Computational Experiments in Markov Chain Monte Carlo

by

Alexander D. Kaiser

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Master of Science

Department of Mathematics

New York University

September, 2013

Professor Jonathan Goodman

Acknowledgements

First, thank you to my advisor, Prof. Jonathan Goodman, whose guidance and wisdom were essential to the entire project. He has been a great mentor and friend.

Thank you to Prof. Andreas Klöckner for his consistent help and expertise with GPU computing, suggestions on drafts and many good conversations. Also thank you to Prof. Timothy Warburton for the gracious use of computing resources. Thank you to Prof. Esteban Tabak and Tamar Arnon, for believing in me early. Thank you also to Profs. Olof Widlund, David Bailey and Jonathan Borwein.

I'd like to thank my parents, sister, the rest of my family and my friends for their ongoing support. And finally, Michelle, still working on Michellebulon.

Contents

List of Figures

List of Tables

Introduction

In this thesis, I investigate computational questions in Markov chain Monte Carlo (MCMC). I am investigating one new MCMC method called the stretch move ensemble sampler [3]. I have looked at the performance of this algorithm, in terms of acceptance rates, autocorrelation time and compute performance. The thesis describes a parallel implementation of the algorithm for graphics processing units (GPUs). I investigate applications of the sampling, including Bayesian inference problems.

I use an MCMC method called the stretch move ensemble sampler [3]. The algorithm is affine invariant, so performs well on skewed distributions with little tuning. It generally has low autocorrelation time. It is computationally efficient and parallelizes completely with relatively minor communication.

One major component of the project is a parallel implementation of the stretch move for GPU hardware. It is written in an extension to C called OpenCL. This code shows that a highly parallel implementation of this algorithm is effective. I investigate performance optimizations and memory management strategies. I discuss how the compute performance scales with additional parallelism and show speedup on real applications.

I have taken care to make the code easy to use, with a simple, clear code interface. The user only needs to specify a probability density function (PDF). He/she does this by providing a C program that evaluates the PDF. The user does not need to write any OpenCL or understand GPU programming. Many applications users do not know about parallel computing — they just want their sampler to run fast. This code allows them to take advantage of modern computing hardware with minimal time invested in learning computing technology only tangentially

related to their science objective.

The second major component is applications. First, I apply the sampler to densities with inequality constraints that all components are nonnegative on the domain. The main example is a Gaussian with restricted domain, or a "hard wall" which . This is a toy model for entropic repulsion, which "refers to the event that the random field moves away from the hard wall in order to gain space for local fluctuations [1]." This paper contains more details on the statistical mechanics behind this behavior. In my experiment, the particles at each edge are attached to the boundary. The entropic repulsion causes particles in the center to be more likely to be farther from the boundary. Also, many inference problems involved distributions with restrictions. For example, a prior may restrict parameters to a specified range. This can have similar effects in the posterior distributions as in the physically motivated case.

I next look at a problem of exoplanet fitting. Consider a star other than the sun. Planets orbiting the star, or *exoplanets*, cannot be seen directly, but their presence can be inferred from their influence on the star. From the classical work of Kepler and Newton, the dynamics of planets are well understood. From five parameters, one can calculate the planet's influence on the velocity of the star. The radial velocity of the star, that is the velocity relative to Earth, can be measured using spectroscopy. From measurements of the radial velocity, one can estimate the orbit parameters for the star's planets. This technique is one of two major successful methods for finding planets outside our solar system. More details are in section 5 and [5]. I discuss the benefits of GPU parallelism on this problem.

The final class of problem I investigate is parameter estimation for stochastic differential equations (SDEs). That is, I consider situations where the dynamics

themselves have noise. Additionally, the dimension may be large and contain many unwanted, or nuisance, parameters. Nuisance parameters are, alas, necessary. To estimate the parameters of interest, one must also estimate the nuisance parameters. Suppose one is interested in coefficients governing an SDE. One may need to sample a distribution that involves an entire path of a solution. For these reasons, this is a challenging situation for inference.

1 MCMC and Applications

Suppose one has a continuous, multivariate random variable X with probability density function (PDF) proportional to a function f, where

$$f : \mathbb{R}^n \to [0, \infty)$$

and

$$\int_{\mathbb{R}^n} f(x)\, dx < \infty.$$

The integral of f need not be known, but must be finite. Additionally, it is known how to evaluate f, perhaps in closed form or perhaps only through some complicated numerical algorithm. For the duration of this thesis, I only consider dimensions N greater than or equal to two. The central question is this: Given f, how can I produce samples that behave approximately according to the distribution of X? What algorithm can be used to produce these samples?

The algorithms I investigate are called Markov chain Monte Carlo, or MCMC, methods. Here I summarize some relevant definitions and concepts. I discuss *Metropolis*, the oldest and most used MCMC sampler, autocorrelation time and

the main applications of MCMC.

1.1 Random Walks, Markov Chains and Detailed Balance

(These descriptions are loosely based on [6].)

An MCMC method generates samples by moving in a *random walk*, that is some sequence $X_1 \dots X_t$. The subsequent samples are determined according to stochastic dynamics, which depends on the individual algorithm.

A sequence of samples $X_1 \dots X_k$ has the *Markov property* if

$$P(X_{t+1} = x | X_t \dots X_1) = P(X_{t+1} = x | X_t).$$

The conditional distribution of X_{t+1} given all past elements is independent of all but the previous state. A random walk with the Markov property is called a *Markov chain*.

A Markov chain is *irreducible* if any state in the chain can reach any other in a finite number of steps. Informally, there are no "islands" of probability such that the chain cannot get to the other area. A state in a Markov chain has integer period i if it is only possible to return to the state in multiples of i steps. A Markov chain is called *aperiodic* if all states have period one. A state is recurrent if, starting from the state, the probability of returning to a state eventually is one. A state is positive recurrent if the expectation of the time to return is finite, and a Markov chain is positive recurrent if all states have this property. An aperiodic, positive recurrent Markov chain is called *ergodic*. See [6].

Another important property for proving convergence is called *detailed balance*. Suppose the random walk is determined by stochastic dynamics $K(Y|X)$, where

K is a conditional PDF of Y given the current state X. The detailed balance condition is specified

$$K(X|Y)f(Y) = K(Y|X)f(X).$$

For a continuous distribution, the quantities involved are not probabilities, but probability densities at a point.

To prove convergence, the following facts and theorems are used. The Perron-Frobenius theorem says that there exists a unique invariant distribution for the Markov chain. If an ergodic MCMC preserves the invariant distribution, then the iteration converges to the unique invariant distribution [6]. If the initial distribution creates an ergodic chain, this always occurs. To prove the invariant distribution is preserved, detailed balance is usually used. Other proofs are valid, but detailed balance is usually the most direct.

Convergence of an MCMC method is asymptotic. This means that the probability distribution of X_t converges to f as $t \to \infty$, but X_t never has the exact distribution for finite t. Thus, it is necessary to run a *burn-in*. The first few iterations are thrown out since the distribution has not yet converged to the invariant distribution. For some distributions, this may need to be tens of thousands or more. There is no general theory for length of burn-in required.

1.2 Metropolis

The original, standard MCMC algorithm is called the *Metropolis algorithm* [7]. A single step of the algorithm is shown in algorithm 1.

Algorithm 1 Metropolis Step

1: Let X_t be the sample at time t.
2: Propose a move Y by sampling from a *proposal distribution* $T(Y|X_t)$.
3: Evaluate a likelihood ratio for the move

$$q(Y|X_t) = \frac{T(X_t|Y)f(Y)}{T(Y|X_t)f(X_t)},$$

or if T is symmetric it cancels

$$q(Y|X_t) = \frac{f(Y)}{f(X_t)}.$$

4: If $q(Y|X_t) > 1$, accept. Otherwise, accept with probability q.
5: If accept $X_{t+1} = Y$, else $X_{t+1} = X_t$.

Informally, the algorithm works by taking an arbitrary move near the current point. If the PDF has greater value at that point, then the walk always moves there. If the PDF has a smaller value, then the move is accepted with probability dependent on how much less likely.

The algorithm generates a sequence that is ergodic, so long as the proposal distribution is designed correctly. The proposal distribution T must be made such that the chain is irreducible. If the support of the PDF is connected, (considering only continuous sampling problems) nearly any T which moves locally from the current state satisfies this. But if the PDF is supported only on two or more disjoint sets, T must be designed accordingly. The algorithm includes a rejection step, and thus the dynamics it specifies are aperiodic because the distribution can return to the current state through a rejection. Then with a properly designed T the sequence generated is ergodic. Two common choices for T are uniform or isotropic Gaussian in a neighborhood of the current state.

The dynamics of the algorithm satisfy detailed balance. Let X and Y be arbitrary states. The density to transition from X to Y is given $a(Y|X)\ T(Y|X)$.

The detailed balance condition for this algorithm is

$$a(Y|X)\ T(Y|X)\ f(X) = a(X|Y)\ T(X|Y)\ f(Y).$$

The left hand side is the density to transition from X to Y multiplied by the density of starting at X. The right hand side is the reverse. Suppose first that $q(X|Y) > 1$. Then $q(Y|X) < 1$ and $a(Y|X) = q(Y|X)$. Then

$$\begin{aligned} a(Y|X)\ T(Y|X)\ f(X) &= \frac{T(X|Y)f(Y)}{T(Y|X)f(X)}\ T(Y|X)\ f(X) \\ &= T(X|Y)f(Y) \\ &= a(X|Y)\ T(X|Y)\ f(Y), \end{aligned}$$

and similarly for the $q(X|Y) < 1$ case. Thus, detailed balance is satisfied.

The dynamics preserve the invariant distribution. Suppose that $X_t \sim f$, the invariant distribution. There are two ways to arrive at a new state Y. One is starting at another state X and moving to Y. This has density

$$a(Y|X)T(Y|X)f(X).$$

The second is to start at Y, then stay there through a rejection. This has probability

$$\int (1 - a(X|Y))T(X|Y)f(Y)\ dX.$$

To find the PDF of the new state Y, we must integrate the density to move to Y

for all X and add the probability of staying at Y through rejection. The PDF is

$$\begin{aligned}\phi(Y) &= \int a(Y|X)T(Y|X)f(X)\,dX + \int (1 - a(X|Y))T(X|Y)f(Y)\,dX \\ &= \int \Big[a(Y|X)T(Y|X)f(X) - a(X|Y))T(X|Y)f(Y)\Big]\,dX \\ &\quad + f(Y)\int T(X|Y)\,dX\end{aligned}$$

By detailed balance the first integrand is zero. The expression reduces to

$$= f(Y)\int T(X|Y)\,dX$$

Since the transition density T is a PDF, it integrates to one.

$$= f(Y).$$

So the iteration preserves the invariant distribution.

Since the iteration is ergodic and preserves the invariant distribution, the Mertropolis algorithm converges.

One advantage Metropolis is that it does not require knowledge of normalization constants. Because of the form of the ratio q, the normalization constant for the PDF f cancels.

1.3 Autocorrelation time and difficulties with Metropolis

MCMC methods, however, have limitations. Ideally, the samples generated would be independent, but since they are generated in a chain, they are correlated. This can be measured by the *autocorrelation time* or sometimes *integrated autocorrela-*

tion time, denoted τ, of the sequence [3]. This is defined as

$$\tau = \sum_{t=-\infty}^{\infty} \frac{C(t)}{C(0)}$$

where $C(t)$ is the lag t *autocovariance*. This is defined as

$$C(t) = \lim_{s \to \infty} \mathrm{Cov}(V(X_{t+s}), V(X_s))$$

where $V(X_t)$ is an observable. The quantity $C(0)$ is the steady state variance of the observable. The number of effective independent samples is given by M/τ, where M is the total number of MCMC steps. Metropolis tends to have high τ unless the transition density T is designed carefully. This is a major disadvantage.

Another disadvantage is that the algorithm is inherently serial. If evaluating the PDF is time consuming, it may be possible to parallelize inside that routine, but that is about the limit.

The third major disadvantage is that the algorithm may work poorly for skewed, or *anisotropic*, distributions, depending on the proposal distribution. Consider a two dimensional Gaussian with PDF given

$$f(\vec{x}) \propto \exp\left(-\frac{(x_1 - x_2)^2}{2\epsilon} - \frac{(x_1 + x_2)^2}{2}\right),$$

taken from [3] and displayed in figure 1. This would be easily sampled by direct methods and is only two dimensional. However, Metropolis has difficulty if the standard uniform or isotropic Gaussian proposal distribution is used. To see this, note that if moving in the (1,1) direction the steps would need to be of order one to sample the distribution in a reasonable number of moves. Steps in the (-1,1)

direction would need to be much smaller, because a large step moves to a region with low probability and is likely to be rejected. To get good convergence and low autocorrelation time, the proposal T needs to approximate these contours. The contours may not be clear. Tuning is costly in human time. This problem is worse in high dimensions, where the contours can be difficult to find or match with a T that can be sampled directly.

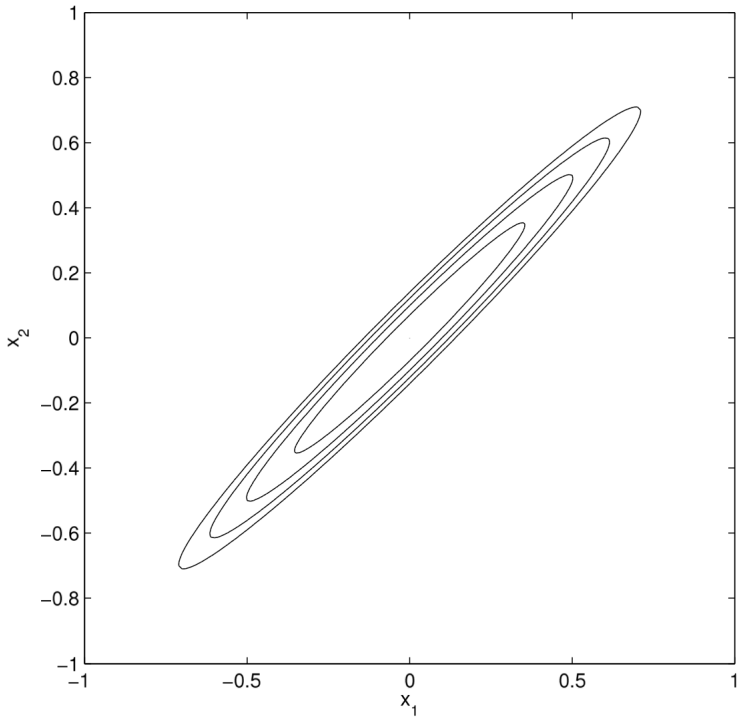

Figure 1: An anisotropic Gaussian distribution

1.4 Applications of MCMC

These samplers have many applications. One is to compute the expected value of something. Let $X_1 \dots X_M$ be samples from the distribution. Then an estimator

for the expected value is given

$$E[\phi(X)] \approx \frac{1}{M} \sum_{i=1}^{M} \phi(X_i).$$

This is the same thing as computing a deterministic integral by phrasing it as an expectation.

$$\int \phi(X) f(X)\, dX = E[\phi(X)] \approx \frac{1}{M} \sum_{i=1}^{M} \phi(X_i).$$

Another major application is Bayesian inference problems. Let θ be a parameter (possibly a vector) that characterizes the dynamics of some system. Suppose we can observe data which depends on θ plus noise. Let D denote a set of observations of the system. We establish a *prior distribution* $P(\theta)$. This represents previously known information, and may be as simple as uniform or log-uniform over a range. Also, we establish a *likelihood* $p(D|\theta)$. Then want to estimate the *posterior distribution* $P(\theta|D)$ using Bayes rule.

$$P(\theta|D) = \frac{p(\theta)p(D|\theta)}{p(D)}$$

Since the normalization constants cancel when doing MCMC simulations, the constant $P(D)$ is typically not known and not used. This is referred to as an *evidence integral* or *partition function*.

There are additional applications to simulations in statistical physics.

2 The Stretch Move Ensemble Sampler

The MCMC algorithm I investigate is called the stretch move [3]. The algorithm has a similar structure to Metropolis, and still uses a proposal and an accept/reject step. In this algorithm, we use a group or *ensemble* of sequences. Each member of the ensemble is called a *walker*. After burn-in, each walker is distributed according to the invariant distribution. On each iteration the algorithm generates multiple samples. Each walker is independent of the other walkers. The walker is correlated with its own previous state and the previous states of other walkers. This is the surprising thing about the stretch move ensemble.

To update one walker, another walker is randomly selected from the *complementary ensemble*. An auxiliary random variable Z is sampled, described below in section 2.1. A move is then proposed alone a line between the current walker k and the complementary walker j according to the distribution

$$Y = X(j) + Z(X(k) - X(j))$$

and then accepted or rejected.

The algorithm is efficient and can be parallelized. The computation required to update a single walker is comparable to other MCMC methods. To accomplish this, split the walkers into two groups X_{red}, X_{black}. First, all the red walkers simultaneously are moved taking the black walkers to be fixed. Once these moves are completed, move all the black walkers with the red walkers fixed.

The algorithm is *affine invariant*, which means that for A, a linear transformation, and vector b, sampling a PDF $g(x) = Af(x) + b$ is equivalent to sampling f then applying A and b after. Heuristically, this means that the algorithm is

not sensitive to skewed distributions. Because of this, the algorithm generally has better autocorrelation time as compared to other MCMC methods.

A single iteration to update one walker is described in algorithm 2. This iteration can be run in parallel for a each group of walkers.

Algorithm 2 Stretch Move Step to update $X_{red}(k,t)$

1: Randomly select a walker from the complementary ensemble $X_{black}(j,t)$
2: Sample $z \sim g(Z)$.
3: Compute a proposed move Y

$$Y = X_{black}(j,t) + z(X_{red}(k,t) - X_{black}(j,t))$$

4: Compute the likelihood ratio

$$q = z^{N-1}\frac{f(Y)}{f(X_{red}(k,t))}$$

5: If $q > 1$, accept. Otherwise, accept with probability q.
6: If accept $X_{red}(k,t+1) = Y$, else $X_{red}(k,t+1) = X_{red}(k,t)$.

The factor of z^{N-1} in the likelihood ratio ensures the invariant distribution is preserved. This is derived in section 2.2.

2.1 The random variable Z

The algorithm requires an auxiliary random variable Z PDF $g(z)$, which must satisfy $g(1/z) = zg(z)$. This distribution is usually $g(z) \propto 1/\sqrt{z}$ on $(1/a, a)$, where $a > 1$. This is the only distribution used in this project and thesis.

To generate the samples of the random variable Z, I will use the standard technique of inverting CDFs. To get the PDF of $g(z)$, define $\alpha = \int_{1/a}^{a} z^{-1/2}\, dz =$

$2\left(\sqrt{a}-\frac{1}{\sqrt{a}}\right)$. Then

$$g(z)=\begin{cases}\dfrac{1}{\alpha\sqrt{z}} & : z\in(1/a,a)\\ 0 & : else\end{cases}$$

The CDF is given

$$F_Z(z)=\begin{cases}0 & : z\leq 1/a\\ \frac{2}{\alpha}\left(\sqrt{z}-\frac{1}{\sqrt{a}}\right) & : 1/a\leq z\leq a\\ 1 & : a\leq z\end{cases}$$

Let $X\sim U(0,1)$, then use the standard relationship of the CDFs, $F_Z(z)=F_X(h^{-1}(z))$ to find the appropriate transformation. Expand each of the CDFs by their definition to find h.

$$\frac{2}{\alpha}\left(\sqrt{z}-\frac{1}{\sqrt{a}}\right)=h^{-1}(z)$$

so

$$h(x)=\left(a-2+\frac{1}{a}\right)x^2+2\left(1-\frac{1}{a}\right)x+\frac{1}{a}$$

Then to sample Z, sample X and return $h(X)$.

2.2 Convergence of the stretch move

First, I will show that the proposal is symmetric. Consider a transition made by multiplying the current state by the random variable Z. Consider the density of making a transition from 1 to an arbitrary value β. The density of this transition is given by $g(\beta)$. Now consider the density moving from β to 1. To be pedantic,

consider this a new random variable $V = \beta Z$. Use the formula form [6] for the PDF of a function of a random variable.

$$f_V(v) \left| \frac{\partial v}{\partial z} \right| = g(z)$$

or

$$f_V(v)\beta = g(z)$$

$$f_V(v) = \frac{1}{\beta} g\left(\frac{v}{\beta}\right).$$

The random variable V must take value 1 to make the correct transition, so the density is

$$f_V(1) = \frac{1}{\beta} g\left(\frac{1}{\beta}\right).$$

Then the required relationship for equality in these transitions is

$$g(\beta) = \frac{1}{\beta} g\left(\frac{1}{\beta}\right)$$

Now, consider the proposal from X to Y.

$$Y = U + z(X - U)$$

Consider this vector equation componentwise.

$$\frac{Y - U}{X - U} = z$$

This is the density to propose $(Y - U)/(X - U)$ starting at one. The proposal from Y to X is given

$$X = U + z(Y - U)$$

or

$$1 = z\frac{Y - U}{X - U}$$

which is the density to propose one starting at the ratio $(Y - U)/(X - U)$. By the symmetry with β shown above, these two densities are equal. Thus, the proposal density is symmetric if g follows the functional relationship. Since the proposal is symmetric, the transition density T does not appear in the likelihood ratio q.

To show the convergence of the stretch move iteration, let f be PDF from which to draw samples. Suppose $X_n \sim f$. That is, the current state follows the invariant distribution. I will show that the transition by the stretch move preserves this distribution. If

$$E\left[\phi(X_{n+1})\right] = E\left[\phi(X_n)\right]$$

for a sufficiently general family of functions ϕ, then the distribution is preserved. Write X for X_n for clarity. All expectations are with respect to f and Z.

Consider the linear integral operator

$$\mathcal{L}\phi(X) = E\left[\phi(X_{n+1})|X\right].$$

The operator $\mathcal{L}$ is called the generator for the Markov chain. Also, let U be a sample from f, independent and constant throughout. Next, expand $\mathcal{L}$ by its definition. There are two possible states for X_{n+1}. The first is a rejection, in which case the new state is also X. The second is an acceptance. The expected

acceptance probability with respect to z, as a function of X and U, is given

$$P_a = \int_z \min\left(1, z^{N-1}\frac{f(U+z(X-U))}{f(X)}\right) g(z)\, dz.$$

The rejection probability is $1 - P_a$. To evaluate $\mathcal{L}$, expand the expectation for rejection and acceptance.

$$\mathcal{L}\phi(X) =$$
$$(1-P_a)\phi(X) + \int_z \min\left(1, z^{N-1}\frac{f(U+z(X-U))}{f(X)}\right) g(z)\phi(U+z(X-U))\, dz$$

The first term is the expected probability of rejection, multiplied by ϕ at the value. Since X does not depend on z in a rejection, $\phi(X)$ comes out of the integral. The second is the expected acceptance. Since the new state is dependent on z, X remains in the integral.

Consider the inner product with respect to the PDF f. With one argument of the constant function $\mathbf{1}$ and the other ϕ, I claim that $\mathcal{L}$ is self adjoint. That is

$$\langle \mathcal{L}\mathbf{1}, \phi\rangle_f = \langle \mathbf{1}, \mathcal{L}\phi\rangle_f. \tag{1}$$

Also, $\mathcal{L}\mathbf{1} = 1$, since expected value of the constant one is always one. The left hand side of (1) is

$$\langle \mathcal{L}\mathbf{1}, \phi\rangle_f = \int_{\mathbb{R}^N} \phi(X) f(X)\, dX = E[\phi(X)].$$

Expanding $\mathcal{L}$, the right hand side of (1) is

$$\langle \mathbf{1}, \mathcal{L}\phi \rangle_f =$$

$$\int_{\mathbb{R}^N} \phi(X) f(X)\, dX \tag{2}$$

$$- \int_{\mathbb{R}^N} \int_z \min\left(1, z^{N-1} \frac{f(U + z(X-U))}{f(X)}\right) g(z) \phi(X) f(X)\, dz\, dX \tag{3}$$

$$+ \int_{\mathbb{R}^N} \int_z \min\left(1, z^{N-1} \frac{f(U + z(X-U))}{f(X)}\right) g(z) \phi(U + z(X-U)) f(X)\, dz\, dX \tag{4}$$

The integral (2) is equal to the left hand side. I claim that integral (4) is equal to (3), so their difference is zero. To see this, change variables in the right hand integral. Use the change of variables

$$Y = U + z(X - U), \qquad \zeta = 1/z$$

which has inverse transformation

$$X = U + \zeta(Y - U), \qquad z = 1/\zeta.$$

This gives the Jacobian as

$$\frac{\partial(X, z)}{\partial(Y, \zeta)} = \left[\begin{array}{ccc|c} \zeta & & & Y_1 \\ & \zeta & & Y_2 \\ & & \ddots & \vdots \\ & & \zeta & Y_N \\ \hline & & & -1/\zeta^2 \end{array}\right]$$

where unspecified entries are zero. This upper left block is $N \times N$, so the modulus of the Jacobian determinant is given

$$\left|\frac{\partial(X,z)}{\partial(Y,\zeta)}\right| = \zeta^N \ \frac{1}{\zeta^2} = \zeta^{N-2}.$$

Apply the change of variables to the integral. The bounds on Y remain as $\mathbb{R}^N$. The bounds on ζ remain $(1/a, a)$, by the symmetry of the range of z specified.

$$\begin{aligned}
&\int_{\mathbb{R}^N}\int_z \min\left(1, z^{N-1}\frac{f(U+z(X-U))}{f(X)}\right) g(z)\phi(U+z(X-U))f(X)\ dz\ dX \\
&\quad = \int_{\mathbb{R}^N}\int_\zeta \min\left(1, \frac{1}{\zeta^{N-1}}\frac{f(Y)}{f\left(U+\zeta(Y-U)\right)}\right) \\
&\qquad\qquad \zeta^{N-2} g\left(\frac{1}{\zeta}\right)\phi(Y) f\left(U+\zeta(Y-U)\right)\ d\zeta\ dY
\end{aligned}$$

Apply the functional relationship $g(z) = \frac{1}{z}g(\frac{1}{z})$ to remove a $1/\zeta$.

$$\begin{aligned}
&= \int_{\mathbb{R}^N}\int_\zeta \min\left(1, \frac{1}{\zeta^{N-1}}\frac{f(Y)}{f\left(U+\zeta(Y-U)\right)}\right) \\
&\qquad\qquad \zeta^{N-1} g\left(\zeta\right)\phi(Y) f\left(U+\zeta(Y-U)\right)\ d\zeta\ dY
\end{aligned}$$

Rename the variables back to X, z for clarity.

$$\begin{aligned}
&= \int_{\mathbb{R}^N}\int_z \min\left(1, \frac{1}{z^{N-1}}\frac{f(X)}{f\left(U+z(X-U)\right)}\right) \\
&\qquad\qquad z^{N-1} g(z)\phi(X) f\left(U+z(X-U)\right)\ dz\ dX
\end{aligned}$$

The integrand is pointwise equal to that of integral (3). To see this, note that the acceptance ratios q are reciprocals. If $\dfrac{1}{z^{N-1}}\dfrac{f(X)}{f\left(U+z(X-U)\right)} \leq 1$, then the

acceptance probability in the other integrand is

$$\min\left(1, z^{N-1}\frac{f(U+z(X-U))}{f(X)}\right) = 1.$$

Comparing integrands

$$\begin{aligned}
&\min\left(1, \frac{1}{z^{N-1}}\frac{f(X)}{f\left(U+z(X-U)\right)}\right) z^{N-1}g(z)\phi(X)f\left(U+z(X-U)\right) \\
&= \frac{1}{z^{N-1}}\frac{f(X)}{f\left(U+z(X-U)\right)} z^{N-1}g(z)\phi(X)f\left(U+z(X-U)\right) \\
&= f(X)g(z)\phi(X),
\end{aligned}$$

which is the integrand of 3 so the equality holds.

If $\dfrac{1}{z^{N-1}}\dfrac{f(X)}{f\left(U+z(X-U)\right)} \geq 1$, then

$$\min\left(1, z^{N-1}\frac{f(U+z(X-U))}{f(X)}\right) = z^{N-1}\frac{f(U+z(X-U))}{f(X)}.$$

Comparing integrands

$$\begin{aligned}
&\min\left(1, \frac{1}{z^{N-1}}\frac{f(X)}{f\left(U+z(X-U)\right)}\right) z^{N-1}g(z)\phi(X)f\left(U+z(X-U)\right) \\
&= z^{N-1}g(z)\phi(X)f\left(U+z(X-U)\right) \\
&= z^{N-1}\frac{f(U+z(X-U))}{f(X)}g(z)\phi(X)f(X) \\
&= \min\left(1, z^{N-1}\frac{f(U+z(X-U))}{f(X)}\right) g(z)\phi(X)f(X)
\end{aligned}$$

so the equality holds here as well. Thus integral (3) and integral (4) are equal. Then the self adjoint property holds. Note according to the formalism derived in [4], detailed balance is equivalent to equality of these integrands.

Also, in terms of expectations, the right side is

$$\begin{aligned}\langle \mathbf{1}, \mathcal{L}\phi \rangle_f &= \int_{\mathbb{R}^N} E\left[\phi(X_{n+1})|X\right] f(X) dX \\ &= E[E\left[\phi(X_{n+1})|X\right]] \\ &= E\left[\phi(X_{n+1})|X\right]\end{aligned}$$

Then

$$E\left[\phi(X_{n+1})|X\right] = E[\phi(X)]$$

for arbitrary ϕ so the iteration preserves the invariant distribution. Thus, the stretch move iteration converges to this distribution.

3 The Stretch Move on the GPU

A major component of the project is an implementation of the stretch move for GPU parallel hardware. I discuss relevant issues and definitions for OpenCL, the implementation, performance and user interface. I show some results on performance, autocorrelation time and acceptance rates.

3.1 A short description of GPUs and OpenCL

This package is written in C and OpenCL. OpenCL defines a library interface and programming language based on C99 that allow the user to access the GPU for general computations. The program execution in controlled by a standard CPU program. The CPU is also referred to as the *host*, and the GPU as the *device*. An OpenCL program that runs on the device is called a *kernel*. The device may not be a GPU, as OpenCL also runs on other types of hardware, but I will focus on

the GPU here.

A GPU has a lot of parallelism on a small, relatively inexpensive chip. A 2013 era GPU has thousands of physical cores and a full memory system, distinct from that of the CPU. The GPU has a single, large piece of RAM called that is accessible by all threads. This memory is physically separate from the compute units (described below) and is referred to as "off-chip memory." The host can read or write data from off-chip memory directly.

The physical cores are organized into groups called a *compute units*, which includes a single set of hardware for instruction fetch and decoding. The individual physical core is referred to as a *processing element* or *SIMD lane*, which includes a separate arithmetic logic unit (ALU), but no cannot do instruction decoding. The execution of instructions on a compute unit is SIMD, or "single instruction, multiple data." The single instruction fetched by the compute unit is run on all processing elements simultaneously, in parallel.

Each compute unit has a small, fast piece of memory called on the same physical chip as the compute units, and is referred to as "on-chip." A compute unit shares a physical piece of on-chip memory, which is shared between threads running on the unit, but is inaccessible to threads on other compute units. There is a still smaller, very fast piece of memory called *registers*, that is owned associated with a single processing unit, which is also physically on-chip. This is the default memory location for literals and statically defined arrays. If a kernel uses too much on-chip memory, OpenCL will place the data in off-chip. This is called a *spill*, and usually results in bad performance. The host cannot directly access on-chip memory; they can only be accessed by an OpenCL kernel.

To run software on this hardware, OpenCL automatically manages the parallel

execution of threads on the GPU. An OpenCL thread is called a *work-item*. The work-items run in a group of threads called a *work-group*. Each work-group is executed on a compute unit in SIMD. This means that a work-group size that is a multiple of the SIMD width usually gives best performance. Instructions that would not be executed in a serial program (for example, an else clause in an if statement that evaluates to true) are still executed, but their values are thrown away. This means that branches and loops which run for a variable number of iterations make performance slow. The OpenCL can also perform latency hiding, which means that it can execute instructions while waiting for memory reads and writes to complete.

To manage memory, the programmer specifies where they intend the data to be stored. If they specify *global memory*, the data is stored off-chip (assuming it fits). Attempts to allocate more global memory than the size of the off-chip memory will usually result in runtime errors. If the programmer specifies *local memory*, OpenCL will attempt to place the data in on-chip memory. Local memory must be managed manually by the programmer — all allocations and accesses must be explicitly declared. If they specify *private memory* (or nothing, this is default), then OpenCL will attempt to place the data in registers. For data in local and private memory, there is always a risk that the on-chip memory will become full, and spills will occur. In the even of the spill, all variables behave semantically in the same way — a private variable still cannot be accessed by any other work items. However, this can be a major source of performance loss. OpenCL includes barriers to synchronize within global and local memory within a kernel. However, to synchronize memory traffic between compute units requires that the kernel is stopped.

3.2 Stretch move kernel design

The stretch move kernel updates a group of walkers in parallel. The code is parallel across the entire walker update. Each time the kernel is called, proposals are made in parallel to update each walker. The likelihood function is evaluated, the ratio q is computed and walker positions are updated. Before updating the next group, the kernel must stop to ensure that memory traffic from updates is complete.

One alternate strategy is to compute the proposals in serial on the CPU and send the likelihood evaluations to the GPU to be evaluated in parallel, then send the likelihood back then perform the accept/reject step. This is a worse strategy, because it involves sending high latency messages to the GPU. Computing proposals and accept/reject may be cheap relative to the likelihood, but there is no reason not to compute it on the GPU.

The strategy for memory use is straightforward. The walkers are kept as a "master copy" in global at all times. Each work-item owns a single walker in each work-group, and updates only that walker. A *race condition* occurs when there is ambiguity in the order of two or more operations, and the result depends on the order. Usually, this occurs when one thread attempts to read a location in memory at nearly the same time another writes to the same location. Then it is not defined wether thread reads the new or the old value. A race condition leads to undefined behavior or worse. This clear ownership eliminates race conditions in updating the walkers. On each iteration, the walker is read once to compute a proposal. Thus, the best strategy is to keep the walkers in global, as reading to local on each kernel call doesn't save any memory traffic.

The proposal can be kept in private or local memory. Private memory is faster, but risks "spilling" to global for problems of even moderate dimension. Local is

slower than if the array stays in private, but is much faster if it spills to global. For many problems, such as the planet fitting described in section 5, storing and reading the proposal is not a dominant cost. This choice then makes little difference.

If there is constant data in the problem, it is usually beneficial to place it in local memory. Suppose there are N_{obs} total pieces of data and the workgroup size is W. Since the data is constant, and local memory is shared within a work group. Then each work-item needs to read N_{obs}/W total pieces of data from global memory. If W is a standard work-group size such as 64 or 128, this saves a lot of global memory traffic. If the data is of too large to fit in local memory, then this speedup is not available. It may be possible to use more sophisticated strategies, but this is dependent on the particular distribution that is being sampled.

Communication and computation are overlapped. When updating the a group of walkers in parallel, the other walkers are remain unchanged. Thus the constant walkers can be sent to the host while the current group is being updated. To accomplish this, the implementation uses two OpenCL queues. This allows manual control of the overlap and required synchronization. For expensive PDF's, the communication is completely hidden. This has shown up to a two fold total speedup compared to no overlap. For very inexpensive PDF's, the time to sample is short compared to latency of a single message. The overlap improves computation, but since most of the time is waiting for memory traffic, the GPU cannot be kept busy. This is not fantastic. However, if this happens the problem is fast overall so this is usually not an issue.

3.3 User interface design

Great care has been taken to make the code easy to use. To get started, the user only needs to follow a few simple steps:

1. Place generic C code to evaluate the log of the probability density function to sample in "pdf.h."

2. Pick one of the included examples in "stretch_move_main.c" to run. Check that parameters in the example match your problem, which usually means setting the correct dimension.

3. Type "make" on the command line to build the code with the included makefile.

4. Type "./stretch_move_main" to run the executable. The code will prompt to select an OpenCL implementation and hardware, then run the sampler.

More detail about each step follows.

3.3.1 Setup

In more detail, here are instructions for setup for sampling.

First, specify the PDF. For numerical reasons, the code requires the logarithm of the PDF, rather than the PDF itself. The PDF does not need to be normalized. To do this, place your code in the file "pdf.h." The PDF function is written standard C, with a few restrictions. There is no access to I/O, so all data must be passed to the function in memory rather than read from a file. There is also no dynamic memory allocation, recursion or function pointers.

Suppose one wanted to sample an N dimensional Gaussian with PDF

$$f(\vec{X}) \propto \exp\left(-\frac{1}{2}\sum_{i=0}^{N}(X_{i+1} - X_i)^2\right).$$

where by definition $X_0, X_{N+1} = 0$. Then the log PDF function would be implemented as

```
float log_pdf(PROPOSAL_TYPE *x,
        __global const data_struct *data_st, DATA_ARRAY_TYPE *data){
    float sum = x[0]*x[0] + x[NN-1]*x[NN-1];
    for(int i=0; i<NN-1; i++){
        sum += (x[i+1] - x[i]) * (x[i+1] - x[i]);
    }
    return -0.5f * sum;
}
```

The dimension "NN" is set up at kernel compile time by the initialization routine and is always available in this function. The data structure "data_st" and array "data" are passed even if they are not used. The types `PROPOSAL_TYPE` and `DATA_ARRAY_TYPE` are always float, but using this definition allows easy changes from global to local, as described below. This is for generality, since the package handles all the data movement.

Next, define needed constants in "constants.h." These may define the dimension of the problem or observations, values for the prior, loop bounds, or anything else.

Also, if the PDF requires additional data, add the types to "data_struct.h." Because of OpenCL scoping rules, you must also specify this structure again in "pdf.h" for the kernel to use. By default this contains a float, you may add any

scalars or small statically defined arrays.

If you wish to include arrays, use the buffer "data." This is a single float array, and all the arrays must be packed into this array. OpenCL is not flexible to complicated structures involving variable numbers of pointers or pointers to pointers. The best general structure is to insist that the user gets one data array, and should unpack it manually.

As mentioned above, data is placed into local memory. If you have too much data to fit in local, remove the definition

```
#define USE_LOCAL_DATA
```

from "pdf.h." This will automatically place the data in global memory in all necessary places. This simple switch shields the user from updating OpenCL code for local memory management.

Consider sampling a multivariate normal with mean $\vec{\mu}$ and covariance matrix Σ. The PDF is given

$$f(\vec{X}) \propto \exp\left[-\frac{1}{2}(\vec{X}-\vec{\mu})\Sigma^{-1}(\vec{X}-\vec{\mu})\right].$$

We want to pass a vector "mu" of length NN for the mean, and an NN by NN matrix for the inverse covariance "inv_cov." Set these to be contiguous in memory in the array "data" which is passed to the initialization routine. For example, allocate "data" to be a length NN + NN^2 float array. Suppose one wanted $\mu = (0, 1 \ldots$ NN-1) and Σ^{-1} to be a (-1,2,-1) tridiagonal matrix. Initialize the data as follows:

```
for(int i=0; i < dimension; i++)
    data[i] = (cl_float) i;
for(int i=dimension; i < data_length; i++)
    data[i] = 0.0f;
for(int i=0; i < (dimension-1); i++){
```

```
        data[ i   +     i*dimension + dimension ] =  2.0f;
        data[ i+1 +     i*dimension + dimension ] = -1.0f;
        data[ i   + (i+1)*dimension + dimension ] = -1.0f;
    }
    data[ (dimension-1) + (dimension-1)*dimension + dimension ] = 2.0f;
```

The package handles moving the packed array to the device.

In your code for the PDF, set pointers or otherwise read the data. For example:

```
// The first elements of the data array are the means
DATA_ARRAY_TYPE *mu       = data;


// NN elements later is the inverse covariance matrix
DATA_ARRAY_TYPE *inv_cov  = data + NN;
```

Now these pointers can be used as normal arrays to evaluate the PDF. If you unpack data this way, it must have the same scope and address space as the data array. Use pointers (rather than reallocate and copy) to avoid wasting time on memory operations. This is illustrated in full in the second code example.

3.3.2 Sampling

The package maintains a set of data structures. The user creates a sampler object by calling an initialization routine. This structure contains parameters about the run, and allocates arrays on host and initializes the walkers. It starts an OpenCL context and two queues, then allocates device buffers and transfers the device as appropriate. It also compiles and initializes the random number generator and finally compiles the stretch move OpenCL kernel.

An example of calling the initialization is as follows. Set up some parameters for your run, start with the chain length. This is the number of ensemble samples that will be run when you start the sampler.

```
cl_int chain_length = 100000;
```

Next is the dimension of the problem:

```
cl_int dimension = 10;
```

Set the size of each half of the ensemble, which also corresponds to the total parallelism available, and the work group size, which must divide the number of walkers per group.

```
cl_int walkers_per_group = 2048;
size_t work_group_size   = 128;
```

Set the value of a, the parameter for Z as described in section 2.1. Also set the PDF number, which allows the user to switch the PDF without large modifications to the source.

```
double a                 = 1.5;
cl_int pdf_number        = 1;
```

Tell the sampler which variables are important, and which are nuisance variables that can be thrown out.

```
cl_int num_to_save       = 2;
cl_int *indices_to_save  = (cl_int *) malloc(num_to_save * sizeof(cl_int));
indices_to_save[0]       = 0;
indices_to_save[1]       = 3;
```

Call the initialization routine to allocate all the necessary arrays and compile the OpenCL kernels.

```
sampler *samp = initialize_sampler(chain_length, dimension,
                                   walkers_per_group, work_group_size,
                                   a, pdf_number,
                                   data_length, data,
                                   num_to_save, indices_to_save,
                                   CHOOSE_INTERACTIVELY, CHOOSE_INTERACTIVELY);
```

Use these default values for the last two parameters, which allows the user to select the device interactively. Despite the modestly messy data structure, and setup of OpenCL objects, the user only needs to call this one routine to get everything set.

Now, the user can run the burn-in:

```
int burn_length = 5000;
run_burn_in(samp, burn_length);
```

The sampler is now ready. Run it:

```
run_sampler(samp);
```

The array `samp->samples_host` is now filled with samples ready for use. Samples are stored in "component major" order, so to access component `i` of sample `j`, use

```
samp->samples_host[i + j*samp->N];
```

This set of parameters and functions is all the user should need to sample most distributions. Crucially, none of this involves coding in OpenCL. By following this simple set of instructions, any user can take advantage of GPU parallelism.

There are also basic utilities included for computing mean, variance, autocorrelation time and histograms.

3.4 Related software

One related package is called emcee: the MCMC hammer [2]. This package implements the Stretch Move in Python and has been successfully used in inference problems. The algorithm is sufficiently new that there are not many libraries that use it, so if a significant speedup would have real impact. The code here has performed significantly faster than emcee on some test problems.

The random number generator used is ranluxcl [8]. The kernel requires precisely three random numbers per update of a single walker. The generator takes an

integer from $0 \ldots 4$ as a "luxury" level, which allows the user to request higher quality random numbers with a possible increase in performance cost. The default is the maximum value of 4.

3.5 Histograms and compute performance

In this section, I measure and discuss compute performance. All tests in this section were run on an Nvidia GeForce GTX 590 [9]. The GPU has an instruction clock speed of 1.2 GHz. It has 1024 total compute-units with a SIMD width of 32.

For the sake of comparison, I use the Gaussian with PDF

$$f(\vec{X}) \propto \exp\left(-\frac{1}{2}\sum_{i=0}^{N}(X_{i+1} - X_i)^2\right)$$

with $X_0, X_{N+1} = 0$ by definition, throughout.

Figure 2 shows a correct histogram on this problem.

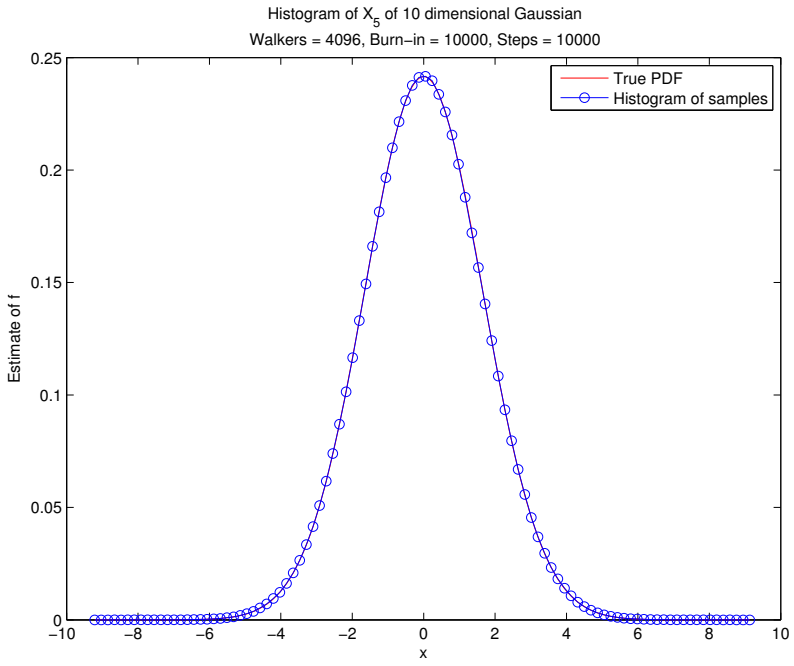

Figure 2: Correct histogram on basic debug problem. Note that the true PDF and the histogram are indistinguishable.

Figure 3 illustrates a common problem. This is Gaussian debug problem, which can be easily sampled by direct methods. The stretch move fails entirely due to inadequate burn-in. The histogram still has a Gaussian-like shape, but this is likely because of the central limit theorem or a partially converged sampler.

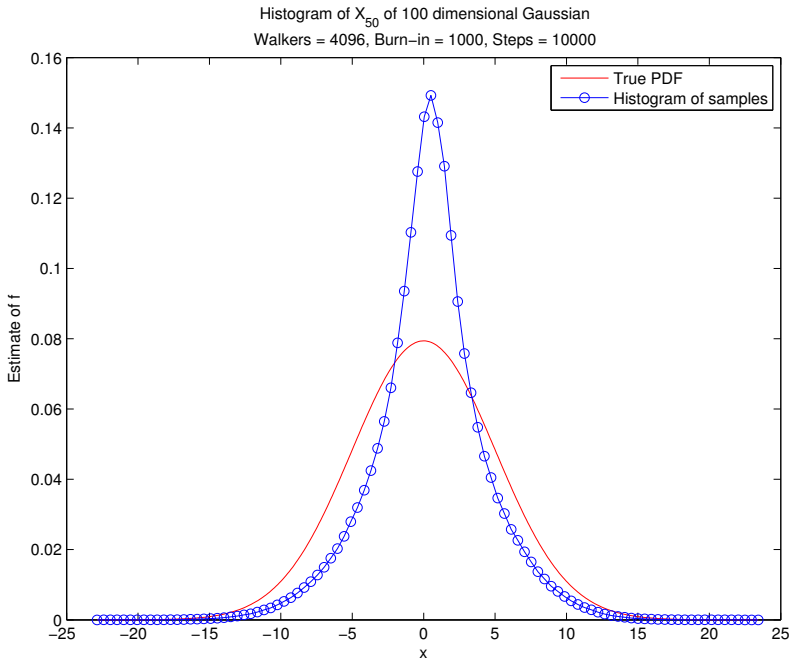

Figure 3: Complete failure to find correct distribution due to inadequate burn-in.

Figure 4 shows convergence on the same problem. More burn-in gives a correct result, even though the other parameters are the same. The number of walkers used or the total samples collected are not changed. Note that the fit is not as clean as the ten dimensional case, despite the additional burn-in.

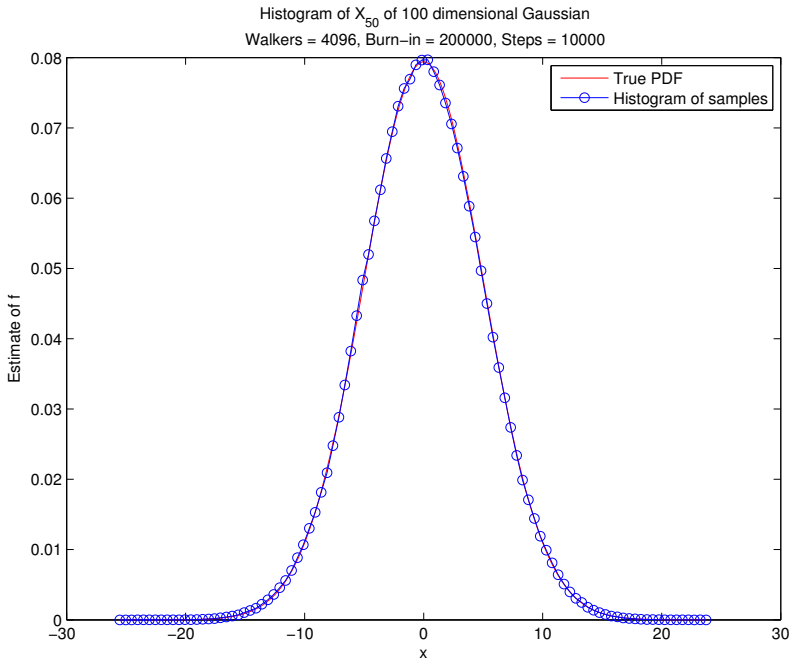

Figure 4: Convergence achieved on 100 dimensional Gaussian

Figure 5 shows the scaling with total number of work-items, which is equal to the total parallelism in the problem. The hardware is not changing, this is run on a single GPU. The figure shows a rapid increase in performance from few work-items to many more. This corresponds to filling the cores of the GPU. For larger numbers of total work-items, the performance continues to increase, but more slowly. This is likely due to further benefits of latency hiding with more work pushed to the GPU. At the highest position, the performance has stabilized using the full parallelism available for this hardware and problem.

The performance curve is not smooth. Performance is better for multiples of 1024, and drops for work-item sizes of the form $1024n + 128$. This is because the hardware has 1024 total physical compute units. Using $1024n + 128$ work-items means that some SIMD units have more work groups to process than others. In

other words, multiples of 1024 have better load balancing. Latency hiding reduces some of this loss. This is why performance is reduced by about .8 and not a factor of two.

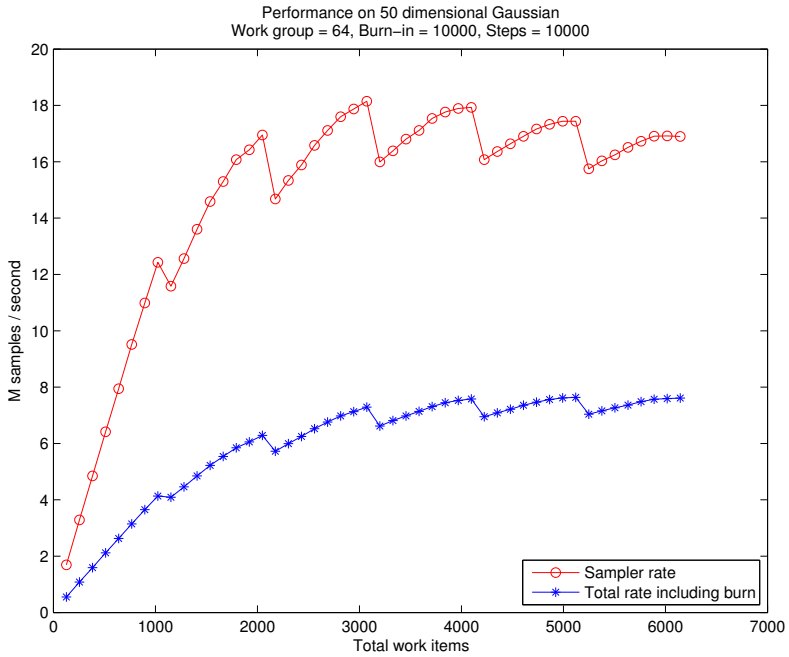

Figure 5: Scaling with changes in total work-items. Note that the number of walkers is twice the total number of work-items.

Figure 6 shows performance scaling with dimension. Note that twice as many burn-in steps are run as sampling steps. Performance diminishes with increased work roughly as expected. The drop is very high at first, because the in the five dimensional case the PDF is very fast to evaluate and most time is spent in launch overhead. As the dimension increases, the cost of evaluating the PDF increases and performing the computation becomes more relevant.

This plot shows a decrease in performance for odd dimensions. This is likely due to memory access patterns and cache alignment. For future work, it would be beneficial to explore memory padding strategies.

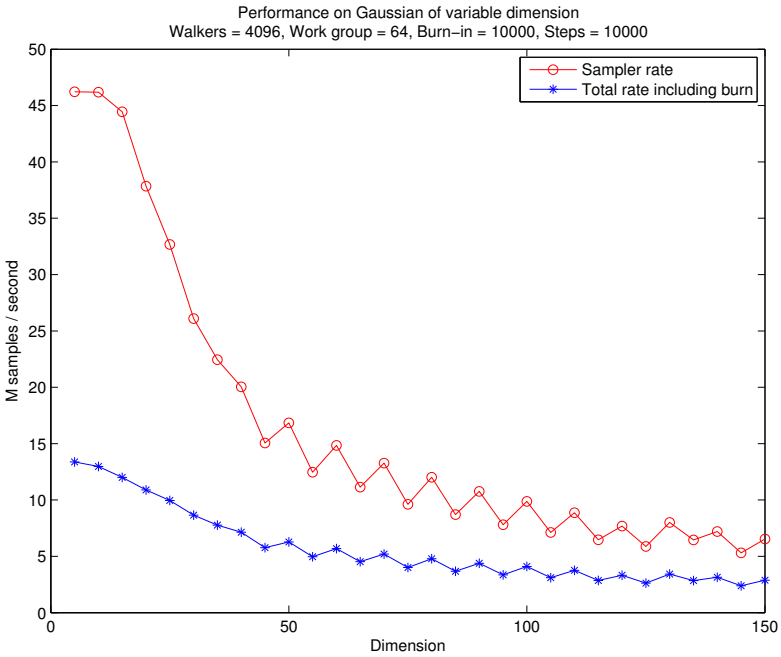

Figure 6: Performance scaling with changes in dimension

Figure 7 shows decrease in acceptance rate with increase in dimension. Also, since the burn-in required for very large dimensions is high, I use a long burn throughout this test, even though it is unnecessary for smaller problems. The length parameter a is two for this test.

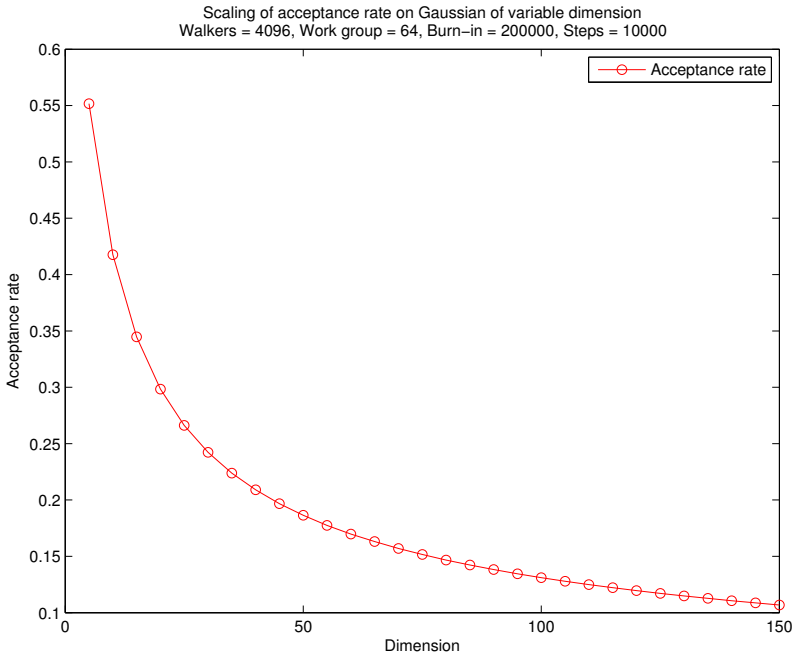

Figure 7: Decrease in acceptance rate with increase in dimension

Figure 8 shows the scaling of the τ with the dimension. The parameter a is selected as $\min(2, 1 + 30/N)$. To pick this ratio, I tuned a to have an acceptance rate of approximately .35 for a 100 dimensional problem. I also wanted a to be of the form $1 + c/N$, so solved for c according to the 100 dimensional problem. The maximum a is capped at the standard value of 2. This resulted in an improvement of τ for high dimensions by a factor of approximately 2 compared to the standard $a = 2$.

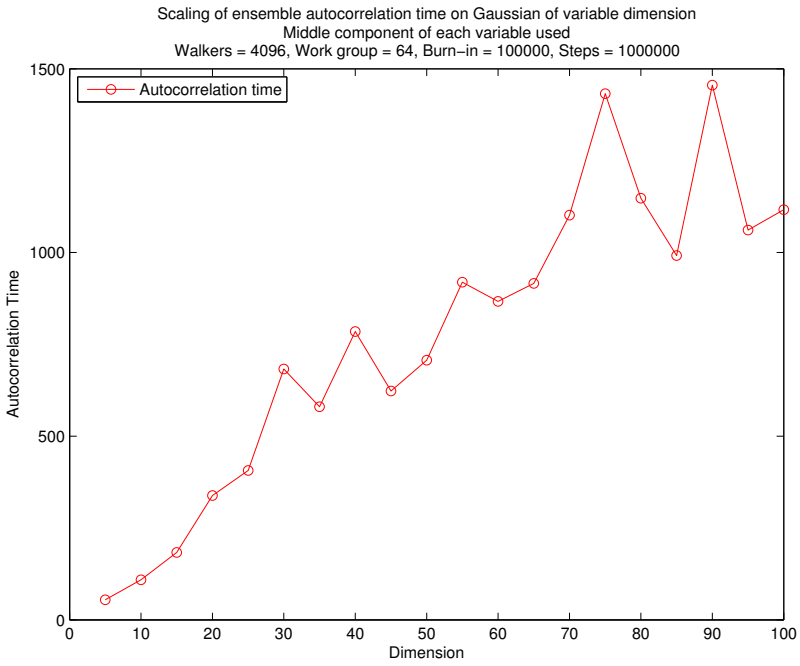

Figure 8: Increase in autocorrelation time with dimension

4 Sampling a Restricted Distribution

In this section, I investigate sampling a distribution with restrictions. The distribution of interest is

$$f(\vec{X}) \propto \exp\left(\sum_{i=0}^{N}(X_{i+1} - X_i)^2\right) r(\vec{X})$$

where $r(\vec{X}) = 1$ if all components of X are nonnegative and zero otherwise.

This is an interesting problem as a toy model for entropic repulsion, the physics of which are explored in more complex setting in [1]. Also, many prior distributions for inference problems have restrictions. This simple distribution exposes mathematical behavior which occur in posterior sampling. The restriction moves the mean of each component to one side, sometimes inducing skewness or bias. Also, some samplers have difficulty with high autocorrelation time or sample rates on these distributions. The performance is evidence that the stretch move is robust for many distributions.

Figure 9 shows a histogram of components of this distribution. The peak of the first component is near the origin. The peaks of subsequent components have peaks increasingly far from zero, with the center component being the furthest away.

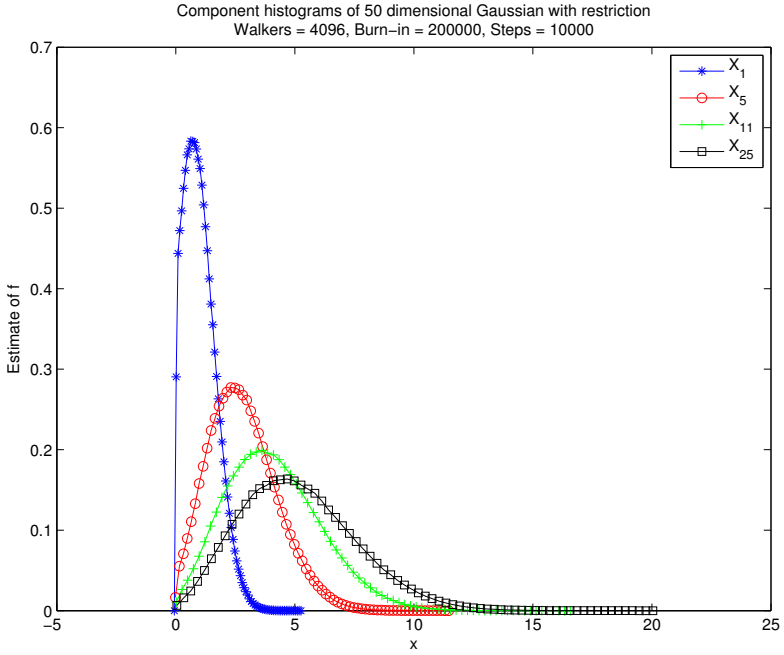

Figure 9: Histograms of three components of a Gaussian with restriction

Also interesting is the difference between the unrestricted PDF and the histogram from restriction. The middle component, X_{25} of the same 50 dimensional problem is shown in figure 10.

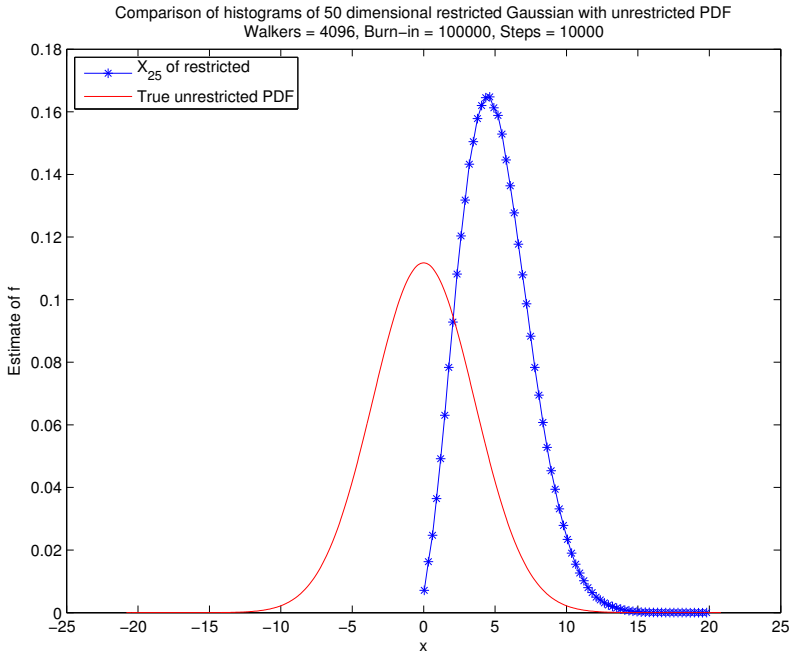

Figure 10: Comparison of histogram of Gaussian with restriction and PDF of unrestricted Gaussian

Computational performance is unaffected by the restriction. This is evident comparing figure 11, which shows that the performance on the restricted distribution, and figure 6, which shows the unrestricted.

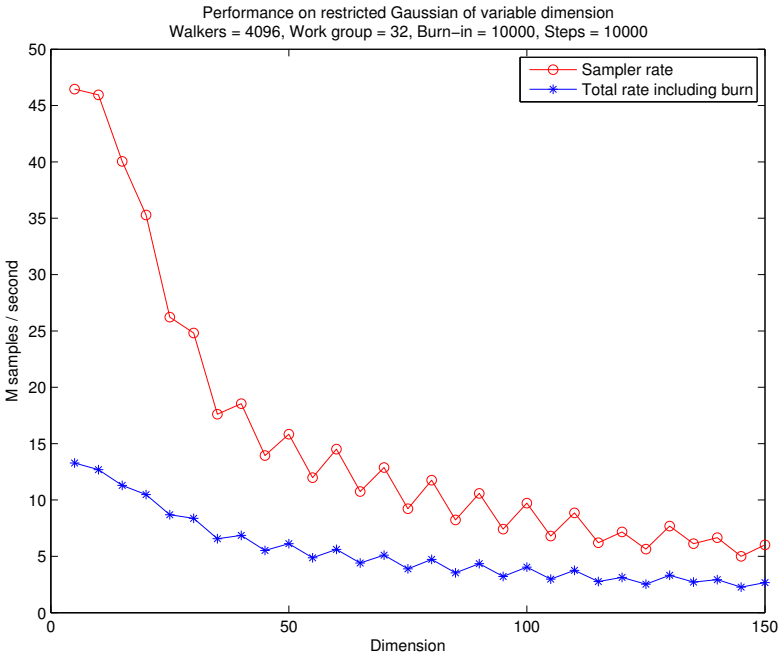

Figure 11: Compute performance on restricted Gaussian

Another question is the variation in autocorrelation time. Figure 12 shows the growth of τ with the dimension. As in figure 8, a is selected as $\min(2, 1 + 30/N)$. The autocorrelation time is larger on the restricted distribution, but remains within the same order of magnitude. This resulted in an improvement of τ for high dimensions by a factor of approximately 4 compared to the standard $a = 2$. This is a larger reduction than in the unrestricted case.

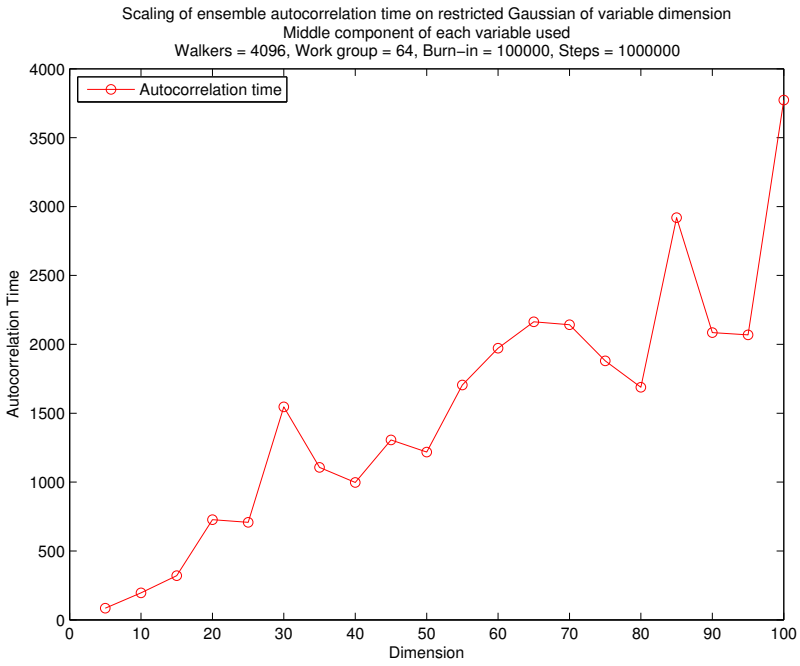

Figure 12: Scaling of autocorrelation time on restricted Gaussian

5 Exoplanet Fitting Using Inference

Another application is exoplanet fitting. Astronomers have developed techniques to measure the radial velocity of a star using spectroscopy. Observations of the radial velocity are used as evidence of planets orbiting a star. The orbit parameters of the planet can then be inferred by Bayesian posterior sampling. I have investigated the application of the GPU parallel sampler to this problem.

Here I summarize the problem as described in [5]. I assume that the planets follow a Keplerian orbit. The interactions between the any pair of planets is negligible. The radial velocity depends on a set of parameters, and can be computed by Kepler's classical formulas. The first parameter is v_0, the velocity offset, or the approximate radial velocity of the entire system. Each planet contributes a perturbation Δv to the radial velocity of the star. Then the radial velocity is given

$$v_{rad}(t) = v_0 + \sum_{i=1}^{P} \Delta v_i(t)$$

where P is the total number of planets orbiting the star. Five independent parameters for each planet determine the orbit and thus Δv. They are A, the amplitude, ω, the mean angular velocity, ϕ, the phase of pericenter passage or angle offset, e, the eccentricity of the orbit and ϖ, the longitude of periastron, the angle of the ellipse with our view. The perturbation at time t is computed with the algorithm 3.

I wish to put a prior on the period, rather than the mean angular velocity. Thus, I use $p = 2\pi/\omega$ in the code instead. There is one extra parameter, the jitter s, which is overall system noise. Define θ to be a vector containing all these parameters. That is $\theta = (A_1, p_1, \phi_1, e_1, \varpi_1, \ldots, A_P, p_P, \phi_P, e_P, \varpi_P, v_0, s)$.

Algorithm 3 Compute $\Delta v(t)$

1: Compute the mean anomaly M as

$$M = \omega t + \phi.$$

2: Solve the implicit equation

$$M = E - e \sin E$$

for the eccentric anomaly E.

3: Compute the true anomaly f according to

$$\cos f = \frac{\cos E - e}{1 - e \cos E}.$$

4: Compute the perturbation for this planet

$$\Delta v(t) = A \left[\sin(f + \omega) + e \sin(\varpi) \right].$$

Now, I set up the necessary distributions for inference. The data in this problem involves N_{obs} measurements of the radial velocity. Denote the data $D = (t_1, v_1, \sigma_1, \ldots t_{N_{obs}}, v_{N_{obs}}, \sigma_{N_{obs}})$ where t_i is the observation time, v_i is the radial velocity and σ_i is the estimated standard deviation on this observation. The prior is uniform or log-uniform (Jeffreys prior) on each component of θ, see [5]. The likelihood is given by

$$P(D|\theta) \propto \prod_{i=1}^{N_{obs}} (\sigma_i^2 + s^2)^{-1/2} \ \exp\left[-\frac{1}{2} \sum_{i=1}^{N_{obs}} \frac{(v_i - v_{rad}(t_i))^2}{(\sigma_i^2 + s^2)} \right].$$

The dependence on θ is implicit in the value $v_{rad}(t)$. The denormalized log of the PDF is given

$$l(D|\theta) = -\frac{1}{2} \left[\sum_{i=1}^{N_{obs}} \frac{(v_i - v_{rad}(t_i))^2}{(\sigma_i^2 + s^2)} + \log(\sigma_i^2 + s^2) \right].$$

I sample from the distribution

$$P(\theta|D) = \frac{P(\theta)\ P(D|\theta)}{P(D)}$$

This is the standard setup for Bayesian inference problems. The partition function or "evidence integral" $P(D)$ is a constant for fixed observations. This cancels in the acceptance ratio q and is ignored.

5.1 Results

The sampler correctly finds histograms on a one planet fit. Results on a simple set of synthetically generated data are discussed here. The noise on the individual observations, $\sigma_i \sim U[0,1]$. The total noise added to each observation is $\mathcal{N}(0, \sigma_i^2 + s^2)$. Posterior histograms are shown in figures 13 and 14.

The parameters for this run are as follows. The initial time is 0.00, final is 20.00 with a time step of 0.10. The total number of observations is 201. This run was performed with a chain length of 100 thousand. Simulated annealing was run with a cooling schedule of $1/20, \ldots 1/2, 1$. Each temperature of the cooling schedule is run for 50 thousand steps. The burn-in is run for 1.0 million steps. The number of walkers is 2048 and work-group size is 32. The sample rate is 0.44 M samples/second for kernel time and 0.02 M samples/second for total time. The reason for this large disparity is that very long simulated-annealing and burn-in were run. The tuning parameter a is 1.80. The acceptance rate is 0.541.

Table 1 shows some statistics about the parameters estimated.

	true	est max likelihood	mean	65% CI	95% CI	std dev	autocorr time
a	15.000	14.847	14.833	± 2.245e-06	± 4.489e-06	0.000102	80.3
p	8.000	8.036	8.036	± 4.537e-07	± 9.075e-07	0.000021	286.0
ϕ	2.000	2.024	2.034	± 3.553e-07	± 7.107e-07	0.000016	125.2
e	0.500	0.494	0.494	± 1.226e-07	± 2.452e-07	0.000006	94.2
ϖ	3.000	3.037	3.043	± 3.529e-07	± 7.057e-07	0.000016	106.1
s	1.000	1.068	1.092	± 1.099e-06	± 2.197e-06	0.000050	105.5
v_0	4.000	3.933	3.933	± 1.595e-06	± 3.189e-06	0.000072	137.4

Table 1: Parameters and confidence intervals for planet fitting problem

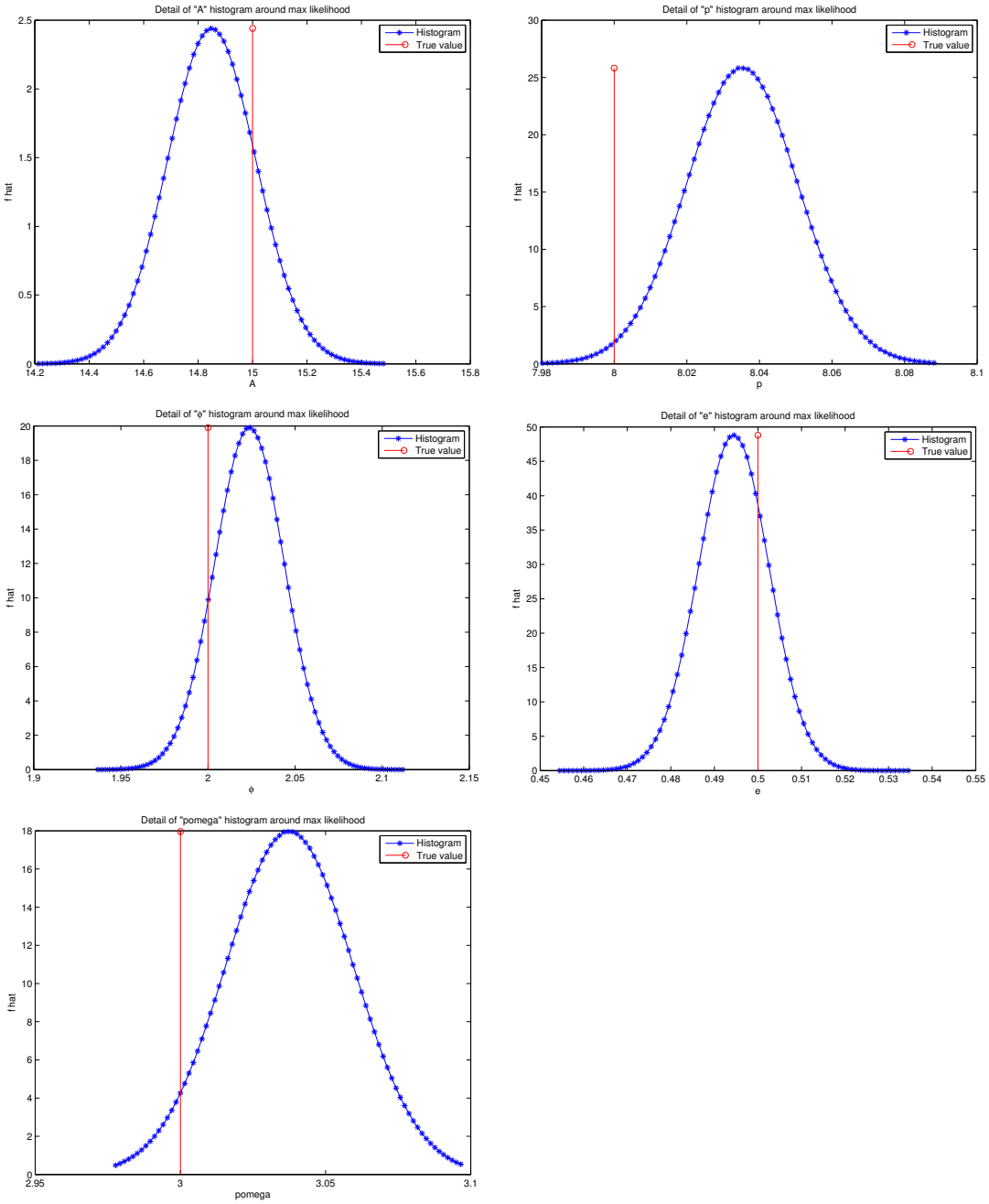

Figure 13: Posterior histograms on a one one planet fit, planet parameters

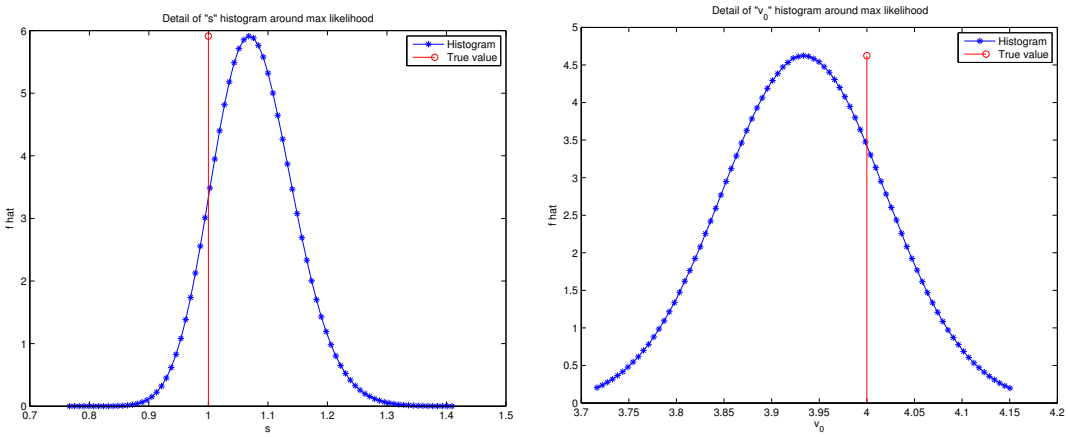

Figure 14: Posterior histograms on a one one planet fit, reference velocity and jitter

The posterior distributions have multiple peaks, many of which are far from the true values. A very long burn-in and simulated annealing run allows the sampler to leave these other local maxima. The walkers then all end up in a neighborhood of the true value. One cause is periodic variables. If the orbit has period p, then it is also $2p$ periodic. I have observed local optima at twice the true value. On variables that are angles, ϕ and ϖ, multiple peaks occur if the true value is close to the boundary. Values near the opposite boundary are far on the linear histogram, but they are close in angle. This creates a strong local maximum far from the true value. Walkers frequently get stuck in this region. There are extraneous peaks which lack a clear explanation, beyond that local optima happen. For example, sometimes there are extra peaks on s. Perhaps the sampler has difficulty distinguishing noise on each component and jitter.

The biggest impact on performance was modifications to the Newton's solve for step 2 of algorithm 3. The right hand side is strictly monotonic, so the equation has a unique solution. However, as e approaches one, the derivative approaches zero

at points, and it is possible for the Newton's iteration to make a bad guess. This sometimes causes very long iterations or iterations that fail to converge entirely. The instructions are SIMD, so if one Newton's solve is very slow then an entire work-group must wait.

There are two strategies that I have tried to improve the iteration. One is restarting the iteration. The true solution is bounded as $M - e \leq E \leq M + e$. If a Newton's iteration leads to a value outside the range, I restart the iteration halfway between the current state and the boundary. Another strategy is to use a Taylor series for sine to get a good initial guess. Suppose e is near one.

$$M = E - e\sin(E) \approx E - e(E - E^3/6) = (1 - e)E + eE^3/6.$$

Since e is near one, the first term can be neglected. The initial guess is

$$E = \left(\frac{6M}{e}\right)^{1/3}.$$

For e close to one, this is a good approximation to the solution.

These strategies have a dramatic influence on performance. Table 2 shows timing on various Newton's strategies with $e = .95$. There are 4096 walkers with a work group size of 64. The maximum iterations is set to 1000 so that the algorithm does not terminate early. First, I tested with $M = E$. This provides an estimate of the cost of running the Newton's method. For $e = .95$ near one as shown in the, this experiment reveals the Newton's solve takes over 80% of the total time. This is improved slightly on the restart. With both restart and Taylor series, the Newton's method takes around 56% of the total time. This is a over a two fold speedup on the entire sample rate, not just Newton's, from basic Newton's method.

Strategy	Sample rate (M samp / s)
$E = M$ (no Newton's)	1.7181
Basic	0.2744
With restart	0.3766
Taylor series and restart	0.6224

Table 2: Sample rates for different strategies on Newton's method. Eccentricity $e = 0.95$

There are many potential improvements for future work. One is to apply a clustering algorithm. This would improve problems with the local optima and get better overall estimates. Another is to consider a nonlinear change of variables that would eliminate the problems with angles.

6 Inference Problems with Stochastic Dynamics

The next problem I consider is inference problems with stochastic dynamics. I create numerical solutions to stochastic differential equations, and make noisy observations of the numerical solutions. I then estimate the coefficients from these noisy observations. The noise in the solution itself creates new difficulties that are not present with deterministic dynamics.

6.1 Model

The setup is as follows. Let N denote the total dimension of the sample, X the state variable and N_X the number of X components. Let Y denote the observation and N_Y its dimension. Also, denote the number of model parameters as N_θ and the number of steps in the path as N_{steps}.

Each sample also keeps a path $X(1) \dots X(N_{steps})$ for the solution of the dynamics. The total dimension of the state variable is $N = N_\theta + N_{steps}$, which includes all the parameters and the state at each discrete time. The initial condition $X(0)$ is fixed and known throughout. Let f describe the deterministic part of the dynamics and g the deterministic part of the observation.

6.1.1 Dynamics

The dynamics evolves according to the SDE

$$dX = -aXdt + \sigma dW$$

where dW is Brownian motion.

The observations are given

$$Y = bX + \zeta Z$$

where $Z \sim N(0, 1)$. All observation noise is assumed to be independent.

6.1.2 Prior

I use a uniform prior

$$P(\theta) \propto 1$$

if θ is in a particular range. The range is problem specific, with defaults shown in table 3.

	min	max
a	-10	10
σ	0	10
b	-10	10
ζ	0	10
X	-100	100

Table 3: Default boundaries for the uniform prior

As discussed below, changes to these boundaries can make a big difference in convergence and accuracy.

6.1.3 The likelihood function $P(D|\theta)$

Define $X_{j,pred}$ to be the deterministic model update by integrating the deterministic part of the dynamics. The initial condition for this timestep is the previous sample value. Define $X_j(t)$ to be the current value in the sample. Define $Y_{j,pred}(t) = g(X_j(t))$. This is the no-noise prediction for the given dynamics. It is dependent

on the X in the current sample according to the observation model. Define $Y_{j,obs}(t)$ is the true noisy observation at the same time. Finally, define h to be the timestep for the dynamics.

The distribution is given

$$P(D|\theta) \propto \frac{1}{(\sigma_1 \dots \sigma_{N_X})^{N_{steps}}} \frac{1}{(\zeta_1 \dots \zeta_{N_Y})^{N_{obs}}} \exp\left[-\frac{1}{2}\left(\sum_{t=1}^{N_{steps}} \sum_{j=1}^{N_X} \frac{(X_{j,pred}(t) - X_j(t))^2}{h\sigma_j^2} + \sum_{t_{obs}=1}^{N_{obs}} \sum_{j=1}^{N_Y} \frac{(Y_{j,pred}(t_{obs}) - Y_{j,obs}(t_{obs}))^2}{\zeta_j^2} \right) \right]$$

The double sums are for the components of X and Y. Also the h in the X component is a constant. It does not appear in the leading coefficient because it is absorbed into the proportionality.

The de-normalized log of the distribution is given

$$l(D|\theta) = - (N_{steps}) \sum_{j=1}^{N_X} \log(\sigma_j) \quad - (N_{obs}) \sum_{j=1}^{N_Y} \log(\zeta_j) - \frac{1}{2}\left(\sum_{t=1}^{N_{steps}} \sum_{j=1}^{N_X} \frac{(X_{j,pred}(t) - X_j(t))^2}{h\sigma_j^2} + \sum_{t_{obs}=1}^{N_{obs}} \sum_{j=1}^{N_Y} \frac{(Y_{j,pred}(t_{obs}) - Y_{j,obs}(t_{obs}))^2}{\zeta_j^2} \right)$$

This is the form used in experiments.

6.1.4 Distribution to sample

As before, I use the standard setup for inference problems

$$P(\theta|D) = \frac{P(\theta)\ P(D|\theta)}{P(D)}$$

The value $P(D)$ is a constant, unknown and ignored.

6.2 Results

Posterior sampling correctly finds the parameters on many problems, but convergence and accuracy are problem dependent. The numerical values of the parameters have high influence on convergence and histogram quality. The sampler is also sensitive to initial conditions. The contribution of the Brownian motion term at each step is $O(\sqrt{\Delta t})$. Depending on parameters, this is much larger than $O(\Delta t)$, which is the order of the drift term. However, the size of the drift term depends on X. If X is large, then for some time the solution is very close to exponential decay. Empirically, this helps the sampler obtain better estimates and cleaner peaks.

First I look at a simple one dimensional problem. The initial condition is numerically large to show the best example possible.

The parameters for this run are as follows. The initial condition $X(0) = 100.0$. The observation frequency is 1 and 40 observations were made. This run was performed with a chain length of 200 thousand. Simulated annealing was run with a cooling schedule of $1/20, \ldots 1/2, 1$. Each temperature of the cooling schedule is run for 50 thousand steps. The burn-in is run for 1.0 million steps. The number of walkers is 2048 and work-group size is 64. The sample rate is 10.17 M samples/second for kernel time and 1.02 M samples/second for total time. The reason

for this large disparity is that very long simulated-annealing and burn-in were run. The tuning parameter a is 1.40. The acceptance rate is 0.361.

Table 4 shows some statistics about the parameters estimated.

	true	est max likelihood	mean	68% CI	95% CI	std dev	autocorr time
a	2.000	1.986	1.988	± 1.390e-06	± 2.780e-06	0.000063	969.3
σ	1.000	1.140	1.410	± 3.408e-05	± 6.816e-05	0.001542	3830.6
b	-2.000	-1.973	-1.975	± 7.871e-07	± 1.574e-06	0.000036	1121.0
ζ	1.000	1.141	1.108	± 1.511e-05	± 3.023e-05	0.000684	2425.4

Table 4: Estimates of parameters and confidence intervals for SDE. Initial conditions far from steady state makes for accurate estimates.

Figure 15 shows correct histograms on a one dimensional problem. The ranges are selected dynamically. If a sample is far from most of the probability mass, then the plot is shown over a wide range. This is why the ranges are large and the distributions appear sharply peaked.

The histograms for σ and ζ show bias above the true value. Both variables are restricted in the prior to be nonnegative since they represent variance. The restriction causes both terms to overestimate the noise, both in the mean and estimated maximum likelihood from the histograms. This is the same mathematical phenomena seen in section 4, though the physical motivation of entropic repulsion is not relevant here.

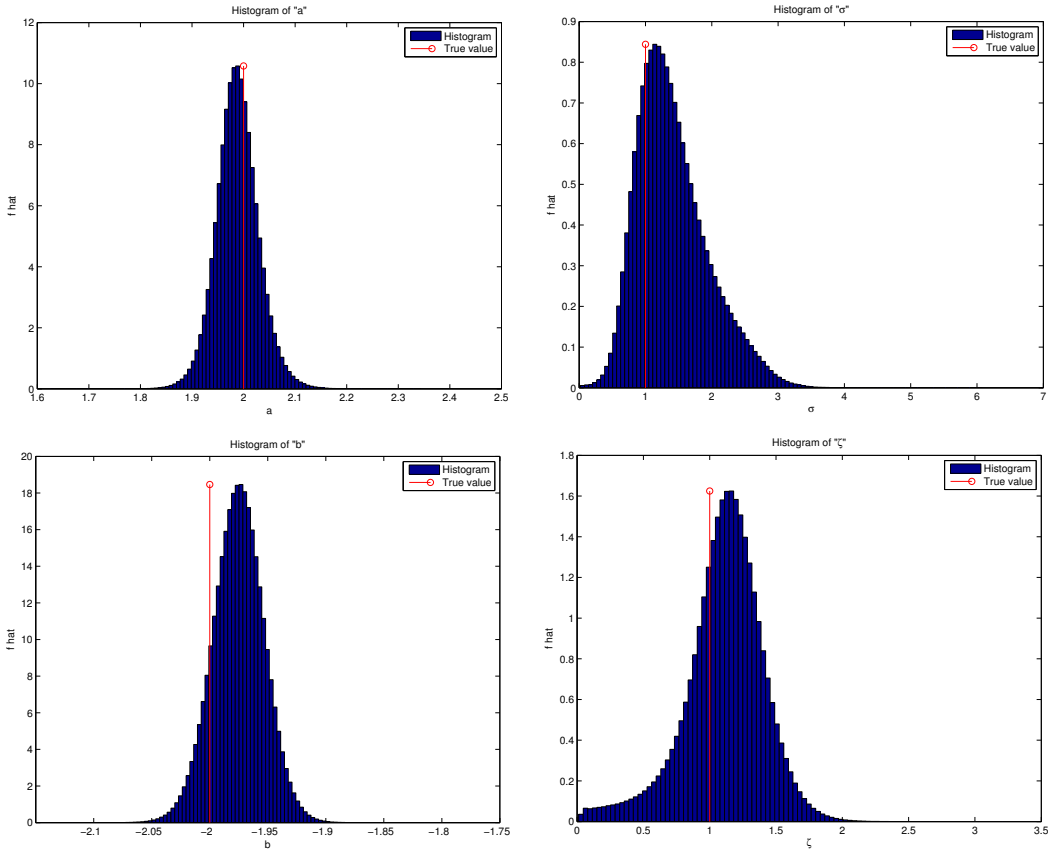

Figure 15: Posterior histograms on a one dimensional SDE

Figure 16 shows more detail around the central region. This is the same sampling as figure 15, but shows more detail around the region of maximum likelihood. Visual inspection of the histograms reveals skewness and bias. Thus, the posterior means are not exactly the true value.

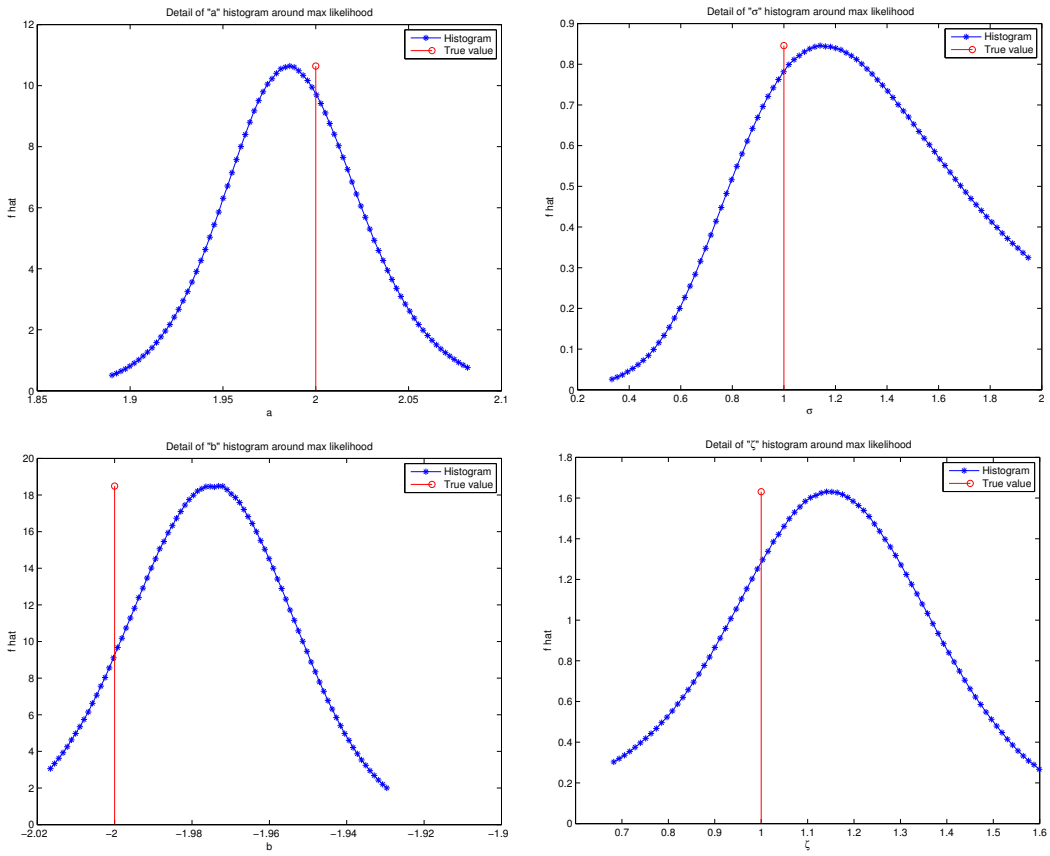

Figure 16: Details of posterior histograms on a one dimensional SDE

The second experiment shows the dependence on initial conditions. This experiment is identical to the previous one except for initial conditions, but the results are less accurate.

The parameters for this run are as follows. The initial condition $X(0) = 10.0$. The observation frequency is 1 and 40 observations were made. This run was performed with a chain length of 200 thousand. Simulated annealing was run with a cooling schedule of $1/20, \ldots 1/2, 1$. Each temperature of the cooling schedule is run for 100 thousand steps. The burn-in is run for 2.0 million steps. The number of walkers is 2048 and work-group size is 64. The sample rate is 10.13 M

samples/second for kernel time and 0.54 M samples/second for total time. The reason for this large disparity is that very long simulated-annealing and burn-in were run. The tuning parameter a is 1.40. The acceptance rate is 0.359.

Table 5 shows some statistics about the parameters estimated.

	true	est max likelihood	mean	68% CI	95% CI	std dev	autocorr time
a	2.000	1.823	1.934	± 2.876e-05	± 5.752e-05	0.001301	2935.6
σ	1.000	1.250	1.560	± 2.205e-05	± 4.411e-05	0.000998	1202.4
b	-2.000	-1.714	-1.758	± 5.141e-06	± 1.028e-05	0.000233	426.3
ζ	1.000	1.159	1.111	± 1.148e-05	± 2.296e-05	0.000520	1598.0

Table 5: Estimates of parameters and confidence intervals for SDE inference. Initial conditions near steady state for larger standard deviations.

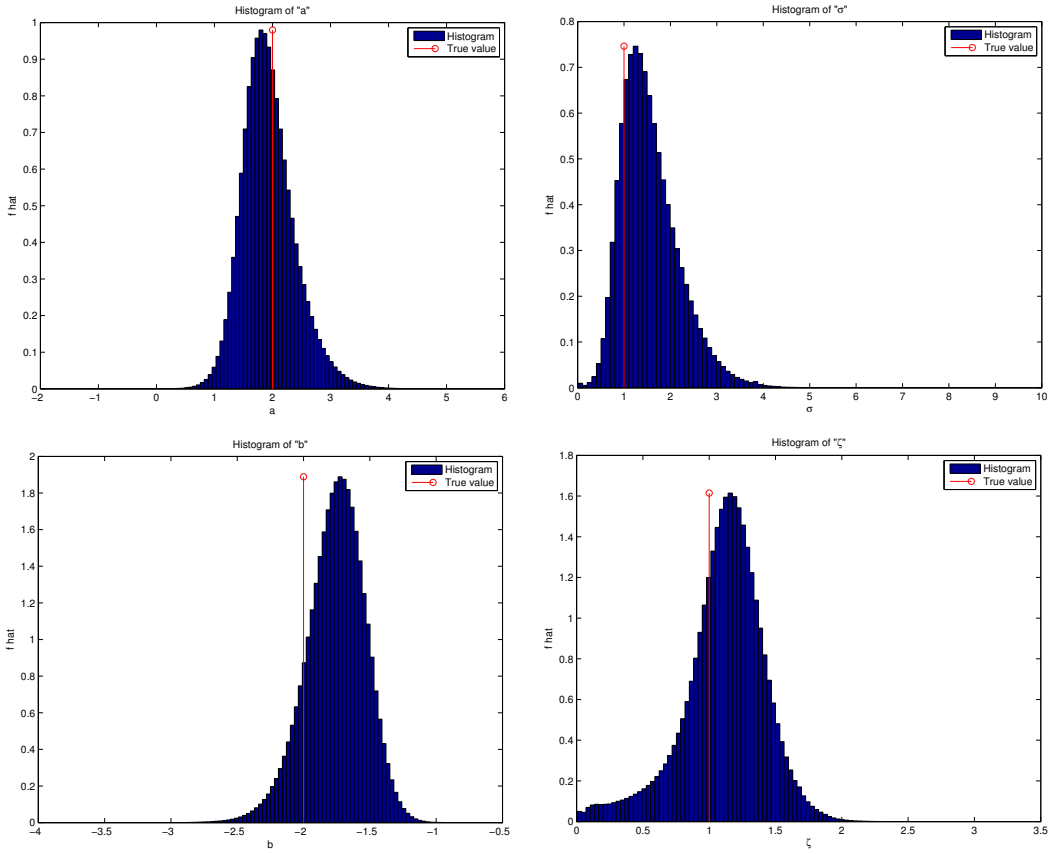

Figure 17: Less accurate parameter estimates due to changes in initial conditions

The histograms are qualitatively similar to the previous case. However, the estimates are less accurate. Also, variance of the marginal distributions are larger. For example, the width of the peak on the a histogram in figure 17 is approximately ten times larger than in figure 15.

This same problem fails entirely if the initial condition is set to one. The sampler never finds correct drift and does not converge in a reasonable amount of time.

This next experiment looks at dependence on parameters. Here, I set $a = 0$, which is standard Brownian motion. However, this is a more difficult problem than

estimating Brownian motion, since the drift term is still in the model. The drift is still of lower order if X is near zero. The issues with resolving the drift remain. This is less true if initial conditions are larger.

Here we use a large initial condition for clean results. Results for numerically smaller initial conditions are similar but less estimates get progressively worse. If initial conditions are zero, then the sampler still fails to converge.

The parameters for this run are as follows. The initial condition $X(0) = 25.0$. The observation frequency is 1 and 40 observations were made. This run was performed with a chain length of 200 thousand. Simulated annealing was run with a cooling schedule of $1/20, \ldots 1/2, 1$. Each temperature of the cooling schedule is run for 50 thousand steps. The burn-in is run for 1.0 million steps. The number of walkers is 2048 and work-group size is 32. The sample rate is 10.11 M samples/second for kernel time and 1.02 M samples/second for total time. The reason for this large disparity is that very long simulated-annealing and burn-in were run. The tuning parameter a is 1.40. The acceptance rate is 0.355.

Table 6 shows some statistics about the parameters estimated.

	true	est max likelihood	mean	68% CI	95% CI	std dev	autocorr time
a	0.000	0.011	0.015	± 1.288e-06	± 2.576e-06	0.000058	1274.0
σ	1.000	1.146	1.460	± 1.099e-05	± 2.198e-05	0.000497	359.8
b	1.000	1.023	1.025	± 1.250e-06	± 2.500e-06	0.000057	1982.1
ζ	0.500	0.572	0.534	± 8.336e-06	± 1.667e-05	0.000377	2720.9

Table 6: Estimates of parameters and confidence intervals with no drift term

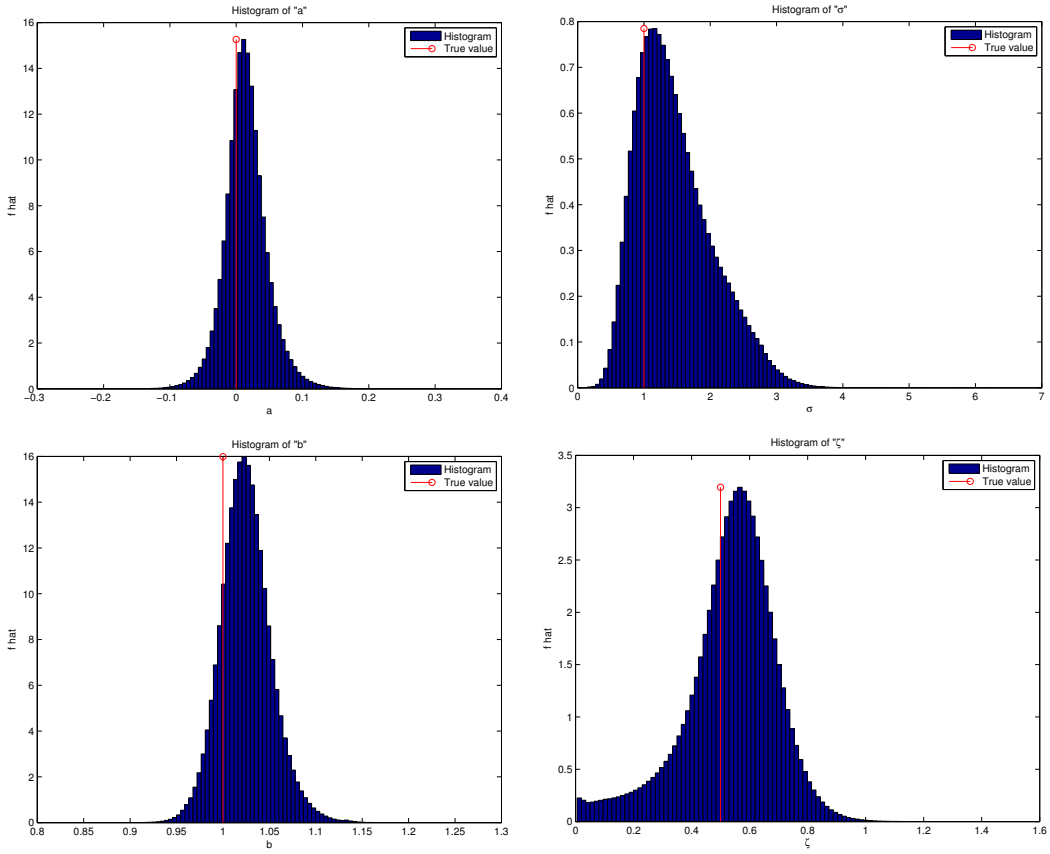

Figure 18: Estimating coefficients for $a = 0$, or Brownian motion

This plot required more manual input to get clean results. With the standard prior used in previous experiments the sampler did not perform well. The histograms had local optima which remained after very long burn-in and simulated annealing.

Better restrictions on the prior improved results. Regions that have a strong local minima in probability can be very difficult to cross. Here there is local minima is around $b = 0$, because if b was zero then there would be no observations. I changed the prior to restrict b to be nonnegative. This greatly improved the results. Even with minimal burn-in and simulated annealing, the peaks are clear

and in the correct location and extraneous local optima disappear quickly. In some situations where a coefficient is completely unknown, it might not be possible to make such restrictions. However, if analysis can be done to better restrict the coefficients, this can make a big difference.

This final experiment looks at effects of dimension scaling. Higher dimensions, regardless of the problem, generally show increased autocorrelation time. More observations should create a have lower posterior variance. To examine this, one hundred observations are made. A subset is used at each time to change the dimension of the MCMC without chaining the problem. The run length is 800 thousand. Simulated annealing is run with a cooling schedule of $1/20 \ldots 1/2, 1$ for 50 thousand steps each. Burn-in is run for one million steps. The parameter a is set to 1.4.

Figure 19 shows the scaling of autocorrelation time and variance. The autocorrelation time appears noisy. It ranges from around 1800 to 7200 with no obvious pattern. The variance on the order of .2 throughout, with no obvious pattern. The changes in variance are negligible as the problem is scaled.

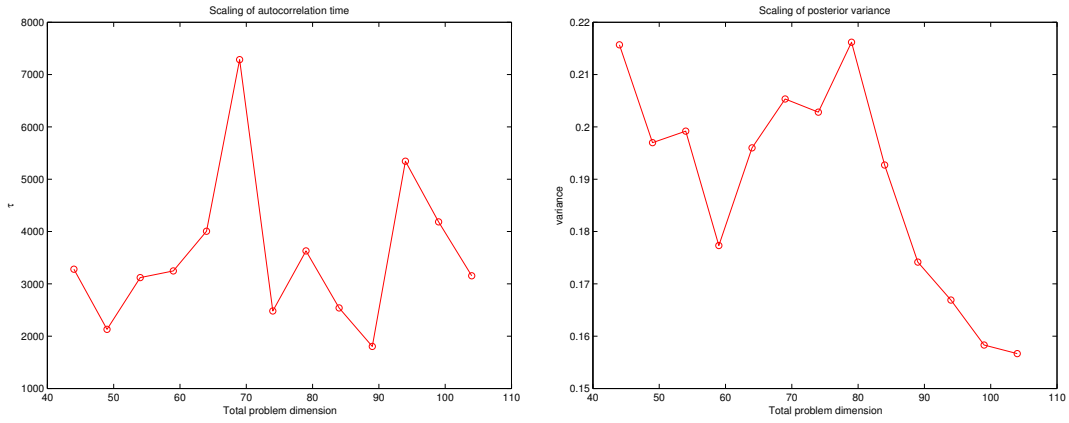

Figure 19: Scaling of autocorrelation time and posterior variance with changes in dimension

6.3 Future work

In the future, I would like to investigate more difficult and chaotic dynamical systems. I take the true dynamics, but produce data for chaotic system with first order method and see if parameters can be found. Sometimes, noise is added to the dynamics as well. One such system is the Lorenz attractor. The system depends on three parameters, σ, ρ, β, all positive.

$$\begin{aligned} dX_1 = & \quad a_1(X_2 - X_1)\, dt & + \sigma_1 dW \\ dX_2 = & \quad X_1(a_2 - X_3) - X_2\, dt & + \sigma_2 dW \\ dX_3 = & \quad X_1 X_2 - a_3 X_3\, dt & + \sigma_3 dW \end{aligned}$$

where $(a_1, a_2, a_3) = (\sigma, \rho, \beta)$. One standard set of coefficients is $\sigma = 10, \beta = 8/3, \rho = 28$.

I use the same noisy observation model as in the SDE case.

$$\begin{aligned} Y_1 &= b_1 X_1 + \zeta_1 Z \\ Y_2 &= b_2 X_2 + \zeta_2 Z \\ Y_3 &= b_3 X_3 + \zeta_3 Z \end{aligned}$$

where Z are independent $\mathcal{N}(0, 1)$.

For this problem, the sampler fails completely. This will have to remain as future work.

References

[1] Chen, J. P., and Ugurcan, B. E. Entropic repulsion of gaussian free field on high-dimensional sierpinski carpet graphs. *arXiv preprint arXiv:1307.5825* (2013).

[2] Foreman-Mackey, D., Hogg, D. W., Lang, D., and Goodman, J. emcee: The MCMC Hammer. *ArXiv e-prints* (Feb. 2012).

[3] Goodman, J., and Weare, J. Ensemble samplers with affine invariance. *Commun. Appl. Math. Comput. Sci. 5* (2010), 65–80.

[4] Green, P. J., and Mira, A. Delayed rejection in reversible jump Metropolis-Hastings. *Biometrika* (2001), 1035–1053.

[5] Hou, F., Goodman, J., Hogg, D. W., Weare, J., and Schwab, C. An affine-invariant sampler for exoplanet fitting and discovery in radial velocity data. *The Astrophysical Journal 745*, 2 (2012), 198.

[6] Kalos, M. H., and Whitlock, P. A. *Monte Carlo methods. Vol. 1: basics.* Wiley-Interscience, New York, NY, USA, 1986.

[7] Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. Equation of State Calculations by Fast Computing Machines. *Journal of Chemical Physics 21* (June 1953), 1087–1092.

[8] Nikolaisen, I. U. ranluxcl v1.3.1, 2011. https://bitbucket.org/ivarun/ranluxcl/.

[9] NVIDIA CORP. GeForce GTX 590 Specifications, 2013. http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-590/specifications.