Reversible Jump MCMC for Model Evaluation

EECS 126

Rahul Kumar
`rahulkumar@berkeley.edu`

Sawyer Brunn
`sawyerbrunn@berkeley.edu`

Mengyang Zhang
`mengyang63@berkeley.edu`

December 6, 2019

1 Introduction

Metropolis-Hastings is a common tool for sampling from distributions where a direct calculation of the density may be intractable to compute. Instead, Metropolis-Hastings permits sampling from a distribution if we can calculate the density up to a normalizing factor. Reversible jump sampling can be seen as a generalization of Metropolis-Hastings where the chain is permitted to jump between states that may have different dimensions. We first describe reversible jump sampling, and then discuss the results of running it on a weather dataset. We note that this is a significant departure from our proposal, as it turns out that RJMCMC needs quite a bit of tuning and is often slow to converge except for fairly simple models.

2 Methods

Much like Metropolis-Hastings, the goal of Reversible Jump MCMC is to create a Markov chain with a given stationary distribution π, which we only know up to a constant factor. Much of our discussion on RJMCMC is based on [1] and [2], and we approximately follow the notation they use. Here, we only aim to summarize the key equations in RJMCMC - there is much more depth to the derivation than can be presented here.

Since our chain must be reversible, we must satisfy the detailed balance equations:

$$\pi(x)P(x,y) = \pi(y)P(y,x)$$

for all states x and y, where $P(\cdot,\cdot)$ is the transition probability function for the chain.

Metropolis Hastings ensures that the detailed balance equations hold by proposing a new state as a sample from $g(x,\cdot)$ (where x is the current state), and then moving to that new state with probability

$$A(x,y) = \min\{1, \frac{\pi(y)g(y,x)}{\pi(x)g(x,y)}\}$$

Reversible jump functions similarly to Metropolis-Hastings, except that it must deal with the additional complexity of jumping between states. To derive the acceptance probability of cross-dimensional jumps, we integrate the detailed balance equations, and note that $P(x,y) = g(x,y)A(x,y)$ - that is, we transition only when a state is proposed and accepted. This gives:

$$\int_X \int_Y \pi(x)g(x,y)A(x,y)dydx = \int_X \int_Y \pi(y)g(y,x)A(y,x)dydx$$

Here, X and Y are subsets of R^k for some k. At this point, many derivations of RJMCMC introduce an additional piece of notation: we assume that given the state x of the chain and a vector of random variables w distributed according to a known proposal density $g(w)$, the proposed state y is determined. That is, if $g(x,y) = g(w)$ for a particular choice of w, then there is a deterministic funtion h such that $[y,z] = h(x,w)$, where y is the new state, and z is some extra data padded so that the dimensions match. We also assume that there is a known density $f(z)$ that describes the reverse move. After some substitutions (see [2]), we can rewrite the integrated DBEs as

$$\iint \pi(x)g(w)A(x,y)dwdx = \iint \pi(y)f(z)A(y,x)dydz$$

(We will ignore the requirements of what sets the integration must be carried out over, as a proper discussion of this requires a bit of set theory and calculus).

The final step is to choose an $A(x, y)$ that makes the quantities being integrated the same, ie. we want

$$\pi(x)g(w)A(x,y)dwdx = \pi(y)f(z)A(y,x)dydz.$$

To satisfy this, we introduce the Jacobian matrix J, which permits a change of coordinates from $dwdx$ to $dydz$ [2].

Then, we choose

$$A(x,y) = \min\{1, \frac{\pi(y)f(z)}{\pi(x)g(w)}|J|\},$$

where $|J|$ is the determinant of the Jacobian from from (x, w) to (y, z). Our implementation uses TensorFlow for automatic differentiation, although it is not difficult to calculate the Jacobian symbolically for the models we test, as we now describe.

3 An Example

Here, we present an example application of RJMCMC that we used to validate our implementation and theory.

Our goal was to fit a model to some given data vector Y, where each element of Y is assumed to be i.i.d. More precisely, we wanted to determine if $Y_i \sim$ Poisson(λ) or if $Y_i \sim$ Binomial(n, p), and we also wanted to find the parameters of the distribution. Note that the reversible jump machinery is required here, since the Poisson distribution has one parameter, whereas a Binomial distribution has 2.

Our reversible jump sampler was setup to choose between two models. Model 0 was the Poisson model, and model 1 was the Binomial model. Let the variable $k \in \{0, 1\}$ denote the index of the currently selected model. To better structure our proposals, we first sample from a model proposal distribution $g(\cdot|k)$ that proposes a new model given the current model. Then, given the new model proposal k', the current model k, and the current model parameters θ, we select the candidate model's parameters from a distribution $g(\cdot|k, k', \theta)$. Finally, our acceptance probability is as described above.

Our state representation is $(1, \lambda)$ if we are in the Poisson model ($k = 0$) and $(2, n, p)$ if we are in the Binomial model ($k = 1$). Within-model moves were described using a clipped normal offset: we propose a zero mean, normally distributed offset, add it to the previous state, and produce a proposal state by clipping the result to a specified range. For instance, we require the Binomial parameter p to be in the range $[0, 1]$.

The primary challenge in setting up the problem is describing appropriate across-model moves.

For moves from model 0 to model 1, we select a random variable $w \sim$ Uniform$(0.1, 0.9)$, and append this value to our state vector. Thus, our modified state is a vector (λ, w). To propose a state under the binomial model, which requires parameters n and p, we chose $n = \lambda/w$ and $p = w$. In this manner, we ensure that the proposal state has the same mean as the previous state. The determinant of the Jacobian in this case is (after a bit of algebra) $1/u$.

For moves from model 1 to model 0, we have a deterministic proposal designed to reverse the process described above. That is, if our current state is (n, p), we propose a new state (λ, w) so that $\lambda = np$ and $w = p$. The Jacobian here has determinant p, which makes sense, as it is the reciprocal of the 0 to 1 transition Jacobian.

Our likelihood function is straightforward: we just multiply the PMFs of the model evaluated at each of the data points in Y. Some plots of our sampler running on test data are included in the experiments section. The results section contains plots showing our sampler distinguishing between Poisson and Binomial distributions on simulated data.

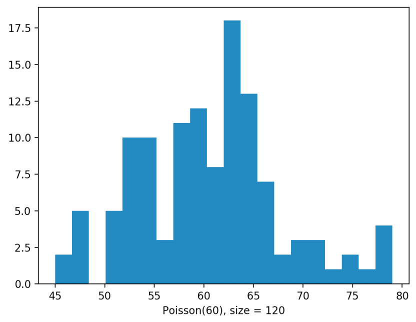
4 Experiments

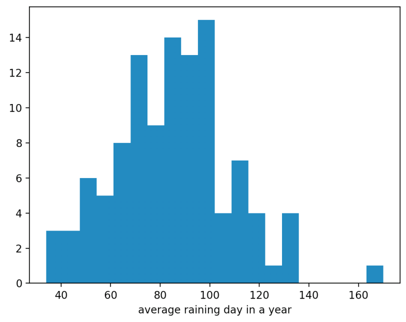
For a more real-life application of the reversible jump sampler, we explored Australian weather data [3]. We wanted to see if RJMCMC could be used to identify and fit a distribution for the number of rainy days in 365 day intervals. The

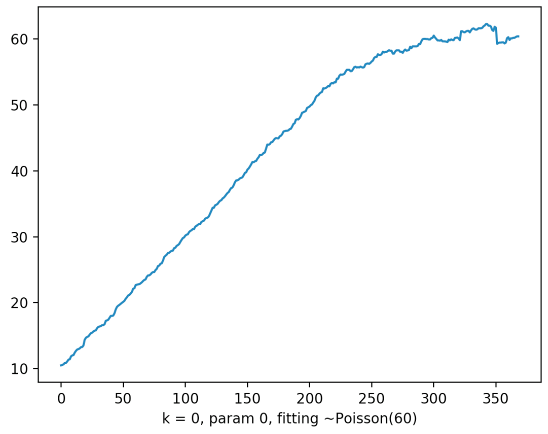
Data set we use records the whether condition of more than 140,000 days for different areas in Australia, and each area has a cords of more than 10 years. [4] We represent no rain as a 0 and rain as a 1. We count the number of rainy days in a year. We feed 100 years of data to the RJMCMC sampler to fit a model better, and generate graphs based on the result of our prediction.

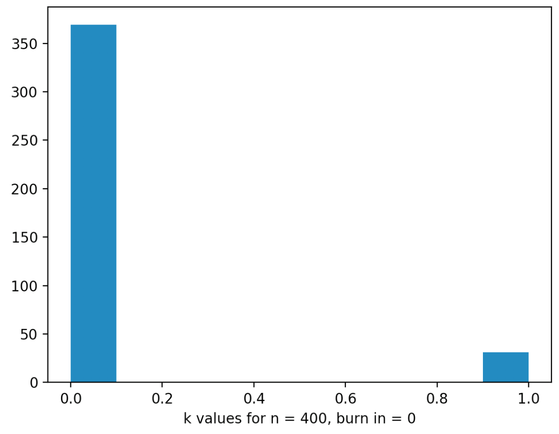
5 Results

We first show the results of running our sampler on the simple case where we want to distinguish between a Binomial and a Poisson distribution. That is, data is sampled from a distribution (either Poisson or Binomial) unknown to the algorithm, and the sampler must select the correct distribution and its parameters. A histogram of the training data is shown, followed by a histogram of the weather data, the estimated Poisson mean, and the fraction of time spent in each model.

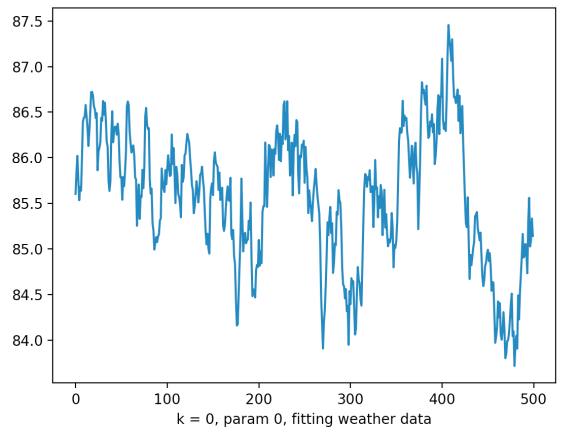


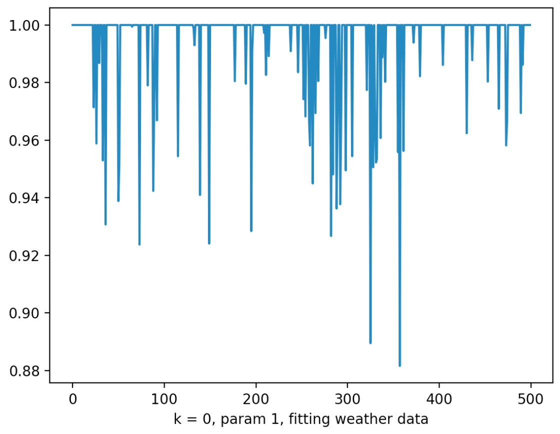




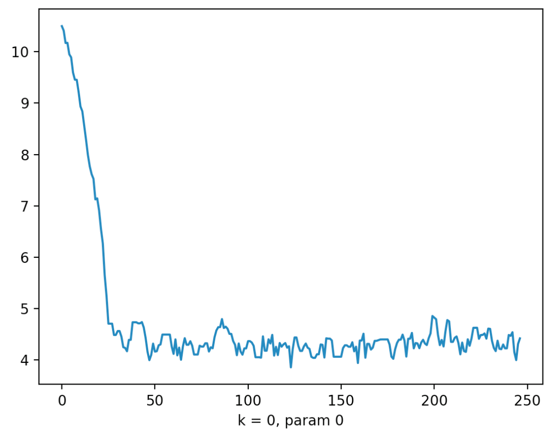


The behavior of the sampler on the weather dataset is shown below:





Finally, a very strong example of convergence of the estimated mean when the training data comes from a Poisson distribution with parameter 4:



The model was close to convergence when estimating the Poisson distribution with $\lambda = 60$ (this can be confirmed by running the included code). However, the model mean does not converge for the weather model, at least not for the number of samples we ran. We believe this is because the proposal function needed additional tuning.

6 Discussion

We should note that tuning the proposal distribution can be extremely finicky - overly conservative proposals (eg. only straying a little bit from the current state) can be problematic, especially if the start state is far from the true distribution, as we run into issues of numerical precision. On the other hand, more aggressive proposals can sometimes make the state oscillate more. Even after a reasonable effort to tune the chain, the samples can still be quite noisy. Additionally, some experiments have very long burn in times (again, more of an issue when the chain starts far from stationarity), and we imagine that for some instances, the chain may not converge in any reasonable time. In some cases, altering the sample size can alter the stability of the accepted samples because the likelihood function can become much more sharp or dull. Increasing the number of samples and burn in time usually let to more accurate, reliable results, but in some distributions it made the proposals and acceptance probability so unstable that the distribution could jump around even after being relatively close to the correct distribution parameters.

References

[1] Peter J. Green and David I. Hastie. Reversible jump MCMC. June 13, 2009. http://people.ee.duke.edu/~lcarin/rjmcmc_20090613.pdf.

[2] University of Colorado Boulder, Department of Applied Mathematics. Reversible Jump MCMC (RJMCMC) https://www.colorado.edu/amath/sites/default/files/attached-files/rjmcmc.pdf

[3] Brian M. Hartman and Jeffrey D. Hart. Using Reversible Jump MCMC to Account for Model Uncertainty https://www.soa.org/globalassets/assets/files/static-pages/research/arch/2009/arch-2009-iss1-hart.pdf

[4] Joe Young. Rain in Australia. https://www.kaggle.com/jsphyg/weather-dataset-rattle-package