Markov Chain Monte Carlo on Manifolds

Pranav Krishna[a], Viacheslav Karpii[a]

[a] *EPFL, Chemin du Barrage, Lausanne, Lausanne, 1015, Switzerland*

Keywords: MCMC, Manifolds, Integration

1. Introduction

The usual Markov Chain Monte Carlo algorithm as given by Metropolis-Hastings is only applicable for distributions defined over $\mathbb{R}^n$. Sampling from a manifold is useful in many applications, such as protein folding(4). In the the project proposal, we learned about an algorithm (introduced in (5)) which allows us to generate samples from an explicitly defined manifold in the style of Metropolis-Hastings, that is, without knowledge about the normalisation constant of the distribution. In this report, we provide a proof that the detailed-balance condition holds in the algorithm discussed in the paper referred to in the proposal, give two examples of distributions sampled using the algorithm, a torus and SO(11), and finally Monte-Carlo integration on the torus to find out the moment of inertia about a diameter of the torus.

2. Metropolis Hastings Algorithm

Metropolis Hastings algorithm becomes useful when it is not easy to sample directly from the target distribution, either because the distribution is too complicated due to being too high dimensional, or because of lack of knowledge about the normalising factor. The lack of knowledge about the normalising factor may be because it is too expensive to calculate it, or because it is impossible. While this setup seems contrived, it is exactly the setup of Bayesian Computation, and the MH algorithm is one of the most useful algorithms in Bayesian inference.

As mentioned in the course notes, we intend to construct a Markov Chain $\{X_n\} \sim$ Markov (λ, P) on $\mathcal{X} \subset \mathbb{R}^d$ which has a given invariant measure π with density $f : \mathcal{X} \to \mathbb{R}_+$ with respect to the Lebesgue measure. In the following discussion, we accept that the density f may be know only up to a multiplicative constant, i.e. it does not necessarily integrate to one (in which case, the invariant density is $\tilde{f}(x) = \frac{f(x)}{\int_{\mathcal{X}} f(y)dy}$).

Let $Q : \mathcal{X} \times \mathcal{B}(\mathcal{X}) \to [0, 1]$ be a Markov transition kernel with density $q : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_+$, i.e. $Q(x, A) = \int_A q(x, y)dy$ for all $x \in \mathcal{X}$ and $A \in \mathcal{B}(\mathcal{X})$, also called the proposal or instrumental density, and define the following acceptance rate $\alpha : \mathcal{X} \times \mathcal{X} \to [0, 1]$, $\alpha(x, y) = \min\left\{\frac{f(y)}{f(x)}\frac{q(y,x)}{q(x,y)}, 1\right\}$, if $q(x, y) \neq 0$, and $\alpha(x, y) = 0$, if $q(x, y) = 0$.

The Metropolis-Hasting algorithm then can be written as in Algorithm 1.

We again point out that the integration constant is not required for this algorithm as we require only the ratio of the densities in the acceptance-rejection step.

Algorithm 1: General MH algorithm

```
Given: $\lambda$ (initial measure), $q$ (proposal density), $f$ (target density)
1  Generate $X_0 \sim \lambda$
2  for $n = 0, 1, \ldots$ do
3      Generate $Y_{n+1} \sim q(X_n, \cdot)$ ;                // proposal state
4      Generate $U \sim \mathcal{U}(0,1)$
5      if $U \leq \alpha(X_n, Y_{n+1})$ then
6          set $X_{n+1} = Y_{n+1}$ ;                          // accept proposal
7      else
8          set $X_{n+1} = X_n$ ;                              // reject proposal
9      end
10 end
```

3. Metropolis Hastings Algorithm on a Manifold

Suppose we have a d-dimensional manifold $\mathcal{M}$ described by equality and inequality constrains, which is embedded on a larger d_a dimensional Euclidean space i.e., let

$$\mathcal{M} = \left\{ x \in \mathbb{R}^{d_a} \text{ such that } q_i(x) = 0, i = 1, 2, \ldots, m, \text{ and } h_j(x) > 0, j = 1, 2, \ldots, l \right\}$$

be a d-dimensional manifold embedded on an ambient space $\mathbb{R}^{d_a}$ (with $d_a > d$), subject to m equality constraints, described by m continuously differentiable functions $q_i(x), i = 1, 2 \ldots m$, and l inequality constrains, described by l functions $h_j(x), j = 1, 2, \ldots, l$. Furthermore, denote by G_x the matrix whose columns are the gradients $\{\nabla q_i(x)\}_{i=1}^{m}$, which is assumed to have full-rank m at any $x \in \mathcal{M}$ hence the manifold has a dimension $d = d_a - m$. Lastly, let $\mathcal{T}_x$ be the tangent space to $\mathcal{M}$ at $x \in \mathcal{M}$. Let us denote the target measure on the manifold by

$$\rho(\mathrm{d}x) = \frac{1}{Z} f(x) \sigma(\mathrm{d}x),$$

where, $\sigma(\mathrm{d}x)$ is the d-dimensional surface area measure, and f is a given (unnormalized) probability density function defined on the manifold. Our goal is then to sample from $\rho(\mathrm{d}x)$ and compute integrals of the form

$$I = \int_{\mathcal{M}} g(x) \sigma(\mathrm{d}x) = Z \int_{\mathcal{M}} \frac{g(x)}{f(x)} \rho(\mathrm{d}x)$$

for some σ-integrable function $g : \mathcal{M} \mapsto \mathbb{R}$. We assume here that $f(x) \neq 0$ whenever $g(x) \neq 0$. To that end, we need to generate samples $x_n \in \mathcal{M}, n = 0, 1, \ldots, N$, distributed as ρ. This can be done by the MCMC algorithm which we describe in what follows:

The **MCMC for Manifolds** algorithm iteratively repeats the following procedure:

1. Proposal: Given some state x_n, the proposal process begins with a tangential move $x_n \to x_n + v$, with $v \in \mathcal{T}_{x_n}$.

1.1 Projection: Given x_n and v, the projection step looks for some $w \perp \mathcal{T}_{x_n}$, such that $y = x_n + v + w$ satisfies all the equality constraints. It does so by finding an m-dimensional column vector a, and setting $w = \sum_{j=1}^{m} a_j \nabla q_j(x_n) = G_{x_n} a$ such that a solves

$$q_i(x_n + v + G_{x_n} a) = 0, \quad i = 1, 2, \ldots, m.$$

This can be done using any non-linear equation solver. If such a solution w can be found, we set as a proposal $y = x_n + v + w$ and advance to 1.2. Otherwise, we set $x_{n+1} = x_n$ as the new state of the chain.

1.2 Check inequality constraint: Check if any constraint is violated, that is, check if $h_i(y) \leq 0$ for some i. If so, reject y and set $x_{n+1} = x_n$. Otherwise, advance to 1.3.

1.3 Check for lack of reversibility: In order to satisfy the detailed-balance condition (i.e., reversibility of the chain), we need to verify that we can propose x_n starting from y. To that end, we need to find $v' \in \mathcal{T}_y$ and $w' \in \mathcal{T}_y^{\perp}$ such that $x_n = y + v' + w'$. Such w', v' always exist uniquely and are given by the projection of $x_n - y$ onto $\mathcal{T}_y$ and $\mathcal{T}_y^{\perp}$, respectively; they can be computed using the QR decomposition of G_y. However, one needs to verify that the non-linear solver would find x_n starting from $y + v'$. If it doesn't (in a given number of steps nmax), y is rejected and we set $x_{n+1} = x_n$. Otherwise, we continue to step 2 .

2. Acceptance-rejection step: We set $x_{n+1} = y$ with probability $\alpha(x_n, y)$, with

$$\alpha(x_n, y) = \min\left\{1, \frac{f(y)p(v' \mid y)}{f(x_n)p(v \mid x_n)}\right\},$$

otherwise, we set $x_{n+1} = x_n$ with $\alpha(x_n, y) = 0$ if $p(v \mid x_n) = 0$.

It is indeed easy to see that the chain generated using this algorithm gives rise to a Markov Chain. We show that this algorithm gives rise to a chain which has the correct stationary distribution by showing that the chain satisfies detailed balance. Since the proof for the general case is mentioned in the course notes, we just write the transition density, and the appropriate acceptance probabilities.

We note that the problem of making a tangential move is solved by finding an orthogonal matrix from the QR-decomposition of the constraint set and taking the last d columns. Then we generate a random normal variable with fixed variance and thus, we make a move in the Tangent plane by multiplying this with the orthogonal matrix.

The MCMC algorithm for manifolds then reads:

Algorithm 2: MCMC for manifolds

```
Given: Manifold $\mathcal{M}$, $x_0$ (initial point), $p$ (proposal density), $f$ (target density)
1 for $n = 1, 2, \ldots$ do
2     Generate $v \in \mathcal{T}_{x_n},\ v \sim p(\cdot|x_n)$ ;                // proposed tangential move
3     Find proposal $y$ by projecting $x + v$ on $\mathcal{M}$ ;               // proposal state
4     if projection failed or $h_j(y) \leq 0$ for some $j$ or $x$ cannot be reached from $y$ by reverse
        projection then
5         Set $x_{n+1} = x_n$ and continue
6     end
7     Set $x_{n+1} = y$ with probability $\alpha(x_n, y) = \min\left\{1, \frac{f(y)p(v'|y)}{f(x_n)p(v|x_n)}\right\}$
8 end
```

4. Main results

4.1. Selection of proposal distribution

In all our experiments we set the density for the proposed tangential move v to be Gaussian - $\mathcal{N}(0, \Sigma)$. The main question is how to choose the appropriate covariance matrix. In general we have no reason to favour any particular direction in the tangential plane, so the natural (and reasonable) choice for Σ would be the multiple of the identity matrix: $\sigma^2 I$. The parameter σ still remains to be chosen manually. Since average length of the proposed move v is proportional to σ, by setting it to be very small we might get trapped in a local neighbourhood and as a result not explore the state space in a meaningful way. Similarly, if we choose σ to be to large, we are at risk of having to many failures of projection step. In our work we experimented with different values and settled on those that led to 25-40% acceptance probability for each problem.

4.2. Generating points on the torus using MCMC

One of our tasks was to generate uniformly distributed points on a surface of a torus for $r = 0.5$ and $R = 1$. The implicit definition goes like this:

$$\mathbb{T} := \left\{(x, y, z) \in \mathbb{R}^3 | (R - \sqrt{x^2 + y^2}) + z^2 - r^2 = 0\right\} \tag{4.1}$$

And the explicit parametrization:

$$\mathbb{T} := \left\{\Big((R + r\cos(\phi))\cos(\theta), (R + r\cos(\phi))\sin(\theta), r\sin(\phi)\Big)\ \phi, \theta \in [0, 2\pi]\right\} \tag{4.2}$$

We implemented the MCMC algorithm for manifolds as described in Section 3 and generated a chain of length 10^6 of points on torus. We started from a point $(1, 0, 0.5)$ and chose $\sigma = 0.5$. This yielded the acceptance probability to be around 33%. Figure 1 shows every 100^{th} point of

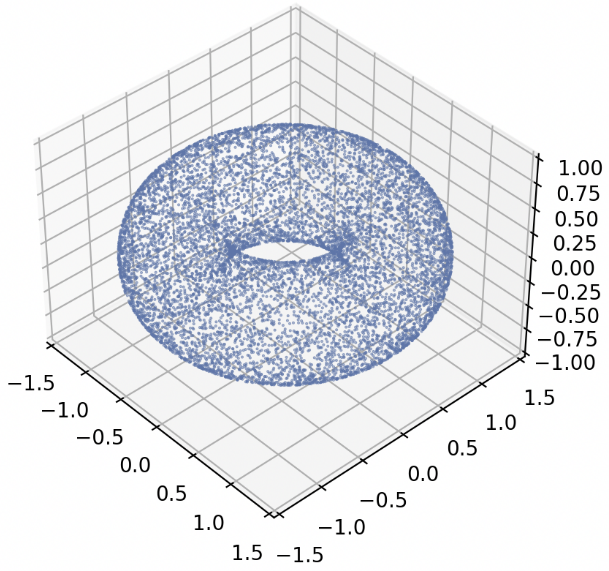

Figure 1: Points on a torus generated by the MCMC algorithm

that chain and as we can see the distribution indeed looks visually uniform. Another assessment of the correctness of the distribution are the marginal densities of torus angles ϕ and θ. Angle θ is the angle around the torus from the origin and is uniform on $[0, 2\pi]$. And ϕ has the density proportional to $R + r\cos(\phi)$ - the distance from the origin to the projection of the point onto the xy plane. As can be seen from (1) density of ϕ takes the form $\frac{R+r\cos(\phi)}{2\pi R}$ on $[0, 2\pi]$. Figure 2 shows the empirical histogram of the angles extracted from the points in the chain generated by MCMC and the theoretical distributions. Additionally, Kolmogorov-Smirnov test on the decorelated chain (taking every 10^{th} entry) fails to reject the Null hypothesis that empirical θ, ϕ are distributed according to their theoretical distributions. at significance level $\alpha = 0.05$. We have to decorrelate the chain because Kolmogorov-Smirnov test requires data to be independent.

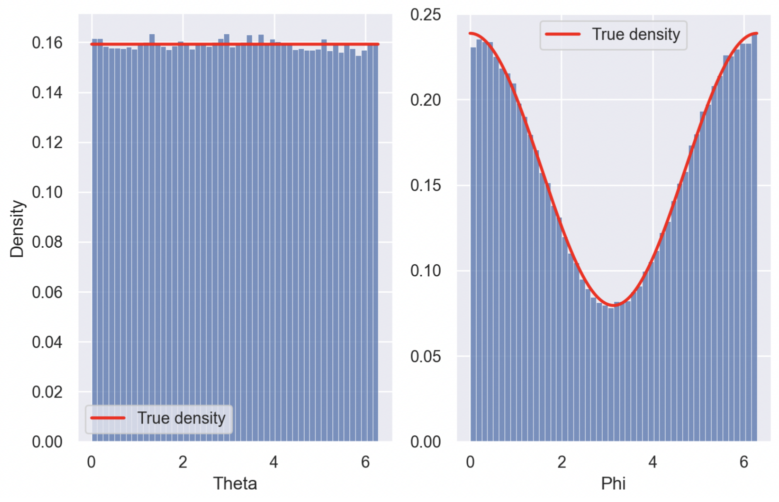

Figure 2: Empirical distributions of torus angles of the points from MCMC chain

4.3. Alternative way to sample from a torus

Using the explicit parameterization (4.2) and the theoretical densities of θ and ϕ we can come up with another way to sample points from the surface of a torus. To sample from the distribution of ϕ we will use the Acceptance-Rejection algorithm: chose uniform distribution on $[0, 2\pi]$ as auxiliary with constant $R+r$. All the steps needed to generate a point on a torus are summarized as follows:

Algorithm 3: AR algoritm to sample from a torus

```
1 Generate $\Theta, \Phi \sim \mathcal{U}(0, 2\pi)$;
2 Generate $U \sim \mathcal{U}(0, 1)$;
3 if $U \leq \frac{R+r\cos(\Phi)}{R+r}$ then
4     $X = ((R + r\cos(\phi))\cos(\theta), (R + r\cos(\phi))\sin(\theta), r\sin(\phi))$;
5 end
```

This method has a constant acceptance probability of $\frac{R}{R+r} = \frac{2}{3}$ and is much more efficient since it doesn't require as many computations at every iteration as compared to MCMC.

4.4. Generating elements of SO(d) using MCMC

SO(d) can be viewed as a manifold embedded in d^2-dimensional space and satisfying $\frac{1}{2}d(d+1)$ equality constrains:

$$q_{kk}(x) = \sum_{m=1}^{d} x_{km}^2 = 1, \qquad q_{kl}(x) = \sum_{m=1}^{d} x_{km}x_{lm} = 0 \quad \forall k = 1 \dots d,\ l > k$$

Therefore, we can use Algorithm 2 in order to sample from SO(d). We generated a chain of length 10^6 from SO(11). We used identity matrix as a starting point and set $\sigma = 0.28$ which resulted in acceptance probability of around 35%. In order to somehow verify the correctness of obtained

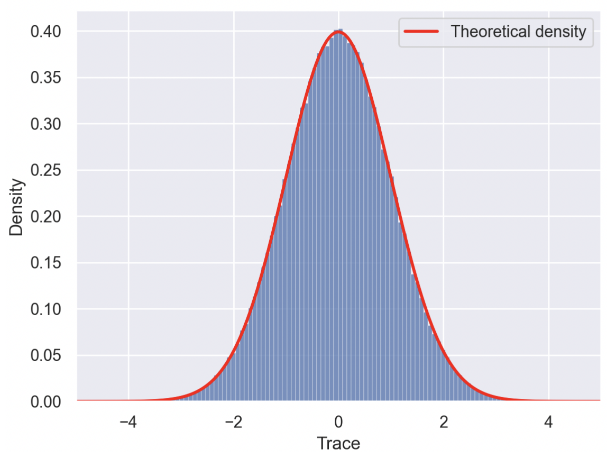

Figure 3: Empirical distributions of the trace of SO(11) elements from MCMC chain

distribution we used the fact that $Tr(x) \sim \mathcal{N}(0,1)$ as $d \to \infty$. The distribution of the traces of our chain is shown on Figure 3.

Another intuitive way to asses the quality of obtained distribution would be generating samples from SO(2) since it contains the rotation matrices with rotation angles being uniformly distributed on $[-\pi, \pi]$. We generated a Markov chain of length 10^6 of SO(2) elements and extracted the rotation angle from each of them. After that we plotted the empirical distribution as is shown in Figure 4. The Kolmogorov-Smirnov test on the decorrelated chain (taking every 10^{th} entry) fails to reject

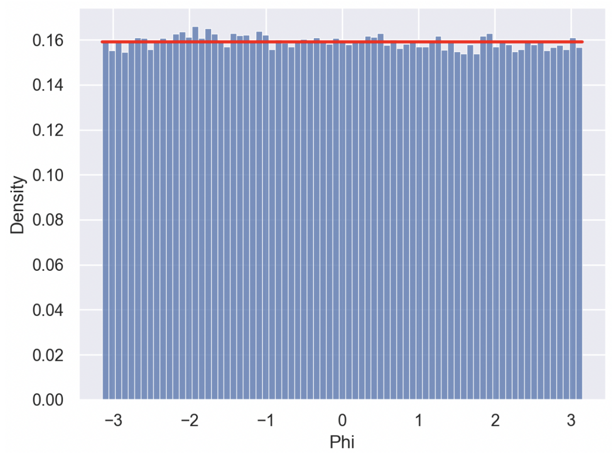

Figure 4: Empirical distributions of rotation angles of SO(2) Markov chain

the hypothesis that empirical ϕ is uniformly distributed on $[-\pi, \pi]$ at significance level $\alpha = 0.05$.

4.5. Alternative way to sample from SO(d)

Another way to generate from SO(d) would be to first generate a $d \times d$ matrix X whose entries are i.i.d. $\mathcal{N}(0,1)$ and then do a QR-decomposition $X = QR$ where Q is orthogonal and R is upper triangular. And if $\det(Q) = 1$ we can take Q as our sample. However, this approach requires some tweaks to work properly. For instance, QR-decomposition is not unique and depends on the algorithm used to implement it. That is important because we need the decomposition to be carried out by Gram-Schmidt Orthogonalisation. In other words, we need to make sure that the matrix

R has positive diagonal entries. As shown in (3), this approach produces the correct distribution. The entire procedure is described in the algorithm below:

Algorithm 4: Simple way to sample from SO(d)

```
1 Generate X $\in \mathbb{R}^{d\times d}$ where $x_{ij} \sim \mathcal{N}(0,1)$ and are independent
2 Find Q and R s.t. $X = QR$   where Q is orthogonal, R is upper triangular
3 Take $\Lambda$ to be the diagonal matrix with $d_{ii} = \frac{r_{ii}}{|r_{ii}|}$
4 Set $Y = Q\Lambda$ ;                                        // proposed sample
5 if det(Y) = 1 then
6     accept Y
7 end
```

This algorithm might produce matrices with determinant 1 or -1 with equal chance, so it has a constant 50% acceptance probability. On top of that, it is much faster than MCMC algorithm 2 because it does not require to perform an expensive projection operation at every iteration.

Arguably, we can improve Algorithm 4: Whenever it proposes a matrix with determinant -1 we can swap the first two rows. This operation doesn't affect orthonormality but changes the sign of the determinant. Intuitively by symmetry of the uniform distribution the resulting distribution still has to be uniform. If we generate one sample with and one without swapping of the rows, we indeed observed that distributions of the traces are the same according to the Kolmogorov-Smirnov test.

5. Application

5.1. Theoretical framework

As was mentioned earlier the main application for Algorithm 2 is the computation of integrals on manifolds. In this section we tried to estimate the moment of inertia I of a torus rotating about one of its diameters, say the x axis for a torus with the axis of symmetry as z axis. The analytical formula gives us the true value for comparisons (we assume mass to be $4\pi^2 Rr$):

$$I_{\text{True}} = \frac{1}{4}m(5r^2 + 2R^2) \approx 16.04$$

Let's denote the Markov chain generated by Algorithm 2 as $\{X_n = (X_{n,1}, X_{n,2}, X_{n,3})\}_{n=1}^N$. Then since our target function is the distance to the x axis, the estimator for I is:

$$I_N^{\text{MCMC}} = \frac{4\pi^2 Rr}{N}\sum_{n=1}^{N}(X_{n,2}^2 + X_{n,3}^2)$$

From the course know that the variance of I_N^{MCMC} is given by $\mathbb{V}ar[I_N^{\text{MCMC}}] = \frac{\sigma^2_{\text{MCMC, N}}}{N}$. And to get an estimate on $\sigma^2_{\text{MCMC, N}}$ we can use the batch means method by splitting $\{X_n\}_{n=1}^N$ into M blocks with size $T = N/M$. Then we have the following estimates:

$$I^{(i)} = \frac{1}{T}\sum_{j=(i-1)T+1}^{iT} X_{j,2}^2 + X_{j,3}^2, \quad I_N^{MCMC} = \frac{1}{M}\sum_{i=1}^{M} I^{(i)}$$

$$\hat{\sigma}^2_{\text{MCMC, N}} = \frac{N}{M(M-1)} \sum_{i=1}^{M} (I^{(i)} - I_N^{MCMC})^2$$

This method was chosen among others mainly because it's easy and fast to compute. Now in order to estimate sample size N needed to achieve a standard deviation less than some prescribed tolerance we made use of the sequential algorithm:

Algorithm 5: Sequential algorithm

```
  Given: $N_{\min}$, $N_{\max}$, k (step size), tol
1 Generate a chain $\{X_n\}_{n=1}^N$ with $N = N_{\min}$
2 while $\sqrt{\mathbb{V}ar[I_N^{MCMC}]} > tol$ and $N < N_{\max}$ do
3     Generate new k sapmles of the chain starting from $X_N$
4     Set $N = N + k$
5     Re estimate $\hat{\sigma}^2_{\text{MCMC, N}}$ and $\mathbb{V}ar[I_N^{\text{MCMC}}]$
6 end
```

We also wanted to see how I_N^{MCMC} compares to estimator obtained using a sample generated by Algorithm 3. We denote this sample as $\{Y_n = (Y_{n,1}, Y_{n,2}, Y_{n,3})\}_{n=1}^N$ and we will refer to its estimator of moment of inertia as CMC (crude Monte-Carlo):

$$I_N^{\text{CMC}} = \frac{4\pi^2 Rr}{N} \sum_{n=1}^{N} Y_{n,2}^2 + Y_{n,3}^2$$

For its variance $\mathbb{V}ar[I_N^{\text{CMC}}] = \frac{\sigma^2_{\text{CMC, N}}}{N}$ we can use a standard unbiased estimate of $\sigma^2_{\text{CMC, N}}$:

$$\hat{\sigma}^2_{\text{CMC, N}} = \frac{1}{N-1} \sum_{n=1}^{N} (Y_{n,2}^2 + Y_{n,3}^2 - I_N^{\text{CMC}})^2$$

5.2. Numerical results

In this section we present the results of numerical comparisons between I_N^{MCMC} and I_N^{CMC}. Figure 5 shows how our estimators evolve with bigger sample sizes. And indeed both of them seem to converge to the true value.

In Figure 6 we can observe the relative errors of both estimators in logarithmic scale. MCMC estimator has smaller fluctuations but both errors decay at seemingly the same rate.

Comparing the variances of the two estimators (Figure 7) we can see that MCMC has bigger variance. Finally, we can estimate sample sizes needed by each estimator to achieve a root mean squared error smaller than a prescribed tolerance. For instance setting tolerance to be 0.2 we discover that MCMC estimator needs around $N_{\text{MCMC}} = 29000$ samples while CMC only requires $N_{\text{CMC}} = 2400$. Thus making the CMC estimator more efficient. And if we remember that CMC is also much faster we can come to the conclusion that in case of our simple problem setting there is no need to go through all the trouble to use MCMC.

6. Conclusion

During this project we got familiar with the Metropolis-Hastings styled algorithm for sampling from arbitrary manifolds. In our opinion its strength is it's generality: we need to know only

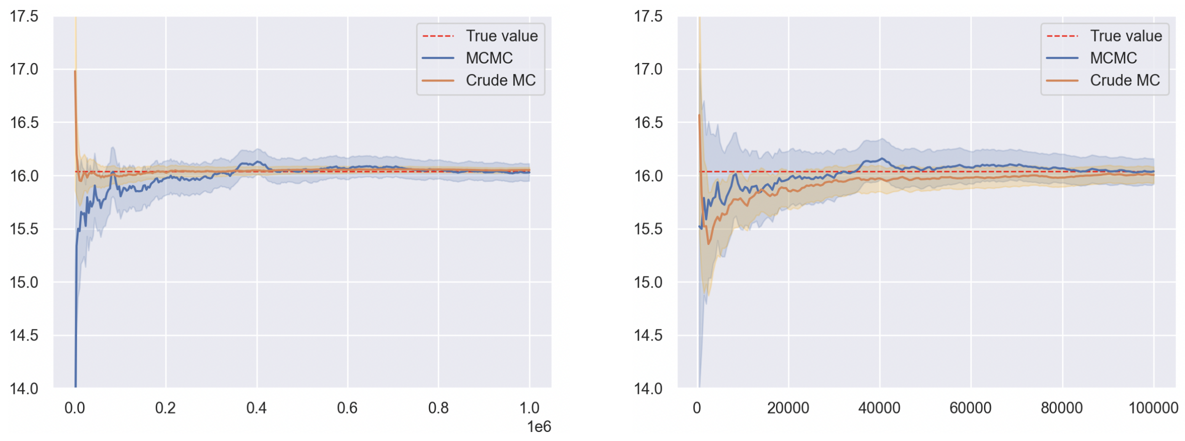

Figure 5: Evolution of MCMC and CMC estimates of moment of inertia. On the left - using the entire chain of 10^6 sapmles; on the right - every 10th sample. Translucent regions represent asymptotic 95% confidence intervals.

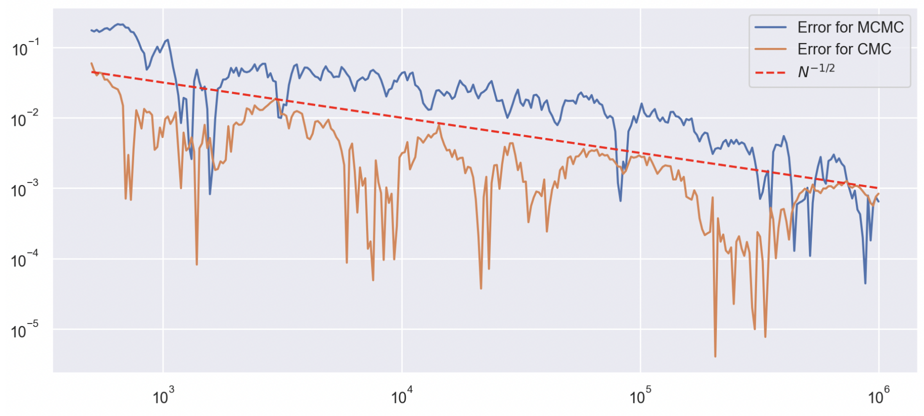

Figure 6: Relative error of MCMC and CMC estimators in log-log scale.

the equality and inequality constrains of a manifold. However, in some sense this is also the algorithm's main weakness: in order to achieve this kind of generalization we have to rely on a very computationally expensive operation of projecting onto the manifold. During code profiling of our implementation we discovered that over 90% of the total run time is spent on projection. This severely limits how useful this algorithm is especially in high dimensions because to our knowledge there is no general way to project efficiently. In our simulations as soon as we had more information about our manifold which we could exploit (e. g. distribution of torus angles or uniformity of distribution obtained from QR), we were able to create faster and more robust methods for sampling.

Overall, we would say that the best use cases for this algorithm are scenarios when we either don't know anything about our manifold at all due to its high complexity or when we are able to come up with efficient way to project onto it.

7. Proof of Detailed Balance

Suppose we are at $X_n = x$. We generate a tangential move v from a proposal density p,i.e.$v \sim p(v|x)$. Suppose the deterministic function that finds the projection is called g, such that $g(x+v) =$

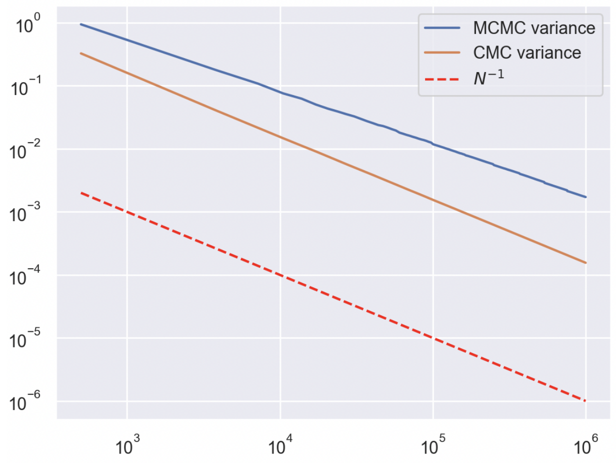

Figure 7: Variances of MCMC and CMC estrimator in log-log scale.

y, where y is a point on the manifold. As we move only in the subspace orthogonal to $\mathcal{T}_x$, we can express the projection as $y = x + v + w$ where $w \perp \mathcal{T}_x$ and $v \in \mathcal{T}_x$.

Because we are doing the projection using Newton's method which may not converge in some cases, we define another function $\hat{g}$, the deterministic function that finds, or fails to find the projection such that $g(x + v) = y$ is either a point on the manifold when the projection succeeds, or the point x when it fails. Then, the proposal density is $\hat{g}(x + v)$, a mixed random variable with a point mass at $x \in R^{d_a}$. Let us express the set of points on the manifold that can not be reached by the Newton solver implemented in the algorithm as $\mathcal{S}_x$. The density of the proposal can be expressed as

$$\hat{q}(x, y) = \mathbf{1}\left\{y \in \mathcal{S}_x^{\mathcal{C}}\right\} \cdot p(v|x) \cdot J_{x+v}(y) + \zeta(x)\delta_x(y) \tag{7.1}$$

where $\zeta(x)$ is the probability that the projection fails and $\delta_x(\cdot)$ is the unit point mass distribution at x and $y = x + v + w$. Here, $J_{y+v'}(x)$ denotes the Jacobian of the function g on the set $\mathcal{S}_x^{\mathcal{C}}$ and zero otherwise. Similarly, $J_{x+v}(y)$ denotes the the Jacobian of the function g on the set $\mathcal{S}_y^{\mathcal{C}}$ and is 0 otherwise. Observe that we have, by definition that $\zeta(x) + \int_{\mathcal{S}_x^{\mathcal{C}}} p(v|x) \cdot J_{x+v}(y) \cdot \sigma(dy) = 1$

Now, for the expression of the acceptance probability, we need to calculate the probability of making the reverse step, which turns out to be

$$\hat{q}(y, x) = \mathbf{1}\left\{x \in \mathcal{S}_y^{\mathcal{C}}\right\} \cdot p(v'|y) \cdot J_{y+v'}(x) + \zeta(y)\delta_y(x) \tag{7.2}$$

where $\zeta(y)$ is the probability that the projection fails and $\delta_y(\cdot)$ is the unit point mass distribution at y and $x = y + v' + w'$. Geometrically, we observe that the v' and w' are unique.

This would lead one to conclude that the acceptance probability should be

$$\alpha(x, y) = \min\left\{1, \frac{f(y)J_{y+v'}(x)p(v'|y)}{f(x_n)J_{x+v}(y)p(v|x)}\right\} \tag{7.3}$$

with $\alpha(x, y) = 0$ if $J_{x+v}(y) = 0$.

But, if we restrict to the set $\mathcal{S}_x^{\mathcal{C}} \cap \mathcal{S}_y^{\mathcal{C}}$ we observe that the Jacobian factors cancel out.

Suppose U_x and U_y are matrices whose columns are orthonormal bases for $\mathcal{T}_x$ and $\mathcal{T}_y$, respectively. We denote the i^{th} column of U_x by $u_{x,i}$, and similarly for U_y. For any vector $\kappa \in \mathcal{T}_x$. We can express it as a linear combination of the columns of U_x as $\kappa = \Sigma a_i u_{x,i}$ in Einstein Summation notation. If $\hat{\kappa}$ is the projection of this vector in $\mathcal{T}_y$ perpendicular to $\mathcal{T}_x$, we can express $\hat{\kappa}$ as

$\hat{\kappa} = \Sigma b_i u_{y,i}$. By orthogonality, we have that $u_{x,i}^\top \hat{\kappa} = a_i$ for each i. Thus, we have the following set of equations

$$a_i = \Sigma u_{x,i}^\top u_{y,j} b_j. \tag{7.4}$$

Writing this is Matrix notation, we have that $U_x^\top U_y b = a$ which is equivalent to saying $b = (U_x^\top U_y)^{-1} a$. Thus, the volume element is magnified in magnitude by

$$J_{x+v}(y)^{-1} = \det((U_x^\top U_y)^{-1}) \tag{7.5}$$

Similarly for y, we get that $J_{y+v'}(x)^{-1} = \det((U_y^\top U_x)^{-1})$, Thus, the determinants of the Jacobian factors are equal and cancel out giving us that the acceptance probability is

$$\alpha(x, y) = \min \left\{ 1, \frac{f(y) p(v'|y) \mathbf{1}\{x \in \mathcal{S}_y^{\mathcal{C}}\}}{f(x) p(v|x) \mathbf{1}\{y \in \mathcal{S}_x^{\mathcal{C}}\}} \right\} \tag{7.6}$$

where $\alpha(x, y) = 0$ if $\mathbf{1}\{y \in \mathcal{S}_x^{\mathcal{C}}\} = 0$. Thus, we have that the overall probability of rejecting a proposal is $\xi(x)$, it is the sum of sum of the probability of the failure of projection and rejecting the proposal that was generated, i.e.,

$$\xi(x) = \zeta(x) + \int_{\mathcal{S}_x^{\mathcal{C}}} (1 - \alpha(x, y)) p(v'|y) \cdot J_{y+v'}(y) \sigma(dy) \tag{7.7}$$

Thus, we have the transition density

$$q(x, y) = \alpha(x, y) \mathbf{1} \left\{ y \in \mathcal{S}_x^{\mathcal{C}} \right\} \cdot p(v|x) \cdot J_{x+v}(y) + \xi(x) \delta_x(y). \tag{7.8}$$

Equivalently, the transition kernel P is given by

$$P(x, A) = \int_A \mathbf{1} \left\{ y \in \mathcal{S}_x^{\mathcal{C}} \right\} \alpha(x, y) p(v'|y) \cdot J_{y+v'}(y) \sigma(dy) + \xi(x) \mathbf{1} \{x \in A\}$$

We recall that the crucial steps in the proof of detailed-balance in Lemma 8.11 in the Lecture notes (2) are the interchange of integrals using Fubini-Tonelli Theorem and the fact that

$$\begin{aligned} f(x) q(x, y) \alpha(x, y) &= f(x) q(x, y) \min \left\{ \frac{f(y)}{f(x)} \frac{q(y, x)}{q(x, y)}, 1 \right\} \\ &= \min\{f(y) q(y, x), f(x) q(x, y)\} \\ &= f(y) q(y, x) \alpha(y, x) \end{aligned} \tag{7.9}$$

where f is the target density, q is the transition density and α is the acceptance probability. Furthermore, observe that equation 7 is of the same functional form as the equation in the lecture notes.

We see that the interchange of integrals using the theorem is still allowed as the quantities in this case are still non-negative. Thus, we need to show that equation 7.9 holds for the transition density in this case.

While it is easy to see that 7.9 is true in the case when we can reach x from y and vice-versa as the Jacobian factors cancel out, it the detailed balance equation also holds when we cannot propose y starting from x. To see this, observe that if $\mathbf{1}\{y \in \mathcal{S}_x\}$, the left hand side of the detailed balance is zero as if $\alpha(x, y) = 0$ if $J_{x+v}(y) = 0$. On the right hand side, clearly, the indicator in the numerator would be zero. Hence, we see that the detailed balance relation holds.

References

[1] Persi Diaconis, Susan Holmes, and Mehrdad Shahshahani. Sampling from a manifold. In *Advances in modern statistical theory and applications. A Festschrift in honor of Morris L. Eaton*, pages 102–125. Beachwood, OH: IMS, Institute of Mathematical Statistics, 2013.

[2] Lecture Notes. Stochastic simulation.

[3] Francesco Mezzadri. How to generate random matrices from the classical compact groups. 2006.

[4] Yong Wang, Pengfei Tian, Wouter Boomsma, and Kresten Lindorff-Larsen. Monte carlo sampling of protein folding by combining an all-atom physics-based model with a native state bias. *The Journal of Physical Chemistry B*, 122(49):11174–11185, 2018.

[5] Emilio Zappa, Miranda Holmes-Cerfon, and Jonathan Goodman. Monte carlo on manifolds: Sampling densities and integrating functions. *Communications on Pure and Applied Mathematics*, 71(12):2609–2647, 2018.

8. Appendix

Here you can find an extract from our code. In particular there are two main functions: projection onto the manifold and the implementation of the MCMC algorithm itself. The rest of the code we used is attached with the submission.

```python
import line_profiler
import numpy as np
from tqdm import tqdm
from scipy.linalg import inv

# we used it to asses the performance of our code
profile = line_profiler.LineProfiler()


@profile
# function to perform projection using Newton's method
# solve q(z + Qa) for a
def project(q, z, Q, grad, dim, nmax=0, tol=0.0001):
    a = np.zeros(dim)
    i = 0
    # for the sake of performance the default max iterations count is set to be low
    if nmax == 0:
        nmax = min(3 * dim, 40)
    stop = False
    while not stop:
        arg = z + Q @ a
        q_arg = q(arg)
        try:
            # solve grad(z + Qa)^T * Q * da = - q(z+ Qa) for da
            da = -inv(grad(arg).T @ Q) @ q_arg
        except:
            # in case numerical issues arise we catch the exceptions and use pseudo
    inverse
```

```
            da = -np.linalg.pinv(grad(arg).T @ Q) @ q_arg
        a = a + da
        i += 1
        stop = np.linalg.norm(q_arg) < tol
        if i > nmax:
            return a, False

    # return the solution and the status of projection
    return a, True


@profile
# main function to perform MCMC on an arbitrary manifold
# N - desired length of the chain
# da - ambient dimensionality
# m - number of constrains
# q - function that returns an evaluation of equality constraints (q_1(x), .... q_m(
    x))
# grad - function that has gradient of q_i as its ith column
# x0 - starting point on a manifold
# sigma - scaling factor for tangential proposal moves
# ineq_constraints - ruturns False if the point doesn't satisfy all the inequality
    constraints
# show_pbar - set to False if you don't want to see the progress bar
def mcmc_manifold(N, da, m, grad, q, x0, sigma, ineq_constraints=None, show_pbar=
    True):
    d = da - m
    X = np.zeros((N + 1, da))
    X[0] = x0
    accepted = 0
    if show_pbar:
        pbar = tqdm(range(N), desc="Elements of Markov chain generated")
    else:
        pbar = range(N)
    for i in pbar:
        X[i + 1] = X[i]
        Gx = grad(X[i])

        # finding the basis of the tangential plane
        qrx = np.linalg.qr(Gx, mode='complete')[0][:, m:]

        t = np.random.normal(size=d) * sigma
        if not isinstance(t, np.ndarray):
            t = [t]

        #  making a proposal move in the tangential plane
        v = qrx @ t

        # try projecting x + v onto the manifold
        a, flag = project(q, X[i] + v, Gx, grad, m)
        if not flag:
            continue

        w = Gx @ a
        # creating the proposal point on the manifold
        Y = X[i] + v + w

        # check if it violates inequality constraints
        if ineq_constraints is not None and not ineq_constraints(Y):
            continue

        # creating a reverse move
        Gy = grad(Y)
        qry = np.linalg.qr(Gy, mode='complete')[0][:, m:]
        v_ = qry @ qry.T @ (X[i] - Y)

        # acceptance probability
        alpha = min(1, np.exp(-(np.linalg.norm(v_) ** 2 - np.linalg.norm(v) ** 2) / 2 / sigma ** 2))
        U = np.random.uniform()
        if U > alpha:
            continue

        # check if we can do the reverse step
        reversebility_check, flag = project(q, Y + v_, Gy, grad, m)
        if not flag or np.linalg.norm(X[i] - Y - v_ - Gy @ reversebility_check) > 0.1:
            continue

        # if all the checks pass we can accept
        X[i + 1] = Y
        accepted += 1

    if show_pbar:
        pbar.close()
        print("Acceptance probability: " + "{0:.2%}".format(accepted / N) + '\n')

    # returns the chain and the acceptance probability
    return X, accepted / N
```

mcmc_manifolds.py