Basis Expansion Monte Carlo

Eric Kernfeld *

University of Washington, Seattle, WA, USA

April 18, 2015

Abstract

We introduce Basis Expansion Monte Carlo, which studies a Gibbs or Metropolis-Hastings sampler to infer the underlying transition kernel. To make inference about the steady state, which is usually the item of interest in a sampler, we compute the steady-state of the approximate kernel. Results show ...

1 Introduction

In many statistical models, it is impossible to find a closed form for the distribution of interest (we will call this π). One work-around, originating in computational physics, relies on the fact that for points x_1 and x_2 in the parameter space, $\pi(x_1)/\pi(x_2)$ may still be calculable, though $\pi(x_1)$ and $\pi(x_2)$ are not. This fact is exploited to produce a Markov chain whose steady-state distribution is guaranteed to be π.

More and references about history, background, and/or tutorials on monte carlo methods

One popular method, the Metropolis-Hastings scheme, consists of the following procedure.

Algorithm 1: Metropolis-Hastings algorithm

Set $x_0 = 0, i = 0$
Repeat ad nauseum:
  Increment i
  Draw x from a proposal distribution $q(x|x_{i-1})$
  Set $\alpha(x|x_{i-1}) = \min(1, \frac{\pi(x)q(x_{i-1}|x)}{\pi(x_{i-1})q(x|x_{i-1})})$
  Set $x_i = x$ with probability α and $x_i = x_{i-1}$ as the complementary event.

Suppose this MCMC algorithm produces a chain $x_1, x_2, x_3, ...$ of samples. Because the algorithm is stochastic, these samples can be viewed as realizations of random variables $X_1, X_2, X_3, ...$ with marginal density functions f_1, f_2, f_3, etc. Because X_i is independent of past draws given X_{i-1}, the conditional distribution $f_{i|i-1}(x_i|x_{i-1})$ contains all the information we need about this method. In particular, to move forward an iteration, we can write $f_i(x_i) = \int f_{i|i-1}(x_i, x_{i-1})f_{i-1}(x_{i-1})dx_{i-1}$. Noting that $f_{i|i-1}$ doesn't depend on i, we can replace it with a function K so that $f_i(x_i) = \int K(x_i, x_{i-1})f_{i-1}(x_{i-1})dx_{i-1}$. This function K, called the Markov kernel, is analogous to the transition probability matrices of discrete-space Markov chain theory. We refer to the linear operator of integrating against K as L, so that $f_i = Lf_{i-1}$. The object of interest is the steady state of this operator, an eigenfunction π that has eigenvalue 1 so that for any x, $\pi(x) = (L\pi)(x) = \int K(x,t)\pi(t)dt$.

In MCMC methods, chains are usually left to run until the Markov chain converges to its stationary distribution. By contrast, in BEMC, we approximate L by methods that allow the chain to run briefly from many different places, and we compute π from the approximation.

*Electronic address: `ekernf01@u.washington.edu`; Corresponding author

1.1 Stage one: approximating the kernel

1.1.1 Notation and setup

Our estimator begins with a fixed set of basis functions $\{h_i\}_1^B$. We will have two sources of error: discretization from the finite basis and stochasticity from the sample. Object suffering from discretization error will be labeled with tildes, while objects also suffering from stochasticity will be labeled with hats. There is one exception: we estimate a matrix called M, with both error sources present, and it will appear with no hat.

For an unknown transition kernel K, define $\tilde{M}$ to be the matrix that minimizes the squared $\mathcal{L}_2$ distance $\|K - \sum_{i,j}^B \tilde{m}_{ij} h_i \otimes h_j\|^2$, or in other terms the matrix that minimizes the integral

$$\int_{\Omega\times\Omega} (K(x,y) - \sum_{i,j}^B \tilde{m}_{ij} h_i(x) h_j(y))^2 dxdy.$$

EMK: Surely this minimum must exist for a quadratic? Then define $\tilde{K}(x,y) \equiv \sum_{i,j=1}^B \tilde{m}_{ij} h_i(x) h_j(y)$ and define $\tilde{L}$ so that $(\tilde{L}f)(x)$ is

$$\int_\Omega \tilde{K}(x,y) f(y) dy.$$

We will attempt to estimate $\tilde{M}$ using a matrix M, and the corresponding approximation for K will be

$$\hat{K}(x,y) = \sum_{i,j=1}^B M_{ij} h_i(x) h_j(y). \tag{1}$$

Similar to before, $\hat{L}$ is defined so that $(\hat{L}f)(x)$ is

$$\int_\Omega \hat{K}(x,y) f(y) dy.$$

We define a matrix $\tilde{G}$ elementwise so that $\tilde{g}_{ij} \equiv \int_\Omega h_i(x)(\tilde{L}h_j)(x)dx$, with the corresponding statement for hatted variables so that $\hat{g}_{ij} \equiv \int_\Omega h_i(x)(\hat{L}h_j)(x)dx$. We also make use of the $\mathcal{L}_2$ inner products $c_{ij} \equiv \int_\Omega h_i(x) h_j(x) dx$.

1.2 The BEMC estimator

Since $\hat{g}_{ij}$ expands as

$$\int_\Omega h_i(x) \left[\sum_{\ell,k} M_{k\ell} h_k(x) \int_\Omega h_\ell(y) h_j(y) dy \right] dx = \sum_{\ell,k} M_{k\ell} C_{\ell j} C_{ik},$$

we know that $\hat{G} = CMC$. One nice special case of this formula: for some choices of $\{h_i\}_1^B$, C is the identity matrix and $\hat{G} = M$. We assume C is readily calculable and not too badly conditioned, so that if we can estimate G, we can estimate M as $C^{-1}GC^{-1}$. We now focus on G.

Think of Lh_j as a probability distribution: it corresponds to initializing the sample from a draw $z \sim h_j$, then running a single step of Metropolis-Hastings. By definition, G can be written as an expectation $G_{ij} = E_{Lh_j}[h_i]$. This motivates us to sample from a normalized version of Lh_j and approximate G_{ij} as an average. All we need to do is sample z from h_j, run an M-H iteration on z to get w, and retain w as our sample from Lh_j. EMK: Conjecture: the hats converge to the tildes as you run it for longer, and the tildes converge to the truth as you lengthen the basis.

1.2.1 Basic Estimator Properties

Our approximation can only imitate continuous kernels, i.e. situations where $\int K(x,y) f(y) dy$ can be done with respect to the Lebesgue measure. This presents an obstacle, because with positive probability, the

Metropolis-Hastings algorithm will reject a proposed sample and stay in place. As a workaround, we can approximate the kernel not of a single M-H iteration but of r iterations for r around 10 or 20. The probability of r consecutive rejections is much smaller, pushing the true kernel closer to the subspace in which we approximate it. In section 3, we discuss a variant that explicitly models rejection events.

1.2.2 A more concrete procedure using a Gaussian basis

For something more tangible, we will consider $\Omega = \mathbb{R}^D$, using multivariate Gaussian densities as our basis so that when $h_i(x) = (\frac{1}{2\pi\sigma_i^2})^{\frac{D}{2}} \exp(\frac{(x-\mu_i)^2}{\sigma_i^2})$. This makes it easy to draw starting points for the sampler. Also, C_{ij} can be computed as follows.

EMK: Definitely check this again eventually

$$\begin{aligned}
\int h_i(x)h_j(x)dx &= \int (\frac{1}{2\pi\sigma_i\sigma_j})^D \exp(\frac{-(x-\mu_i)^2}{2\sigma_i^2} + \frac{-(x-\mu_j)^2}{2\sigma_j^2})dx \\
&= \int_{x_1=x_2} (\frac{1}{2\pi\sigma_i\sigma_j})^D \exp(-\frac{(x_1-\mu_i)^2}{2\sigma_i^2} + \frac{-(x_2-\mu_j)^2}{2\sigma_j^2})dx_1dx_2 \\
&= \int_{y_1-y_2+\mu_i-\mu_j=0} (\frac{1}{2\pi\sigma_i\sigma_j})^D \exp(\frac{-y_1^2}{2\sigma_i^2} + \frac{-y_2^2}{2\sigma_j^2})dy_1dy_2 \\
&= \int_{\sigma_i z_1-\sigma_j z_2+\mu_i-\mu_j=0} (\frac{1}{2\pi\sigma_i\sigma_j})^D \exp(\frac{-z_1^2-z_2^2}{2})(\sigma_i\sigma_j)^D dz_1dz_2 \\
&= \int_{\sigma_i z_1-\sigma_j z_2+\mu_i-\mu_j=0} (\frac{1}{2\pi})^D \exp(\frac{-z_1^2-z_2^2}{2})dz_1dz_2 \\
&= f_u(0) \text{ if } u = \sigma_i z_1 - \sigma_j z_2 + \mu_i - \mu_j \text{ and } z\text{'s are standard normal} \\
&= (\frac{1}{2\pi(\sigma_i+\sigma_j)^2})^{\frac{D}{2}} \exp(\frac{-(\mu_i-\mu_j)^2}{2\sigma_i^2+2\sigma_j^2})
\end{aligned}$$

Algorithm 2: BEMC algorithm–stage one

Set M to a matrix of all zeroes.
For $i = 1 : B$
  For $j = 1 : B$
    For $n = 1 : N$
      Draw a sample z_n from h_j, i.e. a normal draw with mean μ_j and variance σ_j^2.
      Run the M-H sampler for ℓ rounds on z_n. Call the result w_n.
      Increment $G_{i,j}$ by $h_i(w_n)/N$.

2 Stage two: calculating the target

We now explain how to get, from our approximation for the kernel, an approximation for the target. We make use of the power method, a simple algorithm for eigenvector computation. Given a matrix M, the power method computes $v \leftarrow M^v$ for any initial vector v, iterating until convergence. For "nice" matrices, this converges rapidly to an eigenvector; for "nicer" ones, the result is always the unique dominant eigenvector. MCMC schemes essentially rely on the same idea: for any initial distribution f, and for a Markov kernel K, turn it into a sample from $Lf = \int K(\cdot, y)f(y)dy$, and iterate; samples from $L^P f$ for some large integer P are good enough because $L^P f$ converges to π.

So, suppose we want a distribution $\hat{\pi}$ with the property $\hat{\pi} = \hat{L}\hat{\pi}$. From the form of $\hat{L}$, we know that $\hat{\pi} \in \text{span}\{h_i | i \in 1...B\}$, so we need only find the vector of coefficients, which we call v. As it turns out, v must be an eigenvector of MC; the dominant eigenvector is the obvious choice.

$$
\begin{aligned}
\sum_{k=1}^{B} v_k h_k(x) = \hat{\pi}(x) \\
= (\hat{L}\hat{\pi})(x) &= \int \sum_{i,j=1}^{B} M_{ij} h_i(x) h_j(y) \sum_{k=1}^{B} v_k h_k(y) dy \\
&= \sum_{i,j,k=1}^{B} M_{ij} h_i(x) v_k \int h_j(y) h_k(y) dy \\
&= \sum_{i,j,k=1}^{B} M_{ij} h_i(x) v_k c_{jk} \\
&= \sum_{i,j=1}^{B} M_{ij} h_i(x) (Cv)_j \\
&= \sum_{i=1}^{B} h_i(x) (MCv)_i
\end{aligned}
$$

Algorithm 3: BEMC algorithm–stage two

Given an estimate $\hat{G}$ of G:
For each i, j pair, compute C_{ij} as $(\frac{1}{2\pi(\sigma_i+\sigma_j)^2})^{\frac{D}{2}} \exp(\frac{-(\mu_i-\mu_j)^2}{2\sigma_i^2+2\sigma_j^2})$.
Compute $MC = C^{-1}\hat{G}$.
Compute the leading eigenvector v of MC.
Return $\sum_i v_i h_i$ as a posterior estimate

3 BEMC-R, a variant modeling rejections

As we mention in section 1.1, our scheme is able to model continuous kernels. On the other hand, the Metropolis-Hastings algorithm sometimes rejects proposed samples, and its kernel will assign positive mass to intervals on the "diagonal" set $\{x, y \in \Omega^2 | x = y\}$. In this section, we introduce a variant of BEMC that explicitly models rejections by the sampler.

Let us look at the Metropolis-Hastings kernel in more detail. Going back to the algorithm, the quantity $\alpha(x|y)$ is the probability of accepting a move from y to x. For convenience, let

$$\alpha(y) = P(\text{accept next move} \mid \text{currently at } y).$$

Splitting up the next draw as an alternative between moving and staying put, we can write $K(x, y) = (1 - \alpha(y))\delta_y(x) + \alpha(y) f_{acc}(x|y)$. In this expression, $f_{acc}(x|y)$ is the conditional density of x given that our move out of y was not rejected. Though it will help in our development, we acknowledge this quantity is strange to consider, because we never observe a rejection or acceptance without also knowing where the proposal was. Certainly, f_{acc} is not the same as $q(x_2|x_1)$, since conditioning on the acceptance increases the chance that we moved into a region of higher probability. To set up the last line below, define the operator D_{rej} from α so that $(D_{rej} f)(x) \equiv (1 - \alpha(x))f(x)$, define $K_{acc}(x, y)$ as $f_{acc}(x|y)\alpha(y)$, and let $(L_{acc} f)(x) \equiv \int K_{acc}(x, y) f(y) dy$.

If y precedes x in the sampler, we can relate their PDF's with these operators.

$$\begin{aligned}
f_x(x) &= \int K(x,y)f_y(y)dy \\
&= \int ((1-\alpha(y))\delta_y(x) + \alpha(y)f_{acc}(x|y))f_y(y)dy \\
&= \int (1-\alpha(y))\delta_x(y)f_y(y)dy + \int K_{acc}(x,y)f_y(y)dy \\
&= (1-\alpha(x))f_y(x) + \int K_{acc}(x,y)f_y(y)dy \\
&= (D_{rej}f_y)(x) + (L_{acc}f_y)(x)
\end{aligned}$$

We can sample from a pdf proportional to $D_{rej}f$ by sampling y from $f_y()$, then running an M-H iteration on y to get x and retaining the sample y if $x = y$. We can sample from a pdf proportional to $L_{acc}f$ by doing nearly the same steps, but retaining x if $x \neq y$. These facts will be useful as we attempt to estimate L_{acc}.

To define another set of "tilde" objects, let $\tilde{M}_{acc}$ be the matrix that minimizes the squared $\mathcal{L}_2$ distance $\|K_{acc} - \sum_{i,j}^B \tilde{m}_{ij} h_i \otimes h_j\|^2$, or in other terms the matrix that minimizes the integral

$$\int_{\Omega\times\Omega} (K_{acc}(x,y) - \sum_{i,j}^B \tilde{m}_{ij}h_i(x)h_j(y))^2 dxdy.$$

We will use a separate set of functions $\{\phi_i\}_{i=1}^{B_\phi}$ to approximate α. Let $\tilde{r}$ be the vector that minimizes the integral $\int_\Omega (\alpha(x) - \sum_{i=1}^{B_\phi} \tilde{r}_i\phi_j(y))^2 dx$ and let $\tilde{r}$ be $\sum_{i=1}^{B_\phi} \tilde{r}_i\phi_j(x)$

This time around, we will try to estimate a function $\hat{\alpha}$ and a matrix M so that $\hat{\alpha} \approx \tilde{\alpha}$ and $\hat{L}_{acc} \approx \tilde{L}_{acc}$.So, we need still need to estimate M, but with the added complication of trying to infer $\hat{\alpha}$ at the same time. Fortunately, it is easy to tell when the sampler rejects and when it doesn't, and this provides a way to tease out information about α. Suppose for a moment that we start the sampler at a point y and it takes a single step to x. If $x \neq y$, then the sampler has shown less of a tendency to reject starting from y, and we label y with a 0. If $x = y$, we label y with a 1. Once Ω is covered in zeroes and ones, there are many probabilistic classifier methods that could give an estimate of $\hat{\alpha}$, which at any given point is just the probability of labeling with a one. Meanwhile, whenever the sampler moves, we gain information about L_{acc}, and we can update M as before.

This strategy still throws away useful information. To see why, recall that the Metropolis-Hastings algorithm makes a proposal, computes an rejection probability, flips a metaphorical coin with that probability, and then discards the rejection probability. When drawing a chain of samples, the rejection probability serves no further purpose, so discarding it is natural. In BEMC-R, though, we can keep it to provide a more efficient estimate of α. If the rejection probability when proposing a move to y from x is p, then the better procedure is to label y with p. Likewise, instead of updating the estimate of M_{ij} using a sample of weight 1 with probability p, we can update it using a sample of weight p with probability 1.

We summarize the procedure in Algorithm 4.

For one further refinement, we could include some prior information about M. Since L_{acc} mimics the action of the sampler as it moves, it might resemble the action of the proposal alone, with no rejections. That would mean $\hat{g}_{ij} \approx \int h_i(x)q(x|y)h_j(y)dydx$. For simple proposal distributions like a uniform or normal centered on the current value, this integral may be easy to find as a convolution.

3.1 A concrete choice of basis for BEMC-R

We can implement this rejection-tolerant version on $\mathbb{R}^D$ using a list of Gaussians for $\{h_i\}_{i=1}^B$ like in section 1.2.2. In selecting $\{\phi_i\}_{i=1}^{B_\phi}$, we want to make it easy to compute inner products of the form $\int \phi_k(x)h_i(x)h_j(x)dx$. We also want something appropriate to express α as it occurs naturally. These requirement suggest using constants, Gaussians, or EMK: if all else fails Hermite polynomials. EMK: ugh what a mess that will be.

Algorithm 4: BEMC-R algorithm–stage one

Set M to 0.
Set a scalar W to zero. W is the effective number of samples in an estimate of an entry of M.
Set $T = \{\}$. T will be the training set for $\hat{\alpha}$.
For $b_{in} = 1 : B$
  For $b_{out} = 1 : B$
    For $n = 1 : N$
      Draw a sample y_n from $h_{b_{in}}$.
      Draw a proposal $x_n|y_n$ and compute its rejection probability p.
      Add (y_n, p) to T.
      Increment $M_{b_{out},b_{in}}$ by $ph_{b_{in}(x_n)}$.
      Increment W by p.
    Divide $M_{b_{out},b_{in}}$ by W.
Train on T to get $\hat{\alpha}$, represented by $\hat{r}$ EMK: Possibility here to choose ϕ's adaptively?

EMK: Can we do those integrals if we include a link function? Maybe a probit link would allow crafty integration by parts??

In BEMC as presented in section 1.1, there was nothing to gain by representing our estimate $\hat{\pi}$ of the target outside of the span of $\{h_i\}_{i=1}^B$: any component orthogonal to span$\{h_i\}_{i=1}^B$ would get zeroed out upon a single application of $\hat{L}$. This is no longer the case; because of the diagonal term D_{rej}, we have no guarantee that our approximation stay within any particular finite-dimensional subspace. We still need to represent $\hat{\pi}$ in computer memory, so for now, we'll sweep the issue under the rug and consider only approximations that we can write as $\hat{\pi}(x) = \sum_{i=1}^B v_i h_i(x)$. Our final approximation to the transition kernel will be

$$P_{\text{span}\{h_i\}}(\hat{D}_{rej} + \hat{L}_{acc}),$$

where $P_{\text{span}\{h_i\}}$ is the orthogonal projector onto the set of functions expressible as $\sum_{i=1}^B v_i h_i(x)$.

EMK: also considered $\hat{\pi}(x) = \sum_{i,k=1}^{B,B_\phi} v_{ik}\phi_k(x)h_i(x)$.This would require "full-house" integrals with three ϕ terms and two h terms. Since we will choose ϕ_1 to be constant, this form is at least as expressive as its rejection-neglecting predecessor. Could also choose yet another basis for this, but it would have to play nice with the ϕ's and h's anyway.

3.2 Computing the steady state in BEMC-R

Given M and $\hat{r}$, we follow the same tactic as in section 2. We define matrices $C^{(k)}$ so that $c_{ij}^{(k)} = \int \phi_k(x)h_i(x)h_j(x)dx$.

$$\begin{aligned}
\sum_{\ell,k=1}^{B,B_\phi} v_{\ell k}\phi_k(x)h_\ell(x) &= \hat{\pi}(x) \\
&= (P_{\text{span}\{h_i\}}(\hat{D}_{rej} + \hat{L}_{acc})\hat{\pi})(x) \\
&= P_{\text{span}\{h_i\}} \sum_{i=1}^{B_\phi} \phi_i(x) \sum_{\ell}^{B} v_\ell h_\ell(x) + \int \sum_{i,j=1}^{B} M_{ij} h_i(x) h_j(y) \sum_{\ell=1}^{B} v_\ell h_\ell(y) dy \\
&= P_{\text{span}\{h_i\}} \sum_{i=1}^{B_\phi} \phi_i(x) \sum_{\ell}^{B} v_\ell h_\ell(x) + \sum_{i,j=1}^{B} M_{ij} h_i(x) \sum_{\ell=1}^{B} v_\ell \int h_j(y) h_\ell(y) dy
\end{aligned}$$

References

[1] Weisstein, E.W.: Hermite polynomial. From MathWorld–A Wolfram Web Resource. (December 2014)

[2] Golub, G.H., Welsch, J.H.: Calculation of Gauss quadrature rules. Math. Comp. **23**(106) (1969) 221–230 Loose microfiche suppl. A1–A10.

[3] Blocker, A.W.: fastGHQuad: Fast Rcpp implementation of Gauss-Hermite quadrature. (2014) R package version 0.2.

[4] Bunck, B.: A fast algorithm for evaluation of normalized hermite functions. Bit Numer Math **49**(2) (2009) 281–295