6.882: Bayesian Modeling and Inference **Lecture Date: March 6, 2018**

Slice Sampling

Lecturer: Kelvin Lu, Edward Park, Brandon Zeng Scribe: Chuanquan Shu, Will Stephenson, Kevin Yang

1 Introduction and Motivation

In the last couple of classes we've been concerned with random sampling, with the idea that if we can draw a large number of independent samples from a given distribution, then we can reasonably approximate any function of that distribution. Unfortunately, our previous methods all fall a bit short in different ways.

- Importance and rejection sampling do poorly in high dimensions, because their acceptance probabilities decay exponentially with the number of dimensions, causing an exponential slowdown.
- Metropolis-Hastings needs a proposal distribution, which requires tuning to avoid undesirable random-walk behavior. To cross the distribution, it needs a number of steps which grows quadratically with the ratio between the distribution's length scale and the random walk's step size, but if the step size is too large then we pay a big penalty in the acceptance rate. The step size must be fixed before the algorithm begins.
- Gibbs sampling may require samples from nonstandard distributions when the conditionals are non-standard. Moreover, in some higher-dimensional cases, it can also exhibit undesirable random-walk behavior. As always, our example is the 2D Gaussian where the two dimensions are highly correlated.

Slice sampling aims to remedy these problems to some extent, and we will see that it succeeds in the one dimensional case. We try to answer the following questions:

- Can we adaptively modify the scale of changes, without needing too much tuning? This is especially important when the length scale differs drastically between different dimensions of the distribution.
- Relatedly, can we suppress random walks?
- Can we adapt to dependencies between variables, such as in the 2D Gaussian where the two dimensions are highly correlated?

For spoilers, see the summary at the end.

2 Main Idea

We now introduce the main idea of slice sampling from Neal (2003). We start with the univariate case to simplify our exposition. Consider a random variable X with a distribution function proportional to $f : \mathbb{R} \to \mathbb{R}$. The basic algorithm behind slice sampling is the following (in slightly simplified form):

1. Choose a starting point x_0.
2. Sample an auxiliary variable $y \sim \text{Unif}[0, f(x_0)]$, and define the *slice* $S = \{x \in \mathbb{R} \ : \ y < f(x)\}$.
3. Sample uniformly from S to get a new point x_1, and repeat.

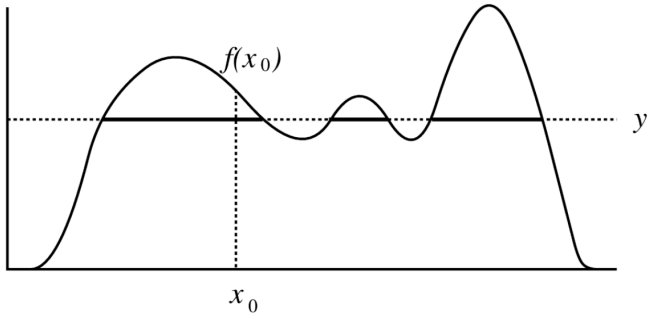

Figure 1: A possibly unnormalized distribution f over $\mathbb{R}$. Starting at the current point x_0, a point $y \sim \text{Unif}[0, f(x_0)]$ is sampled. Drawn in bold is the slice, S, which is the set of x values for which $f(x) < y$.

The next lemma shows that this procedure has the correct invariant distribution:

Lemma 1. *After throwing away the values for the auxiliary variable y, the simplified slice sampling algorithm above has $p(x)$ as its invariant distribution.*

Proof. We can show detailed balance. Observe that for any two points x_0 and x_1, they can transition to each other if and only if the auxilliary variable y is chosen smaller than $min(f(x_0), f(x_1))$. Given this choice of slice S, we are equally likely to end up in either x_0 or x_1.

So, it suffices to show that we're equally likely to end up with slice S starting from x_0 or x_1, that is, $x_1 \in A = \{x : x \in S \cap I, P(\text{select } I \text{ starting from } x) = P(\text{select } I \text{ starting from } x_0)\}$. But the probability of choosing slice S starting from a point x is proportional to $f(x)\frac{1}{f(x)} = 1$, where $f(x)$ is proportional to the probability of starting at x, and $\frac{1}{f(x)}$ is proportional to the probability of selecting slice S given that we start at x. Therefore, we're equally likely to end up with slice S starting from either x_0 or x_1, which completes our detailed balance proof. □

When we modify the algorithm in the next section to make it practical for implementation, we will keep the basic idea of this detailed balance proof in mind so that we can preserve this property in all our implementation choices.

Remark 2. It's also worth noting that we can view slice sampling as a Gibbs sampler that alternates between sampling x and y.

3 Implementation Details

The problem with the simplified slice sampling algorithm above is that it may not be easy to sample uniformly from S. In particular, it may not be tractable to compute the intervals that comprise S, which would require

finding all solutions to $f(x) = y$. We therefore augment our slice sampling algorithm with an additional step:

1. Choose a starting point x_0.
2. Draw a value y uniformly at random in $[0, f(x_0)]$, and define the *slice* $S = \{x \in \mathbb{R} \ : \ y < f(x)\}$.
3. **(New Step)** Find a suitable interval $I = (L, R)$ containing x_0 that approximates S.
4. Sample from I until we find a new point x_1 in the slice S, and repeat.

There are a number of factors that control how we pick the interval I: 1) computational feasibility – given the current state (x_0, y) of the sampler, it should be fast to compute I. 2) The interval should contain as much of S as possible to promote fast mixing of the sampler. 3) The interval should contain as little area not in S as possible so that we reject as few samples as possible in step 4 above. 4) The choice of I should maintain detailed balance.

Below, we discuss two possible methods for selecting the interval I. Essentially we start with some base interval around x_0, and extend it outwards until we leave the slice. This way, we can keep extending the size of the interval as long as we're incorporating more of our desired slice with each extension. Significantly, these methods enable slice sampling to adapt to distributions where the length scale varies drastically across the space.

3.1 Stepping out

The *stepping out* method begins with a fixed w (estimate for width of the slice) and fixed m (an upper bound on number of steps), and does the following:

1. Choose a starting point x_0.
2. Choose a random integer in $[0, m]$ as m_L, the max number of times we "step left." Set $m_R = m - m_L$ as the max number of times we "step right."
3. Randomly place an interval $I = (L, R)$ of length w around x_0.
4. While L is in the slice, decrease L by w to extend the slice leftward. Do this a maximum of m_L times.
5. While R is in the slice, increase R by w to extend the slice rightward. Do this a maximum of m_R times.
6. Sample uniformly from the new (L, R) until we find a point x_1 in the slice, and repeat.

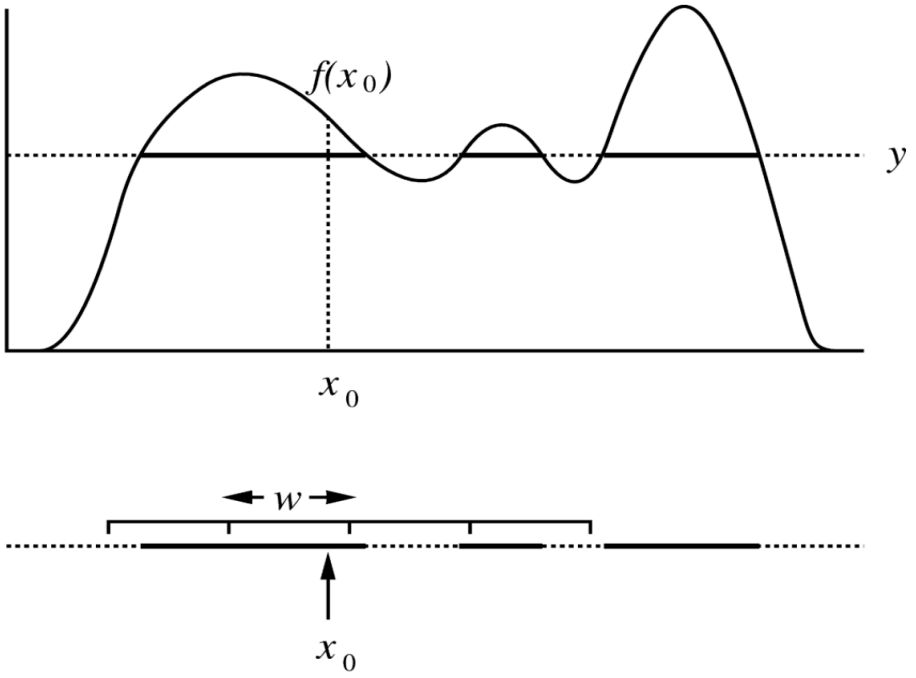

Figure 2: Stepping out: the bolded interval in the top figure corresponds to the slice S. Starting from the point x_0 in the arrow, the lower figure shows the result of stepping out once to the left and twice to the right.

The above figure illustrates the stepping out procedure and its termination when starting from x_0 using width w. Some of the details in our algorithm may seem odd, but many other reasonable-looking choices would fail to preserve detailed balance, as we'll see later.

3.2 Doubling

The *doubling* method is conceptually similar to the stepping out method: it doubles the interval in a given direction rather than just taking one step, and slightly changes how it decides which direction to extend. This expands the size of the interval significantly faster; however, this comes at the cost of destroying detailed balance. To fix this, we have to add an extra rejection step when sampling from the interval. The doubling method also begins with a fixed w (estimate for width of the slice) and a fixed p (an upper bound on number of *doubling* steps), and does the following:

1. Choose a starting point x_0.
2. Randomly place an interval $I = (L, R)$ of length w around x_0.
3. While either L or R is still in the slice, double the interval in either the L or R direction, chosen at random. That is, set $L \leftarrow L - (R - L)$ or $R \leftarrow R + (R - L)$. Repeat this process at most p times.
4. Sample uniformly from the new (L, R) until we find a point x_1 in the set $A = \{x : x \in S \cap$

$I, P(\text{select } I \text{ starting from } x) = P(\text{select } I \text{ starting from } x_0)\}$. (Note that it no longer suffices for x_1 to be in S.) Repeat.

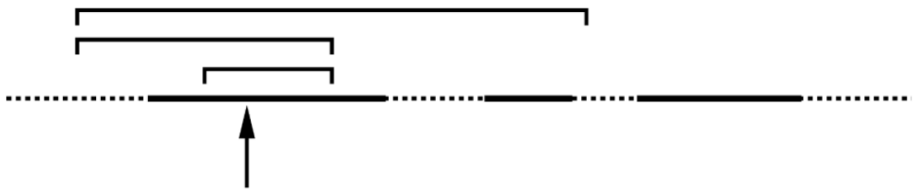

Figure 3: Doubling: starting at the point indicated by the arrow, the doubling procedure is applied, which first doubles to the left, and then to the right before terminating.

As mentioned, when we sample from $I = (L, R)$ in the last step, we can no longer just pick any point in $S \cap I$. Recall that having x_1 be in $A = \{x : x \in S \cap I, P(\text{select } I|x) = P(\text{select } I|x_0)\}$ was a key property in our previous detailed balance proof. Unlike in our first oversimplified slice sampler or the stepping out sampler, it turns out that not every point in $S \cap I$ is necessarily in A when we use the doubling method, such as in the following illustration:

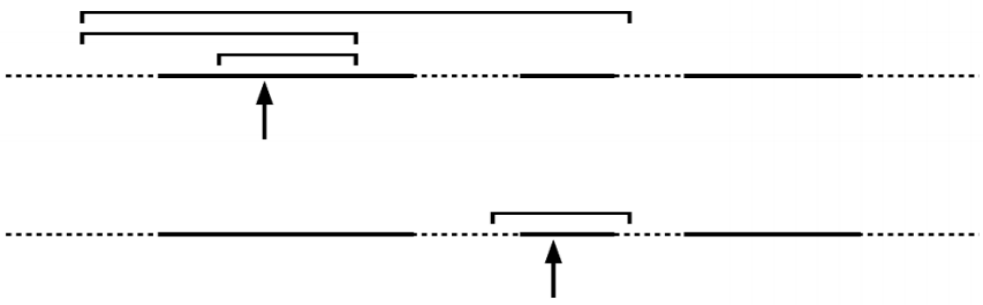

Figure 4: A Rejected Sample in Doubling. Doubling generates an interval I from the point indicated by the arrow in the top diagram. From there, we might sample the point indicated by the arrow in the bottom diagram. However, doubling cannot generate the interval I starting from the second point; as a result, the second point is not in A, and is rejected.

Remark 3. For unimodal distributions our life is a lot simpler, since for sufficiently large m and p, we'll always choose I containing the entire slice S for both the stepping out and doubling procedures. We can even retrospectively tune w since that won't affect our final interval I (more precisely, $S \cap I$), merely the time it takes to find it.

3.3 Interval Shrinkage

It will inevitably happen that $I \cap S$ is a small fraction of the total size of I (e.g. we may choose w to be way too big). Fortunately, we can edit the final sampling steps in both the stepping out and doubling methods to use the following *interval shrinkage* procedure:

1. Randomly draw x_1 from $I = (L, R)$.
2. If $x_1 \in A = \{x : x \in S \cap I, P(\text{select } I|x) = P(\text{select } I|x_0)\}$, accept x_1, and we've found our desired sample. (For stepping out, we just have $A = S \cap I$).
3. Otherwise, cut off half of our interval using x_1 as the cut point, keeping the piece that contains x_0 as the new I, and discarding the other half.
4. Repeat this process with the new I until we find a sample in A.

The great part about shrinkage is that, on expectation, the interval is cut in half for every rejected sample. Even if we selected I much too large, this should quickly generate a valid sample without too many rejections.

4 Correctness (Detailed Balance)

If we ignore the shrinkage procedure, the only modification we need to make to our proof that "simplified" slice sampling works is that we need to show, for any x_0, x_1, that $p(\text{choose } I \mid x_0) = p(\text{choose } I \mid x_1)$. This suffices, as, once I is chosen, the sampling procedure is independent of the initial point. If I is chosen by stepping out, the random alignment of the initial interval, combined with the random selection of which side of the interval to expand, gives $p(I \mid x_0) = p(I \mid x_1)$. In the case of doubling, we explicitly reject points x_1 for which $p(I \mid x_1) \neq p(I \mid x_0)$.

When sampling from I using the shrinkage procedure, the way we sample depends both on I and the current point, which complicates the above argument. The concern is that, if x_0 and x_1 are separated by a region of I that lies outside of S, then, with say x_0 as the current point, we might sample from there and then toss out the portion of I containing x_1. However, there exist a corresponding and equally-likely series of steps by which the opposite can happen with x_1 as the current point, which preserves the property $p(x_1 \mid x_0) = p(x_0 \mid x_1)$ for a given interval I.

5 Multivariate Slice Sampling

To sample a multidimensional distribution with slice sampling, we could just sample one dimension at a time. This would work, but since this basically just turns our algorithm into a Gibbs sampler, we would suffer from the same problems that Gibbs suffers from (e.g. random walk behavior in highly correlated distributions).

In the following subsections, we address three ideas to directly sample from multivariate distributions: hyperrectangles, crumbs, and reflection.

5.1 Idea: Hyperrectangles

Hyperrectangles are the most obvious generalization of slice sampling to multiple dimensions: instead of forming a 1-D interval I that approximates the slice, we instead form a multidimensional hyperrectangle H

that approximates the slice. Let the current state be $x_0 = (x_{0,1}, ..., x_{0,n})$, the new state $x_1 = (x_{1,1}, \ldots, x_{1,n})$, and $f(x)$ the unnormalized pdf from which we want to sample. Then:

1. Get a sample $y \sim \text{Unif}[0, f(x_0)]$. The slice is then $S = \{x \in \mathbb{R}^n \ : \ y < f(x)\}$.
2. Find a hyperrectangle, $H = (L_1, R_1) \times \cdots (L_n, R_n)$, around x_0, which preferably contains a big part of the slice.
3. Get a new sample x_1, uniformly from the part of the slice within this hyperrectangle.

In the 1D case, we choose our interval by an expansion scheme such as stepping out or doubling, which automatically adapt to the distribution's scale by assuring that the interval at least covers one mode. In the multivariate case this involves checking for the endpoints of a hyperrectangle being outside of the slice. As a n-dimensional hyperrectangle has 2^n vertices, this would be computationally expensive for large n. This is Neal's motiviation for just starting with a huge rectangle H and only apply shrinkage scheme. Neal proposes two ways to shrink: "all-axis" and "best-axis".

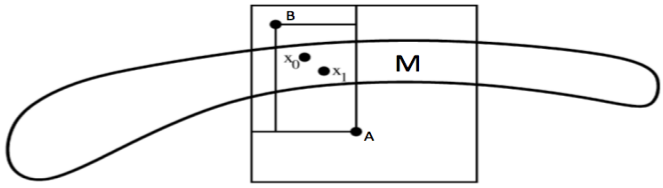

Figure 5: "all-axis" shrinkage

The figure above illustrates "all-axis" shrinkage. The heavy line outlines the slice containing the current point x_0. The largest rectangle is the initial hyperrectangle. We first get a point A but it is outside of the slice, so we apply 1D shrinkage scheme to each dimension. The result is a smaller hyperrectangle with A as one of the vertices. We then get B which is also outside and we shrink again. Finally we get point x_1 that is inside the slice and we stop.

The drawback of the "all-axis" shrinkage approach is that all dimensions are shrunk even though shrinking some of these dimensions is unnecessary. To illustrate, in the previous figure, as a result of shrinking due to point A, region M is subsequently discarded even though it is in the slice (hence a legitimate region where x_1 might come from).

This motivates "best-axis" shrinkage, as illustrated in the figure below:

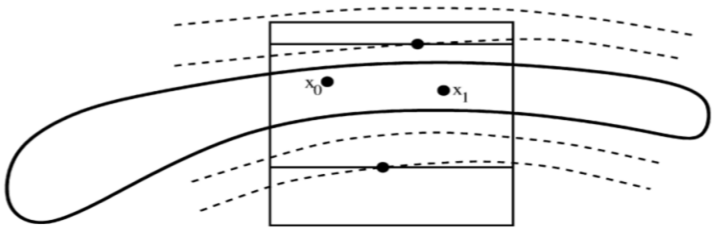

Figure 6: "best-axis" shrinkage

The idea is that we only shrink the dimension in which the probability density function varies the most rapidly. This is achieved by shrinking the dimension i which has the biggest $(R_i - L_i)|G_i|$ where G_i is the gradient of $\log f(x)$ at the last chosen point.

5.2 Idea: Crumbs Framework

The key idea behind hyperrectangles and the shrinkage procedure is that every time we draw a trial point that does not lie in the slice S, we use this information to shrink the rectangle, making it more likely that the next trial point will be in S. One can imagine a generalization of this idea, in which we use previous failures to guide us in more creative ways. Neal refers to this idea as leaving a trail of "crumbs" to guide us. Given a series of crumbs, $c_1, \ldots, c_i$ and failed trial points $x_0, x_1, \cdots x_i$ (including the current point x_0), a new crumb c_{i+1} is drawn from some distribution depending on $\{x_0, c_j, x_j\}_{j=1}^{i}$. The next trial point x_{i+1} is then drawn according to the probability that the previous crumbs would have been generated were the current state x_{i+1} instead of x_0. While this slightly complicates things, drawing x_{i+1} in this manner guarantees detailed balance.

For example, we can consider Gaussian crumbs. We can draw every crumb $c_i \sim N(x_0, \sigma^2 I)$ for some fixed σ^2. Then $x_i \sim N(\frac{1}{i}\sum_{j=1}^{i} c_j, \frac{\sigma^2}{i} I)$ by the property of gaussian pdfs multiplication. In this case, as trial progresses, the new trial points will come from a narrower and narrower pseudo-posterior distribution centered around the x_0. However, this is not much better than the hyperrectangles scheme. To improve on things, we might consider $c_i \sim N(x_0, \Sigma_i)$, where Σ_i is a covariance matrix that has been adapted to fit the shape of S based on the previous crumbs and trial points. This would naturally allow the framework to adapt to dependencies between the variables.

It is important to note that the crumbs need not be numbers. For example, the hyperrectangle idea can be viewed as a special case of the crumbs idea. The randomly placed initial hyperrectangle is the first crumb. The first trial point is chosen from those points that could produce this initial hyperrectangle, which are all points making up the hyperrectangle. The second crumb is a shrunken hyperrectangle based on the shrinkage rule applying to the current point x_0, the first crumb c_1 and the trial point x_1; it can be viewed as drawing from a degenerated distribution, concentrated on that hyperrectangle. The second trial point is then chosen from the points contained in the shrunken hyperrectangle, etc.

5.3 Idea: Reflection

With reflection, multi-variate slice sampling can suppress the random walk behavior more or less exhibited in the hyperrectangle and crumbs methods. The basic idea is that we keep going straight in a direction until we hit the boundary, at which point we reflect to a new direction and keep exploring until finally exhausted. In particular,

1. Get a sample $y \sim \text{Unif}[0, f(x_0)]$. The slice is then $S = \{x : y < f(x)\}$.
2. Pick a n-dimensional momentum vector, p, which serve to indicate the current direction and speed. This p is drawn independently of x and may come from, say, $N(0, I)$.
3. For a predetermined number of steps, update x to x' in the direction of p: $x' = x + wp$, where w is a scale parameter that determines the average step size. The final resting x' is the new point x_1.

During the traversal describe in the third point above, if a step results in x' being outside of the slice, we need to bring it back into the slice by reflecting from the boundary. In other words, when we hit the boundary,

we change the direction vector p to $p' = p - 2h\frac{p \cdot h}{|h|^2}$ (by reflection formula) where h is the gradient of $f(x)$. The following figure illustrates this idea.

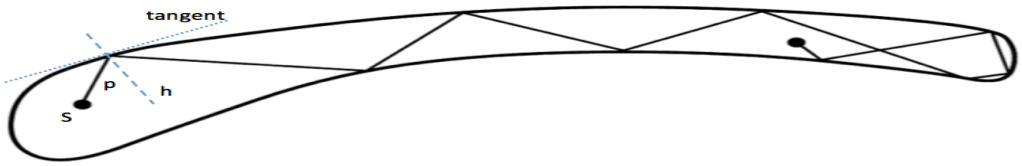

Figure 7: Ideal reflection: the trajectory reflects exactly at the boundary of the slice S; however, this is almost always impossible.

The heavy line is again the slice. The start point is S in the direction of vector p, reflecting by vector h. We can see that the motion can be fairly consistent in direction (until the end of the slice is reached) rather than in a random walk fashion. The drawback with ideal reflection is that we need to reflect *exactly* when we hit the boundary of S. Given that we are following p for a series of discrete steps, this will almost never happen. So, the following two approximate reflection procedures are suggested: inside-point reflection and outside-point reflection.

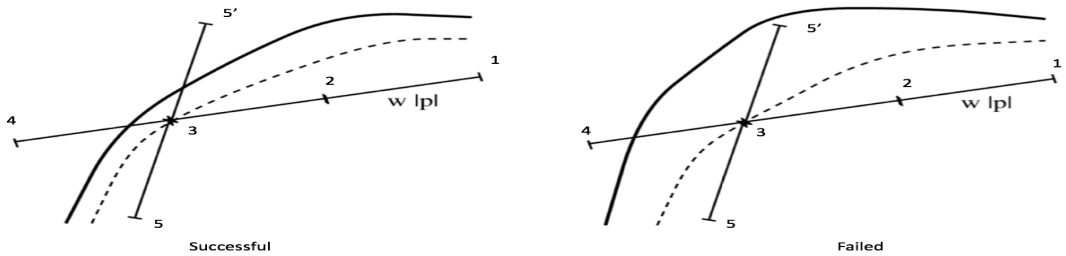

Figure 8: Inside-point reflection

Inside-Point Reflection. The above figure illustrates the idea: on both the left and right, we start at point 1, step through points 2 and 3, and finally end up at point 4, which is outside the slice. We then backtrack to the last point inside S (point 3), and reflect here. The right side of the above figure illustrates an issue with this approach: due to asymmetries in the distribution $p(x)$, it may not be that the reflected trajectory would also reflect at point 3. In order to maintain detailed balance, we need to check for such situations and reject appropriately. Alternatively, we could do outside-point reflection, as illustrated below.

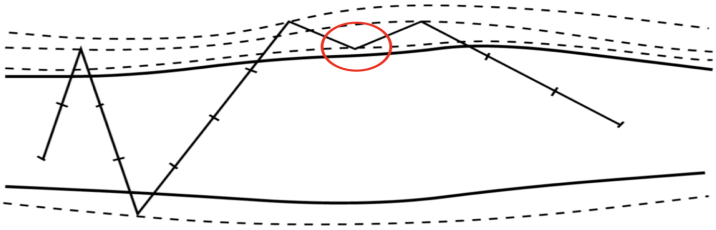

Figure 9: Inside-point Reflection

Outside-Point Reflection. The idea of outside-point reflection is to reflect at every point outside the slice. At the circled point above, this leads to us reflecting back away from the slice. While this more easily satisfies detailed balance, we still may have to reject samples that do not end up inside the slice.

6 Empirical Results

To illustrate the benefits of slice sampling over Metropolis-Hastings (MH), Neal compares the two on a "10-D funnel." That is, the true posterior over variables $v, x_1, \ldots, x_9$ is:

$$p(v, x_1, \ldots, x_9) = N(v \mid 0, 3) \prod_{i=1}^{9} N(x_i \mid 0, e^v).$$

This resembles a funnel in the sense that, for small values of v, the probability mass of $p(x_1, \ldots, x_9 \mid v)$ is concentrated on a very small ball in $\mathbb{R}^9$, whereas for a larger v, its mass is spread over a much larger ball. The main difficulty of this sort of distribution is that the radii of these balls scales with roughly $e^{\frac{1}{2}v}$, making it essentially impossible to explore the entire space without proposals that take scale into account.

Neal's experiment confirms this intuition: using a spherical Gaussian for the MH proposal distribution, he computes twenty million MH iterations and thins this down to two thousand samples. He shows that the marginal distribution of v has not been correctly sampled from; for example, none of the two thousand samples lie within the 5% left-hand tail of the true distribution. Given actual i.i.d. draws from $p(v)$, the probability of this event is roughly 1×10^{-44}. What is worse is that a practitioner would have no indication that such a huge failure has ocurred.

In contrast, slice sampling seems to correctly estimate the moments of $p(v)$. What's left unsaid about these experiments is how the computation time varies, how slice sampling compares to other MCMC methods, or how the relative performance of the methods changes for different posteriors.

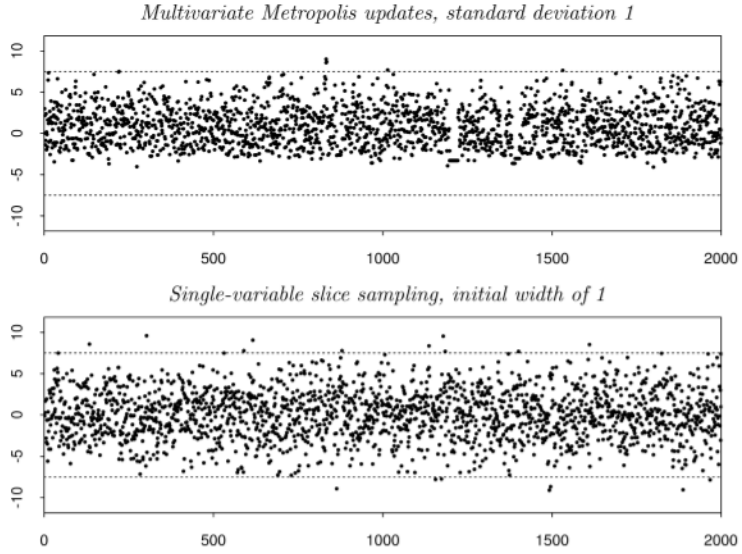

Figure 10: Slice sampling vs MH with a spherical proposal distribution. The dashed lines correspond to the 90% quantile of the true distribution for v. We see that MH completely fails to explore the lower range of this area.

7 Summary

Slice sampling seems to be an effective method for sampling from univariate distributions. In particular, when using doubling, it is able to quickly adapt to the scale of the distribution without too much impact from the parameter w. While Metropolis-Hastings (MH) or dynamical (e.g. Hamiltonian Monte Carlo) methods have similar potential, they seem to be much more sensitive to their input parameters. For a demonstration in the univariate case, see this github repository.

It is not entirely clear that slice sampling is an improvement in the multivariate domain. The author presents very compelling evidence that slice sampling outperforms MH; however, this is based on single-variable updates, and it is not clear how it compares to something like Gibbs sampling in this context. Methods for extending slice sampling beyond single variable updates are proposed, but it is not clear that these are better than other methods. For example, "reflective" proposals are essentially a more restrictive version of those used by Hamiltonian Monte Carlo, which, as mentioned, suffers from serious tuning issues.

References

Neal, R. M. (2003). Slice sampling. *Ann. Statist.*, 31(3):705–767.