Notes on a Local Approximation MCMC Scheme

Ben Tran

February 19, 2021

1 Introduction

Here are some thoughts and results from MCMC experiments, using the local approximation scheme described in this paper: https://arxiv.org/pdf/1402.1694.pdf. The goal was to use this to sample from high-dimensional geology models with computationally expensive log-posteriors. Many of the ideas in this report are concerned with modifying the ideas in the paper to speed up the scheme for high-dimensional problems.

2 Overview of the Approximation Scheme

The first experiments used the scheme exactly as described in the paper, so a brief description of the algorithm will be put here. Essentially, a set S of points over the space of θ, and their unnormalised log-posterior output, will be built up over time as MCMC samples are collected. To compute the value of the unnormalised log-posterior of the model, a regression is used, using a select number of points from S close to the θ where we are sampling from. The paper describes using either linear, quadratic or Gaussian process regression. In this way, we ideally have to compute the log-posterior of less points than we would if we were calling the model log-posterior every time we needed it. The paper describes the criteria for triggering adding a point to S, and also where to add points in S.

3 Preliminary Tests

The main difference between the geology problems we face and the numerical experiments in the paper is the number of dimensions; the smallest of our geology problems have 15 dimensions, whereas the paper did not venture over 6 dimensions. This caused many features of the method in the paper to slow down considerably, so some modifications were made to speed up the algorithm.

3.1 Discarding the Optimisation Problem for Refinement

In the paper, the algorithm prescribes that once refinement is decidedly necessary around a point θ, a new point should be placed at the first local optimizer of the problem in Figure 1.

$$\theta^* = \arg\max_{\theta'} \min_{\theta_i \in \mathcal{S}} \|\theta' - \theta_i\|_2,$$
$$\text{subject to } \|\theta' - \theta\|_2 \leq R,$$

Figure 1: Optimisation problem for deciding on a new point to add to S

In the optimisation problem shown, θ_i are the sample points in S already, and R is the distance between θ and the furthest point that would be used in a local regression around θ. This method was extremely slow for high-dimensional problems. An alternative is generating θ^* uniformly in the ball of radius R around θ. This speeds the algorithm up considerably in higher dimensions. However, another problem arises in higher dimensions: there is much more space nearer the boundary of the ball of radius R, and so most points generated are far away from θ. A good solution in practise is to generate the normalised 'angle' vector of length 1 on a d-dimensional sphere, and generate the distance between θ^* and θ using a uniform distribution. (Alternatively, one can choose to skew the distance more towards theta using another distribution). This generates sample points closer to θ, which can more effectively refine the linear regression around θ.

3.2 Deciding When Refining the Set of Points S is Necessary

The paper uses a leave-one-out cross-validation strategy to determine whether the regression approximation to the model is accurate enough. In the simplest regression case, a linear regression model, a minimum of N_{def} = d + 1 points are required to construct a regression around some point θ, where d is the dimension of the space of θ. The paper recommends that, for a linear regression, $N = N_{def}\sqrt{d}$ is a reasonable number of points to use for a local regression, which works well in practise. In 15 dimensions, this would make $N = 64$ (taking $\sqrt{15} = 4$). So, the leave-one-out cross-validation strategy described in the paper would have to perform 64 linear regressions before computing the error metrics used to decide whether a refinement is needed. Even in 15 dimensions, this is unfeasibly slow, and the size of this would clearly grow as $O(n^(3/2))$ in the linear regression case. So, we propose two other ways of triggering refinement.

K-fold cross validation can be used instead of leave-one-out cross validation. The advantage of this method is that one can compute the same error metrics based on the MH acceptance rate that are used in the paper. Also, this scales well with dimension, as keeping k constant will not significantly slow down the validation step; only k regressions are being performed.

R^2 validation is another method which can be used to trigger refinement. This works very simply; if the R^2 of the regression at a point θ is below a certain threshold, refinement is triggered. This has the advantage of being significantly faster than k-fold cross validation and leave-one-out cross validation. A disadvantage of this method is that it does not use the same error metric, based on the MH acceptance rate, as the paper, so it is not as comparable to the results generated in the paper.

4 Which Type of Regression to Use?

As the problems we are dealing with are high-dimensional, using a quadratic regression scheme is infeasible due to the vast number of interaction terms that would be required. Also, Gaussian process regression was tested on these high-dimensional models and proved to take far too long compared to linear regression to be worth using. So, for the rest of this report, we use a standard linear regression for the local approximations.

5 Finding the K Nearest Points for Regression

This is not a modification to the algorithm, but a description of an efficient way to store the sample set S. To approximate the log-posterior at a point θ using a regression, we need to find the N closest points to θ in the point set S. To perform this as efficiently as possible, it is advisable to store the points in S in a tree structure such as a KD-Tree, which allows for fast k-nearest-neighbours lookups. Unfortunately, these trees are not designed to be added to, so whenever a point is added to S it is necessary to reconstruct the tree. Although the construction of the tree takes longer than a lookup, it is still much faster to use this method in practise than sorting every point in S by their distance from θ.

6 A Useful Metric When Tuning Parameters

The reason for using a local approximation method like this is to reduce the number of function calls needed to sample from an expensive posterior distribution. Essentially, we are trying to maximise the number of MCMC samples we can get per point in S. So, a good metric to use when tuning parameters for a specific model is: $\frac{\text{number of samples}}{|S|}$. It's much easier to get this metric higher for simpler distributions, and for lower dimensional spaces.

7 The Disadvantages of Hamiltonian Monte Carlo

It would seem a good idea to use HMC in this context, as an approximation to the gradient vector of the log-posterior can be easily computed using a local linear regression. However, as the leap frog integrator needs many gradient steps on the way, there must be sufficient sample points generated on the way to compute these gradients. As a result, it is very expensive computationally to generate one sample, more so if there are more steps in the leap frog integrator. However, one possible scenario where HMC would work well with this algorithm is if the gradients are easily computable, but the actual log-posterior is comparatively expensive to compute.

8 A Faster Method of Refining Points

The following is a faster method of refining points, not used in the paper. In high dimensions, if the model needs refinement around a certain θ, it may take many refinements to S before we do not need to refine S anymore, and can perform a regression around θ. Depending on the situation, the number of points used can blow up fairly quickly and inefficiently. If it has been

decided that a refinement is needed at a point θ (using any of the methods described above), we can instead choose to add θ and its log-posterior directly to the set S, and use the exact value of the log-posterior at θ for this sample step. This means that in the worst case, the algorithm will generate a point in S for every sample point it generates in the MCMC chain, which is simply the same performance as a standard MCMC (ignoring regression checking times). Usually, this method performs much better than this, even under very stringent cross-validation parameters. So the metric described above, $\frac{\text{number of samples}}{|S|}$, will always be at least 1, implying that some performance has been gained for some log-posterior which is expensive enough to compute.

9 Experiments on the Geological Model

We will now use some settings of the local approximation scheme described above and compare it to a standard Metropolis-Hastings MCMC sample. This model isn't quite expensive enough to see actual performance gains in terms of speed, however we will use the metric of $\frac{\text{number of samples}}{|S|}$ to determine how much better the local approximation scheme would perform with more expensive models.

The baseline standard MH algorithm was run for $100,000$ samples, using a gaussian walk proposal distribution. The step sizes across each dimension were pre-tuned to roughly match the scale of that dimension. A few slices of the log-posterior KDE are shown in figure 2. This model took 780 seconds to run, with an acceptance rate of 0.15.

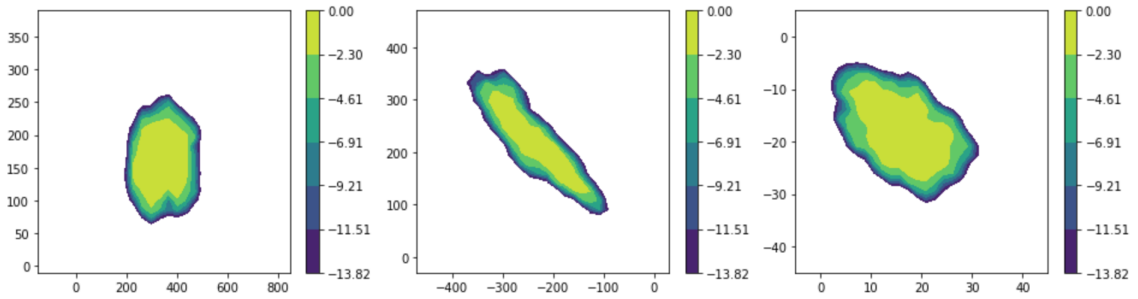

Figure 2: Standard MH MCMC (No Emulator)

The emulator scheme was then run for $30,000$ samples. The following options/parameters were used:

- 3-fold cross-validation
- $\gamma = 2$, $\beta = 0.1$
- Exact refinement, as described in section 8

The results are shown in figure 3. The result was $\frac{\text{number of samples}}{|S|} = 3.08$, taking 4243 seconds, with an acceptance rate of 0.23.

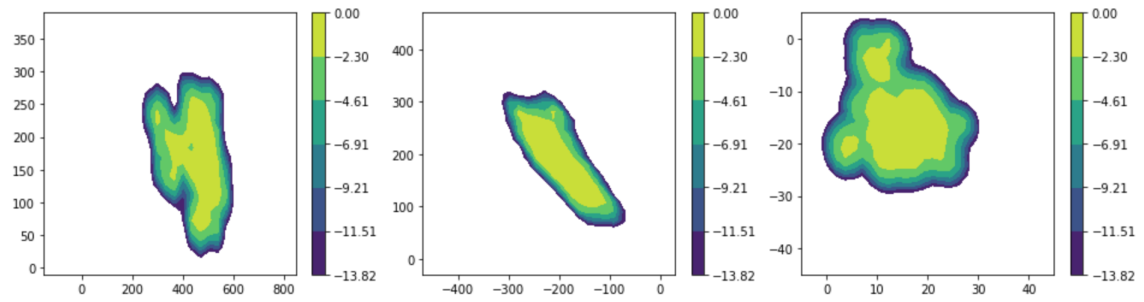

Figure 3: K-fold validated model

We run the emulator again with the same parameters except using R^2 validation, with an R^2 threshold of 0.999. The results are shown figure 4. The result was $\frac{\text{number of samples}}{|S|} = 3.98$, taking 823 seconds, with an acceptance rate of 0.148.

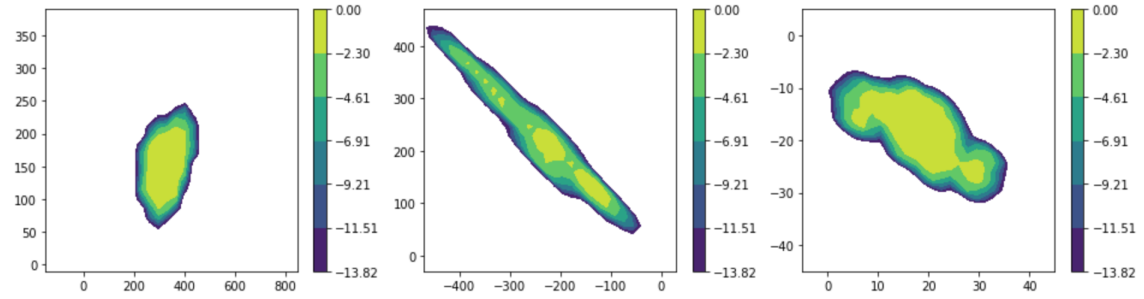

Figure 4: R^2 validated model

One can see with the previous two models in figures 3 and 4, especially figure 3, that the MCMC chain seems to be 'leaking' out of the distribution, implying that the distribution is converging too slowly. We can fix this problem by setting β higher to $\beta = 0.15$, so that random refinement of points happens more often. We do this with the R^2 validation model. The results are shown in figure 5. The result was $\frac{\text{number of samples}}{|S|} = 3.28$, taking 1026 seconds, with an acceptance rate of 0.15.

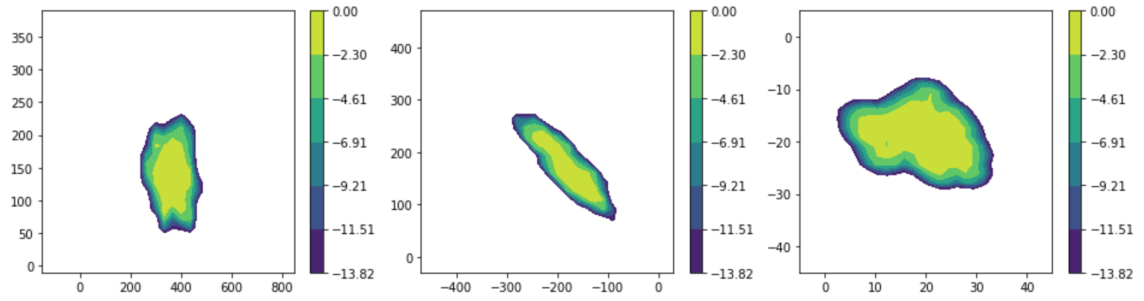

Figure 5: R^2 validated model with $\beta = 0.15$

Based on these experiments, we can see that it was significantly faster to use R^2 validation to determine whether to refine points.

10 Discussion

Although there are some promising results, especially with the final model shown (in figure 5), the emulator scheme still took much longer to run than the standard MH MCMC. In high dimensions (above 4 or 5), it would only really be worth trying this scheme with a really expensive log-posterior distribution. We were reaching ratios of $\frac{\text{number of samples}}{|S|} > 3$, which implies there is some performance to be gained from this method on more expensive log-posteriors.