this case all

of the new samples for a large number of iterations will descend to this one particular leaf node, which will fill up and then split. This split will not, however, be representative of the marginal median (in terms of the posterior) for the node box along the splitting axis. Hence, over the rest of the MCMC run, once the chain has moved on, there will be one side of the split which takes most of the future sample points and one side which takes very few of them, leading to an unbalanced tree.

Now suppose that a preliminary run of n_0 iterations has been carried out and that over this run the chain has been seen to mix reasonably across the posterior. Let us now construct a *balanced* tree from this run. Each leaf node will have b^* (or b^*+1) entries, for some $b^* \in \{b, b+1, \ldots, 2b-2\}$. Consider the node box of any specific leaf node in this tree. Since the chain has mixed reasonably, it should represent the posterior within this node box, and in particular (1) in the component over which the leaf node will split, the median should be reasonably approximated, and (2) in the main MCMC run, the chain should also cover this box in approximately b^* iterations or fewer; hence it will also cover any sub-divisions of the box in approximately b^* (or fewer) iterations.

The second reason a tree can become unbalanced is Monte Carlo error. A leaf splits after it has $2b$ entries by finding the median of $\theta_{d_{split}}$ over the $2b$ entries, but this sample median will not be the true median over the node-box. Table 1 provides, for 4 different values of $2b$, the probability that the estimated median will be at a true quantile which is outside the range shown. A tree with $2b \times 10^5$ iid entries was simulated for each value of $2b$ and each of two dimensions. The table also shows the mean depth of the leaf nodes and the range of depths of 99% of the leaf nodes and, in brackets, of all leaf nodes. Both aspects of Table 1 suggest that $2b = 20$ or $2b = 30$ should lead to a reasonably balanced tree with few leaf nodes requiring much more effort to reach than the majority of the leaf nodes and with little effect on the overall mean amount of effort required to reach a leaf node.

3 Adaptive MCMC algorithm

We briefly review the delayed-acceptance algorithm before describing our adaptive version. This is further extended to an adaptive pseudo-marginal version in Section 3.4.

2b	[0.4,0.6]	[0.3,0.7]	[0.2,0.8]	$d = 3$	$d = 10$
10	0.49	0.15	0.02	18.3, 14-23 (11-25)	18.4, 14-23 (11-28)
20	0.35	0.05	0.002	17.7, 15-21 (13-23)	17.7, 15-21 (12-23)
30	0.26	0.02	0.0002	17.5, 15-20 (14-22)	17.5, 15-20 (14-21)
40	0.19	0.007	0.00001	17.4, 15-19 (14-21)	17.4, 15-19 (13-20)

Table 1: Left: the probability that an estimated median from a sample of size $2b \in \{10, 20, 30, 40\}$ will be at a quantile outside of the range $[0.4, 0.6]$, $[0.3, 0.7]$ or $[0.2, 0.8]$. Right: tree depths when $2b \times 10^5$ independent entries are added sequentially to a tree: mean over all leaf nodes, range of the central 99% of leaf nodes and the maximum and minimum.

3.1 Delayed-acceptance algorithms

Given a current parameter value, $\theta \in \Theta$, the Metropolis-Hastings (MH) algorithm proposes a new value, θ^* from some density $q(\theta^*|\theta)$ and then accepts or rejects according to

$$\alpha_{MH}(\theta, \theta^*) := 1 \wedge \frac{\pi(\theta^*)q(\theta|\theta^*)}{\pi(\theta)q(\theta^*|\theta)}. \tag{1}$$

The delayed acceptance Metropolis-Hastings (daMH) algorithm utilises a cheap (deterministic or stochastic) approximation $\hat{\pi}_c$ in two stages. At Stage One, $\hat{\pi}_c$ is substituted for π in the standard MH acceptance formula:

$$\tilde{\alpha}_1(\theta, \theta^*) := 1 \wedge \frac{\hat{\pi}_c(\theta^*)\ q\left(\theta|\theta^*\right)}{\hat{\pi}_c(\theta)\ q\left(\theta^*|\theta\right)}, \tag{2}$$

A second accept/reject stage is applied to any proposals that pass Stage One and a proposal is only accepted if it passes both stages. The Stage Two acceptance probability is:

$$\tilde{\alpha}_2(\theta, \theta^*) := 1 \wedge \frac{\pi(\theta^*)\hat{\pi}_c(\theta)}{\pi(\theta)\hat{\pi}_c(\theta^*)}. \tag{3}$$

The overall acceptance probability, $\tilde{\alpha}_1(\theta, \theta^*)\tilde{\alpha}_2(\theta, \theta^*)$ ensures that detailed balance is satisfied with respect to π; however if a rejection occurs at Stage One then the expensive evaluation of $\pi(\theta)$ at Stage Two is unnecessary.

3.2 Adaptive, delayed-acceptance algorithm

As in Roberts and Rosenthal (2007, 2009) our adaptive kernel consists of a mixture of a fixed kernel and an evolving kernel. At each iteration, with a user-defined probability $\beta \in (0, 1)$,

the fixed kernel is selected, otherwise the adaptive kernel is used. Our fixed kernel is a standard Metropolis-Hasting kernel and our evolving kernel uses delayed-acceptance with an acceptance rate derived from a cheap approximation $\hat{\pi}_c(\theta^*)$ which is an inverse-distance-weighted average of the expensive evaluations of the true posterior at the k nearest neighbours to θ^* in the KD-tree at iteration n. After iteration $n-1$ let there have been i_{n-1} evaluations of the true posterior, π. Let these evaluations be at values $\theta^*_{i_1}, \ldots, \theta^*_{i_{n-1}}$. Our adaptive algorithm requires a sequence of probabilities, $\{p_i\}_{i \in \mathbb{N}}$, with

$$\lim_{i \to \infty} p_i = 0. \tag{4}$$

In practice, the algorithm proceeds until some n_{tot} iterations have been performed.

Algorithm 1: *adaptive-KD-tree, delayed-acceptance Metropolis-Hastings.*

1. With probability β go to Step 2 (MH) else go to Step 3 (da-MH).

2. **MH**: Propose θ^* from $q(\theta^*|\theta)$. Evaluate the expensive posterior, $\pi(\theta^*)$, and accept the proposal $(\theta \leftarrow \theta^*)$ with probability given by (1); otherwise reject the proposal $(\theta \leftarrow \theta)$. Go to Step 4.

3. **da-MH**: Propose θ^* from $q'(\theta^*|\theta)$.

   (a) **Stage 1**: Evaluate $\hat{\pi}_a(\theta^*)$ using the current KD-tree; with probability $\tilde{\alpha}_1(\theta, \theta^*)$ as defined in (2) proceed to Step 3b (Stage 2); otherwise reject the proposal $(\theta \leftarrow \theta)$, set $i_n = i_{n-1}$, go to next iteration.

   (b) **Stage 2**: Evaluate $\pi(\theta^*)$; accept the proposal $(\theta \leftarrow \theta^*)$ with probability $\tilde{\alpha}_{2,s}(\theta, \theta^*)$ as defined in (3); otherwise reject the proposal $(\theta \leftarrow \theta)$. Go to Step 4.

4. Set $i_n = i_{n-1} + 1$; add $(\theta^*, \pi(\theta^*))$ to a list of recently-evaluated parameter/posterior pairs; with probability p_{i_n} transfer all pairs from this list to the KD-tree; go to next iteration.

3.3 Delayed acceptance random walk Metropolis

It remains to choose the proposal mechanisms, q and q'. The Random Walk Metropolis (RWM) is a MH algorithm where $q(\theta^*|\theta) = q(||\theta^* - \theta||)$ for some suitable norm, and hence

$q(\theta^*|\theta)$ and $q(\theta|\theta^*)$ cancel in the acceptance ratios (1) and (2). We consider the standard choice of

$$q(\theta^*|\theta) = N(\theta^*; \theta, V) \quad \text{and} \quad q'(\theta^*|\theta) = q^{\xi}(\theta^*|\theta) := N(\theta^*; \theta, \xi^2 V), \tag{5}$$

where $N(\cdot; \theta, V)$ denotes a multivariate Gaussian density with mean θ and variance V and where V has been chosen so as to approximately optimise the efficiency of the standard RWM algorithm.

As we shall discover, the cheap approximation, $\hat{\pi}_c$ is reasonably accurate. As ξ increases from 1 the overall acceptance rate and, in particular, the Stage One acceptance rate, α_1, can decrease quite substantially. If all computational expense is negligible except for the evaluation of the true posterior then for a given amount of computational effort, the number of evaluations of the expensive posterior remains approximately constant, although the total number of iterations of the algorithm increases in proportion to the reciprocal of α_1. However, as each proposed jump is larger, moves which are accepted at Stage Two are typically larger. Thus, provided the Stage Two acceptance rate does not decrease too drastically, the mixing of the algorithm (in terms of movement per CPU second) can, and often does, actually increase for intermediate values of ξ. This heuristic has been noted before (e.g. Christen and Fox, 2005; Banterle et al., 2015) and also applies for the delayed-acceptance pseudo-marginal RWM. A rigorous analysis of the behaviour of these algorithms as a function of the scaling is provided in Sherlock et al. (2015).

3.4 Adaptive, delayed-acceptance, pseudo-marginal algorithm

We first overview pseudo-marginal Metropolis-Hastings algorithms and then describe the adjustments to the set-up required for the pseudo-marginal version of our algorithm. The algorithm itself is provided in the supplementary material (Appendix C).

The pseudo-marginal algorithm uses a non-negative stochastic estimator $\hat{\pi}_s(\theta; Z)$ of the posterior $\pi(\theta)$, where Z is a collection of random variables whose distribution may, and usually does, depend on θ. Crucially, we require $\mathbb{E}\left[\hat{\pi}_s(\theta; Z)\right] = c\pi(\theta)$, where c is fixed and non-negative. We may therefore rewrite $\hat{\pi}_s(\theta)$ as $\pi(\theta)W$ with $W \in \mathcal{W} \subseteq [0, \infty)$ sampled from some density $q_\theta(w)$, and

$$\mathbb{E}\left[W\right] = \int_0^\infty w q_\theta(w) \, \mathrm{d}w = c. \tag{6}$$

The pseudo-marginal MH (PsMMH) algorithm is simply a Metropolis-Hastings Markov chain acting on the extended statespace $\Theta \times \mathcal{W}$ with a target of

$$\tilde{\pi}(\theta, w) = \frac{1}{c}\pi(\theta)q_\theta(w)w. \tag{7}$$

This has the required marginal for θ by (6). Detailed balance is ensured with respect to this target by setting the probability for (θ^*, w^*) being accepted to:

$$\alpha_{PM}([\theta, W], [\theta^*, W^*]) := 1 \wedge \frac{\hat{\pi}_s(\theta^*)q(\theta|\theta^*)}{\hat{\pi}_s(\theta)q(\theta^*|\theta)} = 1 \wedge \frac{\pi(\theta^*)q(\theta|\theta^*)W^*}{\pi(\theta)q(\theta^*|\theta)W}. \tag{8}$$

When delayed acceptance is implemented with a pseudo-marginal framework, the Stage One acceptance probability is exactly as in (2). In Stage Two the true posterior in (3) is replaced with the realisation from the unbiased estimator:

$$\tilde{\alpha}_{2,PM}(\theta, \theta^*) := 1 \wedge \frac{\hat{\pi}_s(\theta^*)\hat{\pi}_c(\theta)}{\hat{\pi}_s(\theta)\hat{\pi}_c(\theta^*)}. \tag{9}$$

The kernel $\tilde{P}$ is now a fixed PsMMH kernel on $\Theta \times \mathcal{W}$ and $\{\tilde{P}_\gamma\}_{\gamma \in \mathcal{G}}$ is now a set of pseudo-marginal kernels on $\Theta \times \mathcal{W}$. The common stationary density of all kernels is now given in (7), so, very importantly, all kernels use the same mechanism for generating the estimate $\hat{\pi}_s(\theta^*)$ of the posterior at the proposed value for θ.

The algorithm proceeds as for the non-pseudo-marginal version except that $\hat{\pi}_s$ is substituted for π in (1) (fixed kernel), and (3) is replaced with (9) (evolving, DA kernel). Naturally, instead of storing evaluations of π the KD-tree now stores realisations of the unbiased approximation, $\hat{\pi}_s$.

3.5 Theory and guidance

We show that, subject to conditions, our algorithms (with general proposals, q) are ergodic. We also provide guidance on choosing the probability of using the fixed kernel, β, and on the number of iterations for which the algorithm should be run.

Define $\alpha_{MH}(\theta, \theta^*)$ as follows. For the adaptive KD-tree daMH algorithm of Section 3.2 $\alpha_{MH}(\theta, \theta^*)$ is the acceptance probability for the fixed kernel as given in (1). For the pseudo-marginal version of the algorithm in Section 3.4 it is the acceptance probability for an hypothetical, idealised version of the fixed, pseudo-marginal kernel where the posterior is known exactly, up to a fixed multiplicative constant: $\alpha_{MH}(\theta, \theta^*) = \alpha_{PM}([\theta, 1], [\theta^*, 1])$,

where α_{PM} is defined in (8). We require a minorisation condition and, for the daPsMMH algorithm, and additional assumption of uniformly bounded weights. These assumptions are discussed in in the supplementary material (Appendix D), where their main consequence, Theorem 1 is proved.

Assumption 1. There is a density $\nu(\theta)$ and $\delta > 0$ such that $q(\theta^*|\theta)\alpha_{MH}(\theta, \theta^*) \geq \delta\nu(\theta^*)$ for all $\theta \in \Theta$.

Assumption 2. The support for W is uniformly (in θ) bounded above by some $\overline{w} < \infty$.

Theorem 1. *Subject to Assumption 1 the adaptive KD-tree daMH algorithm of Section 3.2 is ergodic. The adaptive KD-tree daPsMMH algorithm of Section 3.4 is ergodic subject to Assumptions 1 and 2.*

Now consider the specific, scaled, proposal q^ξ defined in (5), where increasing ξ decreases the Stage One acceptance rate, α_1. The algorithm may only accept a proposal after a computationally-expensive evaluation (of π for daMH, or $\hat{\pi}_s$ for daPsMMH). Thus, if the probability, β, that the non-DA kernel will be chosen is unaltered, then as $\alpha_1 \to 0$ nearly all of the expensive evaluations will be by the fixed, non-DA kernel, and the relative contribution from the DA kernels will unintentionally dwindle to zero.

Decreasing β in proportion to α_1 would fix the fraction of all expensive evaluations that are by the DA kernel; however with a smaller β the chain can no longer be guaranteed to be as close to π after the same, fixed number of iterations. Theorem 2, which is stated and proved in the supplementary material (Appendix D), shows that, with $\beta \propto \alpha_1$, as α_1 decreases the total number of iterations, n, of the algorithm should be increased so as to maintain the expected total number of expensive evaluations of the posterior, $\mathbb{E}\left[I_n^\xi\right]$, and that $\mathbb{E}\left[I_n^\xi\right]$ can be set so as to maintain any given upper bound on the total variation distance between the chain and π whatever the value of α_1.

Fixing $\mathbb{E}\left[I_n^\xi\right]$ approximately fixes the expected overall CPU cost. Furthermore, adaptation can only occur when the expensive posterior is evaluated, and it occurs with a fixed set of probabilities that depend on i and not on the iteration number. So, fixing $\mathbb{E}\left[I_n^\xi\right]$ also approximately fixes the expected number of adaptation occurrences.

4 Simulation Studies

In this section we evaluate the empirical performance of the proposed da-PsMMH and da-MH algorithms by considering two examples based upon Markov jump processes (MJPs). The first example (Section 4.1) arises from the Lotka-Volterra system of predator-prey interactions (e.g. Boys et al., 2008). Since the marginal likelihood is intractable, we apply the adaptive da-PsMMH scheme and compare its performance over a range of tuning parameter choices with that of an optimised PsMMH scheme. The second example (Section 4.2) arises from the autoregulatory network proposed by Golightly and Wilkinson (2005). Given the size and complexity of this system, a linear noise approximation (LNA) (van Kampen, 2001) (see also Appendix F of the supplementary material), of the corresponding Markov jump process is taken to be the inferential model of interest. Following the algorithm of Fearnhead et al. (2014) (see Appendix F.1), the marginal likelihood under this model is tractable, but involves the solution of a system of 14 coupled ordinary differential equations (ODEs), which can be time consuming. We therefore apply the da-MH algorithm and compare its performance to a simple MH scheme without delayed acceptance.

Both MJPs are described through a set of r reactions between p different species, $\mathcal{X}_1, \ldots, \mathcal{X}_p$. The hazard rate of each reaction depends on the current species numbers, $X_1, \ldots, X_p$ via an assumption of mass-action kinetics with unknown reaction rate constants $\nu_1, \ldots, \nu_r$; for further details regarding the construction of MJP representations of reaction networks we refer the reader to Wilkinson (2012). Tables E.1 and E.2 in Appendix E.1 list the reactions and associated hazards for each example.

For the Lotka-Volterra model, the Gillespie algorithm (Gillespie (1977)) was applied, using parameter values taken from Wilkinson (2012), to generate a skeleton path comprising 51 values of X_t at integer times in the interval $[0, 50]$. For the autoregulatory system, the LNA itself was used, with parameter values taken from Golightly and Wilkinson (2011), to generate two skeleton paths containing, respectively, 101 and 201 values of X_t at evenly-spaced times covering the intervals $[0, 100]$ and $[0, 1000]$. All skeletons were then corrupted with Gaussian noise to form the data sets on which inference was performed:

$$Y_t | X_t = x_t \sim N(x_t, D), \tag{10}$$

where D is a diagonal matrix with diagonal entries $\sigma_1^2, \ldots, \sigma_p^2$. In the autoregulatory ex-

ample we refer to the data sets with 101 and 201 observations as $\mathcal{D}_1$ and $\mathcal{D}_2$ respectively. Appendix E.1 provides details of the initial conditions and parameter values for each simulation as well as for the variances of the corrupting Gaussian noises.

For inference, in both cases, for simplicity, the initial state of the system is fixed at its (known) true value. As all of the parameters are strictly positive we therefore consider a logarithmic transformation so that the parameter vector of interest is $\theta = (\log(\nu_1), \ldots, \log(\nu_r), \log(\sigma_1), \ldots, \log(\sigma_p))$. In both examples, individual components of θ for which inference is performed are given independent Uniform $U(-8, 8)$ priors. For the Lotka-Volterra model, $r = 2$ and $p = 2$ so that $\dim(\theta) = 4$. For the autoregulatory system, $p = 4$ and, as discussed in Appendix E.1, we perform inference on $r = 6$ rate constants, giving $\dim(\theta) = 10$.

Random-walk proposals of the form (5) are used for the fixed and the adaptive kernels in both examples. The fixed kernels used a proposal variance of $V_{fixed} = \lambda\hat{\Sigma}$, where $\hat{\Sigma}$ is the sample variance from an initial, pilot run using a fixed, non-delayed-acceptance kernel, and λ is chosen to optimise the efficiency of the fixed kernel. The corresponding adaptive kernels use a proposal variance of $\xi^2 V_{fixed}$, with $\xi > 0$ a tuning parameter.

In each example we take the probability of adding to the tree at iteration n to be

$$p_{i_n} = (1 + ci_n)^{-1}. \tag{11}$$

For each example, the computational cost of an evaluation of the expensive stochastic approximation was estimated from the initial training run, and this provided an estimate of the number of expensive evaluations, $\hat{i}$, that would fit within the computational budget. Using the guidelines in Section 2.6, the parameter ϵ was chosen with a desire that points still be as likely as not to be added to the tree after $\hat{i}/2$ evaluations had already been added. For both systems this had the effect of limiting the overall tree size to around four times the size of the initial training set.

4.1 Discretely observed Markov jump process

To implement the adaptive da-PsMMH scheme described in Algorithm 1, we used a bootstrap particle filter with m particles (Andrieu et al., 2010) to obtain each value of $\hat{\pi}_s(\cdot)$. Both m and V_{fixed} were chosen so as to optimise the efficiency of the fixed kernel; see the supplementary material (Appendix E.2). We initialised the KD-tree using the first 10^4 evaluations of the

expensive posterior in the pilot run. For all experiments, we assumed a fixed computational budget of 10^4 seconds, which equated to approximately $\hat{i} = 40000$ evaluations of the expensive posterior and, thus, a maximum tree size of five times the initial size.

To assess the effect of scaling ξ on the overall and relative (to PsMMH) efficiency of da-PSMMH we fixed the number of leaf nodes in the KD-tree upon splitting to be $2b = 20$, the number of nearest neighbours to be $k = b = 10$ and took the parameter controlling the rate of adaptation to be $c = 0.001$ to give around a 5% chance of adaptation after half the computational budget. We then ran the algorithm for values of $\xi \in [1, 4]$, following the practical advice of Section 3.5 by choosing $\beta \propto \hat{\alpha}_1$ with $\beta = 0.05$ for $\xi = 1$. Firstly, the sampled posterior values are consistent with the ground truth parameter values that produced the data (see Figure G.1 in the supplementary material for the marginal posterior distributions for a typical run). Figure 1 shows the effect of scaling on minimum effective sample size (mESS) over each parameter chain relative to that obtained under an optimally tuned PsMMH scheme (with an acceptance rate of 9.9% and an mESS of 528) and the effect of scaling on the Stage 1 acceptance probability. The (scaled) values of β used for each run are also shown. Figure 1 suggests that for the values of ξ considered, $\xi = 3$ is optimal in terms of mESS and gives an improvement on overall efficiency over PsMMH of a factor of 6.8. Even simply taking the same scaling as PsMMH still gives a 3-fold increase in efficiency of da-PsMMH over PsMMH.

To assess the effect of adaptation on the performance of the algorithm we fixed $\xi = 3$, $k = b = 10$ and performed runs with $c \in := \{0.0001, 0.001, 0.01, \infty\}$ with $c = \infty$ representing no adaptation. Table 2 summarises our findings. At $\hat{i}/2$ iterations, these values of c correspond to an expected number of expensive evaluations before adaptation occurs of approximately $\{3, 20, 200, \infty\}$, respectively. The larger the pause between adaptations, the less accurate the tree is between adaptations, and while 200 new evaluations is small compared with 10000 or more existing evaluations, it must be remembered that the most recent evaluations will be from a similar part of the state space to the current position and so will be among the most relevant. The reduction in accuracy especially in new, low-density regions, increases the Stage 1 acceptance rate and decreases the Stage 2 acceptance rate giving an overall reduction in statistical efficiency. Moreover, the increase in the Stage 1 rate results in a larger number of expensive posterior evaluations.

With $c = \infty$ the algorithm runs with no adaptation and just uses the initial training set of $10,000$ posterior evaluations. In this case (last row of Table 2) performance is better than for the simple RWM algorithm but worse than for all of the cases that allow adaptation, providing clear evidence of the importance of adaptation for our algorithm. Further, the more slowly $p_i \downarrow \infty$, the more efficient the algorithm.

We also explore the sensitivity of our method to the choice of the number of leaf nodes in the KD-tree upon splitting ($2b$) and the number of nearest neighbours (k). We fixed $\xi = 3$, $c = 0.001$ and took $2b \in \{4, 10, 20, 30\}$ and $k \in \{2, 5, 10, 15\}$. Table G.3 in Appendix G shows empirical performance for each (k, b) combination considered. Consistent with our findings in Section 2.7, increasing b increases the efficiency until $2b = 20$, but there is little difference when moving from $2b = 20$ to $2b = 30$. Fixing $2b$ and varying k suggests that $k = 5$ is optimal in terms of mESS. Further discussion can be found in the supplementary material (Appendix G). Finally we examine the gain in overall efficiency by using a KD-tree as a storage and look-up method over simply storing posterior evaluations in a list. Running the da-PsMMH scheme with the optimal values ($\xi = 3$, $c = 0.001$ and $k = 5$) gave an mESS of 2009. Thus, in this example, using a KD-tree increases overall efficiency over a naive approach by a factor of 1.9. Naturally, increasing the computational budget (and therefore the number of posterior evaluations to be stored) will increase the advantage of the KD-tree.

c	Tree Size	Mean depth	depth range	$\hat{\alpha}_1$	$\hat{\alpha}_2$	mESS	Rel. mESS
0.0001	41078	11.82	10-14	0.00772	0.339	3845	7.28
0.001	40256	11.79	10-14	0.00915	0.276	3591	6.80
0.01	43248	12.04	10-15	0.0121	0.204	2464	4.67
∞	10000	9.69	9-10	0.0175	0.136	1829	3.46

Table 2: Effect of rate of adaptation c. Final tree size, mean leaf node depth, depth range, empirical stage 1 and 2 acceptance rate, minimum effective sample size (mESS) and relative mESS.

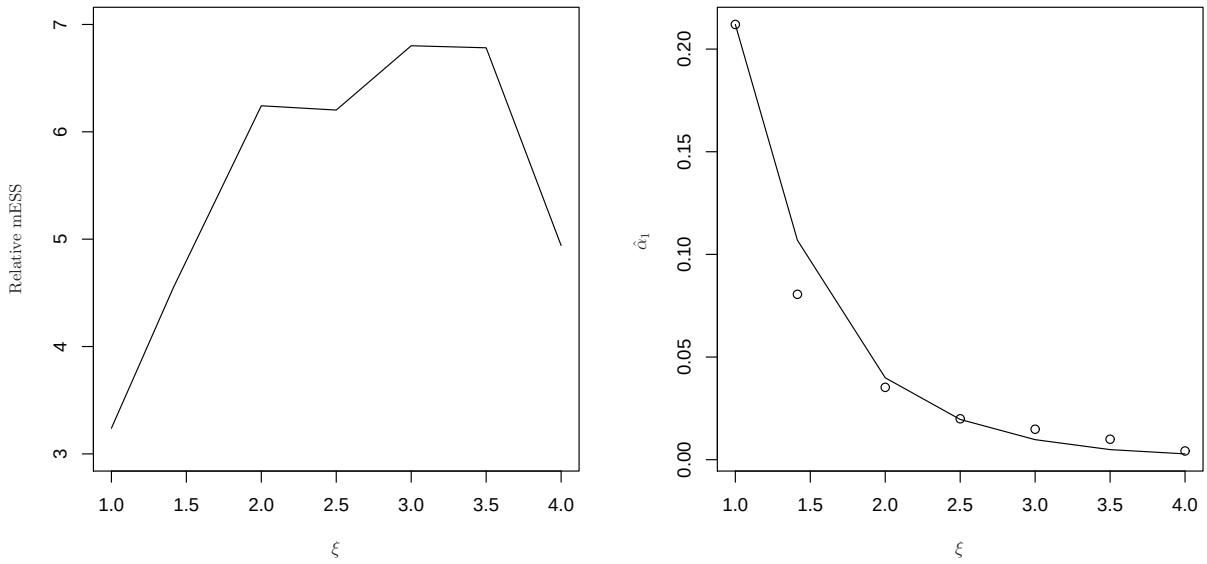

Figure 1: Left panel. Minimum effective sample size (mESS) relative to optimised PsMMH, against scaling. Right panel. Empirical stage 1 acceptance probability $\hat{\alpha}_1$ against scaling. The points represent $\hat{\alpha}_1(\xi = 1)\beta(\xi)/\beta(\xi = 1)$ and show that $\beta(\xi)$ was scaled in proportion to $\alpha_1(\xi)$.

4.2 Discretely observed ODE system

The LNA gives a Gaussian model for X_t which when coupled with the Gaussian observation model above, permits a tractable form for the marginal likelihood which we denote by $\pi(y_{1:n}|\theta)$. An algorithm for evaluating the marginal likelihood, and therefore the posterior (up to proportionality) under the LNA, can be found in the supplementary material (Appendix F.1). Executing one iteration of the algorithm requires calculation of a full numerical solution of the ODE system (F.1) over $[0, 100]$ or $[0, 1000]$. Our implementation uses standard routines from the GNU scientific library, specifically the explicit embedded Runge-Kutta-Fehlberg $(4, 5)$ method. We limit the computational cost of these calculations by applying the da-MH scheme.

The pilot run for each dataset was of 3×10^4 iterations. We initialised the KD-tree with all 3×10^4 evaluations of the expensive posterior obtained from the initial pilot run. For all

experiments, we assumed a fixed computational budget of 5×10^3 seconds, which equated to approximately $\hat{i} = 120000$ evaluations of the expensive posterior and, thus, a maximum tree size of five times the initial size. Following the findings of Section 4.1 we initially set the adaptation rate to $c = 0.001$, and set $2b = 20$ and $k = 5$.

Firstly, Figure G.3 in Appendix G shows that the sampled parameter values are consistent with the ground truth, with a decrease in uncertainty when using more observations. With dataset $\mathcal{D}_2$ the LNA equations must be solved over a longer time period than for $\mathcal{D}_1$, and the steps of the algorithm for calculating the marginal likelihood (in Appendix F.1) must be executed twice as many times. Consequently Figure 2 (left panel) shows that the minimum effective sample size (mESS) obtained under da-MH is smaller when using dataset $\mathcal{D}_2$. However, mESS relative to the same quantity under MH is increased when using $\mathcal{D}_2$, since the cost of evaluating the KD-tree is unchanged (for a fixed tree size). The optimal scaling ξ (for the values considered) for each scheme is reported in Table 3. We also report output of additional runs with $c \in := \{0.0001, 0.001, \infty\}$. An optimally tuned da-MH scheme (with $c = 0.001$) gives an increase in overall efficiency of a factor of 3.2 when using $\mathcal{D}_1$ and 4.4 when using $\mathcal{D}_2$. When $c = \infty$ (representing no adaptation), we see an increase in Stage 1 acceptance rate and a decrease at Stage 2. The resulting decrease in empirical performance provides further evidence of the importance of adaptation. Finally, we again note that the algorithm performs best with a very low value of c whilst still providing posterior output consistent with $c = 0.001$ (results not shown).

5 Discussion

We have presented standard and pseudo-marginal versions of an adaptive, delayed-acceptance random walk Metropolis algorithm. The delayed-acceptance (DA) step is generic, estimating the posterior using an inverse-distance-weighted average of the k-nearest previous evaluations of the posterior, and the search for these neighbours is made fast by storing a subset of previous evaluations in a customised version of a KD-tree. The kernel is a mixture of a fixed (non-adaptive, non-DA) kernel and an adaptive DA kernel. Easy-to-use `C` code for creating a KD-tree and storing and retrieving values from it is provided alongside this article.

We have shown that our algorithm is ergodic, subject to conditions. Furthermore, as the

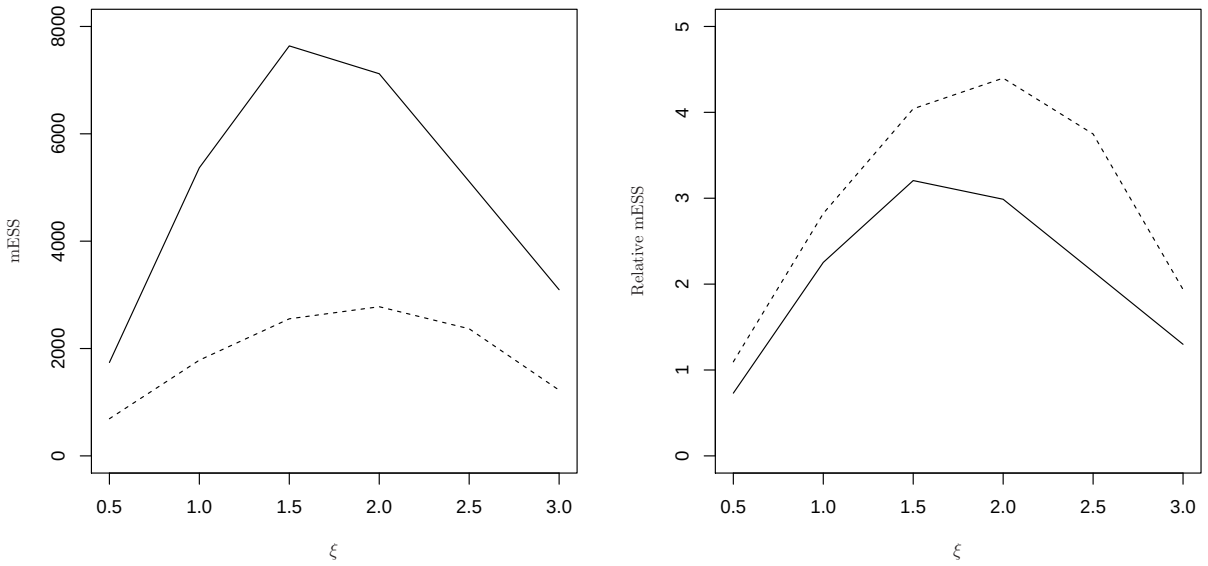

Figure 2: Left panel. Minimum effective sample size (mESS) from da-MH against scaling. Right panel. Minimum effective sample size from da-MH relative to the same quantity from optimised MH, against scaling. For each panel, the solid and dashed lines indicate output using datasets $\mathcal{D}_1$ and $\mathcal{D}_2$ respectively.

scaling of the RWM proposal in the DA kernel is increased, the probability of choosing the fixed kernel should be decreased in proportion to the Stage One acceptance rate of the DA kernel and the total number of iterations should be scaled so that the expected number of evaluations of the expensive posterior remains constant or, equivalently, so that the total computational budget remains fixed.

Pseudo-marginal and non-pseudo-marginal versions of the methodology were applied, respectively, to synthetic data generated from a discretely observed Lotka-Volterra system and an ODE model of species dynamics in an autoregulatory gene network. In these two examples, our proposed scheme outperforms the standard scheme by factors of approximately 7 and 4 for, respectively.

In both of our examples the algorithm was more efficient the more slowly the adaptation probability approached zero, suggesting that it might be best to simply add each new eval-

Algorithm	ξ	c	$\hat{\alpha}_1$	$\hat{\alpha}_2$	mESS	Rel. mESS
	$\mathcal{D}_1$ (101 obs. on $[0, 100]$)					
MMH	1.0	–	0.225	1.000	2383	1.00
da-MMH	1.5	0.0001	0.141	0.482	7699	3.23
	1.5	0.001	0.131	0.480	7638	3.21
	1.5	∞	0.171	0.366	4530	1.90
	$\mathcal{D}_2$ (201 obs. on $[0, 1000]$)					
MMH	1.0	–	0.2291	1.000	632	1.00
da-MMH	2.0	0.0001	0.0455	0.394	2996	4.74
	2.0	0.001	0.0390	0.338	2779	4.40
	2.0	∞	0.0978	0.165	1485	2.35

Table 3: Algorithm, optimal scaling (ξ), adaptation rate parameter (c), empirical stage 1 ($\hat{\alpha}_1$) and 2 ($\hat{\alpha}_2$) acceptance rate, minimum effective sample size (mESS) and relative mESS.

uation of the expensive posterior directly to the KD-tree directly rather than storing them in a queue. Forcing the limit of the adaptation probability to be 0 ensures the diminishing adaptation condition (see, e.g., Theorem 5) is satisfied; diminishing adaptation is itself one of the key conditions required for ergodicity and the concern would be that by removing this direct constraint the algorithm would no longer be ergodic. Whilst this seems likely to be the case in general, Theorem 3 in the supplementary material (Appendix D.5.1) shows that for a variation of Algorithm 1 on a compact state space it is possible to adapt after (at worst) every other expensive iteration and still remain ergodic.

We have focussed on making the cheap approximation to the posterior adaptive by intermittently updating the KD-tree. The covariance matrix of our random walk proposal could have been made adaptive by intermittently updating the covariance matrix of the random walk proposal so that it uses all entries in the chain to date (e.g. Roberts and Rosenthal, 2009; Sherlock et al., 2010). This could even be made 'local', using only the k-nearest neighbours in the KD-tree. It might also be possible to adaptively update the scaling of the proposal; however the mechanism to use is less obvious since, unlike in (e.g. Andrieu and Thoms, 2008; Sherlock et al., 2010; Vihola, 2012), there is no single optimal acceptance rate

for our algorithm. Such adaptations would be a distraction to our main innovation and have not been implemented.

As we were revising this article, Kostov and Whiteley (2016) proposed an algorithm for estimating the variance of the estimator of the likelihood that comes from the particle filter. Although we do not pursue it here, this opens up the possibility of weighting each estimate of the likelihood according to the reciprocal of its variance as well as its distance from the proposed θ^*.

5.1 Alternatives to k-nearest neighbours

As mentioned in Section 1, there are similarities between the non-pseudo-marginal version of our algorithm and that of Conrad et al. (2014) (henceforth denoted CMPS). Here we discuss the approach of CMPS, highlighting both similarities to and differences from our algorithm. We also consider a general GP alternative to our k-NN implementation.

CMPS fit local linear-, quadratic- and Gaussian-process (GP)-based models in neighbourhoods of candidate values, and at any given point in time their algorithm targets the approximate posterior rather than the true posterior. However, the accuracy of the approximation is continually assessed and improved by carefully choosing further local points as the algorithm proceeds so that, asymptotically, the algorithm targets the true posterior distribution. Between adaptations our algorithm targets the true posterior distribution, but it is perturbed every time an adaptation occurs. Thus our algorithm also asymptotically targets the true posterior distribution. Both algorithms also use an increasing set of evaluations of the true posterior. CMPS chooses the next evaluation of the true posterior by design whereas our algorithm is "opportunistic" and potentially adds the value at each new point evaluated; however it is an intelligent opportunist in the detail of how it deals with new points which are very close to existing points.

Our algorithm could be extended to fitting local planes, quadratics or GPs to the k-nearest neighbours in a similar manner to CMPS, however the weighted average is simpler, quicker, and for it to be a sensible approach to take it only requires the posterior to be bounded and continuous, rather than needing additional constraints. When it is the exact log-likelihood that we are approximating these more complex local models might lead to an improvement in the accuracy of the surrogate, at the small expense of possibly having to

use more nearest neighbours to fit. In the pseudo-marginal case, however, the efficiency will depend on the variance in the log-likelihood estimates and the true changes in log-likelihood. If the former outweighs the latter then there is little to be gained as the extra computation may not lead to additional accuracy. The cost of re-estimating the GP hyper-parameters as new training data become available is discussed in the final paragraph of this section. One final difference between the two approaches is that CMPS focusses on the non-pseudo-marginal case whereas our algorithm is applied in both pseudo-marginal and non-pseudo marginal settings.

Our likelihood estimate at a proposed point, θ', is a weighted average of the likelihoods at the k-nearest neighbours. An alternative approach would be to use a Gaussian process (GP) fitted to the set of $(\theta, \log p(y|\theta))$ pairs currently in the kd-tree. GPs have been used to approximate the log-likelihood previously. For example in Rasmussen (2003) and Fielding et al. (2011) a GP provides a cheap surrogate for Hamiltonian Monte Carlo calculations. Alterntively, in the context of ABC-MCMC, Wilkinson (2014) uses a Gaussian process to model the logarithm of the ABC approximate likelihood function, while Meeds and Welling (2014) approximate the joint synthetic likelihood at the current and proposed point using independent GPs for each summary statistic.

As with our approach, the final point estimate from a GP is a weighted average of existing values. However, by estimating the parameters of the GP one may represent the scales of variability more accurately and so obtain more accurate point estimates than with our inverse-distance weighting approach. The GP model also supplies an estimate of the uncertainty in the point estimate of the log-likelihood.

As in Conrad et al. (2014), the estimate of uncertainty allows for a choice of new training points to minimise the variance in some region of interest, with the potential of a large reduction in overhead. Algorithms such as those in Wilkinson (2014) and Meeds and Welling (2014) which use this approach target an approximation to the true posterior and an accurate GP approximation is essential for the algorithm to be useful. By design, our algorithm overlays a standard MH or PMMH algorithm and it automatically targets the true posterior; an inaccurate approximation reduces the mixing efficiency but does not invalidate the algorithm. Again, since our algorithm overlays a standard MH algorithm the true likelihood (or an unbiased estimate) must be evaluated at every accepted point and it is not imme-

diately obvious how estimates of uncertainty might be used to circuvent this requirement while maintaining the true posterior as the target.

Finally, estimating the hyper-parameters of a GP is computationally very costly, and this estimation should be repeated as the training data set grows. For this reason and because of the difficulty in identifying $d(d+1)/2$ kernel range parameters, GP methods often use a diagonal covariance structure for the kernel, limiting the flexibility. Our approach of using the covariance matrix of the sample from an initial training run to provide a map to a new parameter space where Euclidean distance is appropriate has a similar flavour to the pragmatic approach of using an initial training sample to fit the GP hyper-parameters and then keeping them fixed; alternatively, see Shen et al. (2006) for a partial solution.

Acknowledgements

The authors thank Krysztof Latuszynski for a very helpful discussion with regard to Theorem 3, and the Associate Editor and referee for useful suggestions that have improved the clarity of the paper.

Supplementary Material

Computer Code

Cprograms.zip Contains a generic `C` implementation of KD-tree storage and look-up as well as code for implementing the inference algorithms described in the paper.

Appendices

supplementary.pdf Contains appendices describing standard operations on a KD-tree, proofs of Propositions 1 and 2, the daPsMMH algorithm, proofs of Theorems 1, 2 and 3, the LNA and the construction of the corresponding marginal likelihood and additional plots from the simulation study.

References

Andrieu, C., Doucet, A., and Holenstein, R. (2010). Particle Markov chain Monte Carlo methods (with discussion). *J. R. Statist. Soc. B.*, 72(3):1–269.

Andrieu, C. and Roberts, G. O. (2009). The pseudo-marginal approach for efficient Monte Carlo computations. *The Annals of Statistics*, 37:697–725.

Andrieu, C. and Thoms, J. (2008). A tutorial on adaptive MCMC. *Stat. Comput.*, 18(4):343–373.

Banterle, M., Grazian, C., Lee, A., and Robert, C. P. (2015). Accelerating Metropolis-Hastings algorithms by delayed acceptance. `http://arxiv.org/abs/1406.2660`.

Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517.

Bliznyuk, N., Ruppert, D., Shoemaker, C., Regis, R., Wild, S., and Mugunthan, P. (2008). Bayesian calibration and uncertainty analysis for computationally expensive models using optimization and radial basis function approximation. *Journal of Computational and Graphical Statistics*, 17:270–294.

Boys, R. J., Wilkinson, D. J., and Kirkwood, T. B. L. (2008). Bayesian inference for a discretely observed stochastic-kinetic model. *Stat. Comput.*, 18:125–135.

Christen, J. A. and Fox, C. (2005). Markov chain Monte Carlo using an approximation. *Journal of Computational and Graphical Statistics*, 14:795–810.

Conrad, P. R., Marzouk, Y. M., Pillai, N. S., and Smith, A. (2014). Accelerating asymptotically exact MCMC for computationally intensive models via local approximations. `http://arxiv.org/abs/1402.1694`.

Cui, T., Fox, C., and O'Sullivan, M. J. (2011). Bayesian calibration of a large-scale geothermal reservoir model by a new adaptive delayed acceptance Metropolis Hastings algorithm. *Water Resources Research*, 47:W10521.

Fearnhead, P., Giagos, V., and Sherlock, C. (2014). Inference for reaction networks using the Linear Noise Approximation. *Biometrics*, 70:457–466.

Fielding, M., Nott, D. J., and Liong, S.-Y. (2011). Efficient MCMC schemes for computationally expensive posterior distributions. *Technometrics*, 53:16–28.

Finkel, R. A. and Bentley, J. L. (1974). Quad trees - a data structure for retrieval on composite keys. *Acta Informatica*, 4:1–9.

Friedman, J. H., Bentley, J. L., and Finkel, R. A. (1977). An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3(3):209–226.

Gillespie, D. T. (1977). Exact stochastic simulation of coupled chemical reactions. *J. Phys. Chem.*, 81:2340–2361.

Golightly, A., Henderson, D. A., and Sherlock, C. (2015). Delayed acceptance particle MCMC for exact inference in stochastic kinetic models. *Statistics and Computing*, 25:1039–1055.

Golightly, A. and Wilkinson, D. J. (2005). Bayesian inference for stochastic kinetic models using a diffusion approximation. *Biometrics*, 61(3):781–788.

Golightly, A. and Wilkinson, D. J. (2011). Bayesian parameter inference for stochastic biochemical network models using particle Markov chain Monte Carlo. *Interface Focus*, 1(6):807–820.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, New York, second edition.

Higdon, D., Reese, C. S., Moulton, J. D., Vrught, J. A., and Fox, C. (2011). Posterior exploration for computationally intensive forward models. In Brooks, S., Gelman, A., Jones, G. L., and Meng, X.-L., editors, *Handbook of Markov Chain Monte Carlo*, chapter 16, pages 401–418. Chapman & Hall/CRC, Boca Raton, FL.

Joseph, V. R. (2012). Bayesian computation using Design of experiments-based Interpolation technique. *Technometrics*, 54:209–225.

Joseph, V. R. (2013). A note on nonnegative DoIt approximation. *Technometrics*, 55:103–107.

Kennedy, M. C. and O'Hagan, A. (2001). Bayesian calibration of computer models (with discussion). *JRSSB*, 63:425–464.

Kostov, S. and Whiteley, N. (2016). An algorithm for approximating the second moment of the normalizing constant estimate from a particle filter. `http://arxiv.org/abs/1602.02279`.

Meeds, E. and Welling, M. (2014). GPS-ABC: Gaussian process surrogate approximate Bayesian computation. In *Thirtieth Conference on Uncertainty in Artificial Intelligence (UAI)*.

Overstall, A. M. and Woods, D. C. (2013). A strategy for Bayesian inference for computationally expensive models with application to the estimation of stem cell properties. *Biometrics*, 69:458–468.

Payne, R. D. and Mallick, B. K. (2014). Bayesian big data classification: a review with complements. `http://arxiv.org/abs/1411.5653`.

Quiroz, M. (2015). Speeding up MCMC by delayed acceptance and data subsampling. `http://arxiv.org/abs/1507.06110`.

Rasmussen, C. E. (2003). Gaussian processes to speed up hybrid Monte Carlo for expensive Bayesian integrals. In Bernardo, J. M., Bayarri, M. J., Berger, J. O., Dawid, A. P., Heckerman, D., Smith, A. F. M., and West, M., editors, *Bayesian Statistics 7*, pages 651–659, Oxford. Oxford University Press.

Roberts, G. O. and Rosenthal, J. S. (2007). Coupling and ergodicity of adaptive Markov chain Monte Carlo algorithms. *J. Appl. Probab.*, 44(2):458–475.

Roberts, G. O. and Rosenthal, J. S. (2009). Examples of adaptive MCMC. *J. Comput. Graph. Statist.*, 18(2):349–367.

Sacks, J., Welch, W. J., Mitchell, T. J., and Wynn, H. P. (1989). Design and analysis of computer experiments. *Statistical Science*, 4:409–435.

Shen, Y., Ng, A. Y., and Seeger, M. (2006). Fast Gaussian process regression using kd-trees. In *In Advances in Neural Information Processing Systems 18*. MIT Press.

Sherlock, C., Fearnhead, P., and Roberts, G. O. (2010). The random walk Metropolis: linking theory and practice through a case study. *Statist. Sci.*, 25(2):172–190.

Sherlock, C., Thiery, A., and Golightly, A. (2015). Efficiency of delayed acceptance random walk Metropolis algorithms. *In prepartion*. `http://arxiv.org/abs/1506.08155`.

Storer, J. (2002). *An introduction to data structures and algorithms*. Springer-Verlag, New York.

van Kampen, N. G. (2001). *Stochastic Processes in Physics and Chemistry*. North-Holland.

Vihola, M. (2012). Robust adaptive Metropolis algorithm with coerced acceptance rate. *Stat. Comput.*, 22(5):997–1008.

Wilkinson, D. J. (2012). *Stochastic Modelling for Systems Biology*. Chapman and Hall/CRC Press, London, 2nd edition.

Wilkinson, R. D. (2014). Accelerating ABC methods using Gaussian processes. In *JMLR Workshop and Conference Proceedings Volume 33: Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*.