Adaptive Parallel Tempering MCMC

Anonymous Author(s)
Affiliation
Address
email

Abstract

This paper first provides an overview of the Metropolis-Hastings Markov-chain Monte Carlo (MCMC) algorithm, and then discusses Parallel Tempered(PT) MCMC and Adaptive MCMC. These have come into practice in order to address problems that occur in practice when using straight forward implementations of the Metropolis-Hastings algorithm. An experiment is then proposed that compares two methods for automatically tuning the parameters of the PTMCMC method.

1 Metropolis-Hastings Markov-chain Monte Carlo Algorithms

MCMC algorithms are a very widely used tool for calculating integrals of complicated and high dimensional distributions that occur in a range of contexts, from computational physics and biology to Bayesian statistics[1]. If one were to try to do these integrations in a straight forward manner,namely by evaluating the distribution deterministically over the entire state space at a set resolution, the time necessary for the computation would quickly become prohibitive. These distributions, as a function of their dimensionality, take an exponential number of evaluations to integrate in this manner for a given sampling resolution. MCMC algorithms take a different approach. Rather than providing a high precision result only at the end of the computation, the Metropolis-Hastings MCMC algorithm (Algorithm 1) takes a stochastic approach, and provides an approximation that gradually becomes more accurate over the time the program executes.

Algorithm 1: Metropolis-Hastings MCMC

Input : $p(x)$: Target distribution
$q(x)$: Proposal distribution
N_{iter}: Number of sample iterations
Output: $\{x^i\}$: Chain of samples
begin
  Initialize $x^{(0)}$
  for $i = 1 \dots N_{iter}$ **do**
    Sample $u \in U[0, 1]$
    Sample $x^* \in q(x^*|x^{(i)})$
    $\alpha = \min\left(1, \frac{p(x^*)q(x^{(i)}|x^*)}{p(x^{(i)})q(x^*|x^{(i)})}\right)$
    if $u < \alpha$ **then**
      $x^{(i+1)} = x^*$
    else
      $x^{(i+1)} = x^{(i)}$

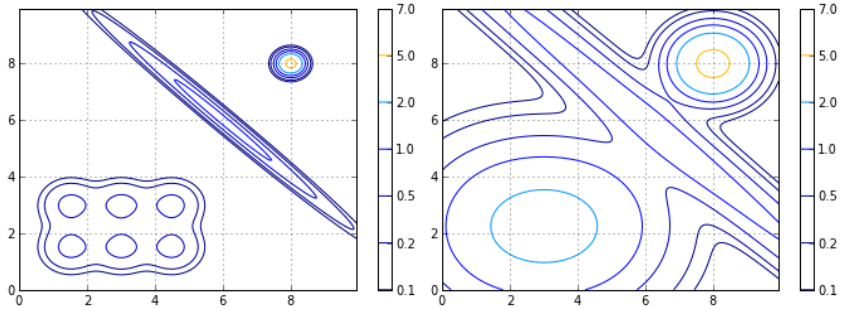
Starting from a given, or randomly generated, initial sample over the state space, the algorithm uses a stochastic transition function to produce a new, though not necessarily different in value, sample

using the proposal distribution $q(x^*|x^{(i)})$. The α value determines the acceptance probability for the newly generated sample, and it is what guarantees the chain will become an approximation of $p(x)$, the target distribution. The transition function is then recursively applied to each new sample, producing a chain of samples. As long as the transition function can take the chain over the entire state space of the distribution, the chain will eventually approximate the target distribution, and as the algorithm runs that approximation will become more accurate. Once the chain has covered the range, where the range consists of the areas of the state space of statistical interest, of the distribution in question, the chain is said to have mixed.

It is one very desirable property to have a fast mixing chain, as this means the distribution will have 'forgotten' where it started and have no bias towards being in the starting location. However, it is also often the case that the chain will be initialized in a location of extremely low probability. The first stretch of the chain will then make the rest of the chain a biased approximation for all but very large numbers of samples, so it is often the case that a section of the chain at the beginning is removed once it has reached a region of non-negligible probability, or after a few thousand samples (depending on the distribution). The removed section of the chain is called the burn-in. For more complete information on MCMC see [1].

The basic MH algorithm uses a $q(x^*|x^{(i)}) = x^{(i)} + N(0, I)$ which is a step of random direction and length in the state space from the previous point[2]. Using this transition function the M-H algorithm will eventually approximate any distribution it is given, but here 'eventually' is in the mathematical sense and it guarantees that we will have good results assuming that we have forever to wait for our program to run. In practice, however, we want reliable results as soon as possible. In order to make sure the chain mixes quickly, the random step should be wide enough to mix quickly, meaning the average acceptance isn't too high, and to make sure that the average rate of new states that are accepted is not too low which reduces deficiency of the algorithm.

This is where the idea of adaptive MCMC comes in [2], which aims to automatically tune the parameters of the transition function towards good acceptance rates [3], here the parameter is the width of the random step distribution. Figure 1(a) shows an example distribution that the basic MH algorithm would have trouble sampling for any single transition function. In this distribution, regions of high probability are separated by regions of low probability, making it unlikely that the chain will visit all regions in a short amount of time for a small random step, or making for a very low average acceptance rate for a large random step. The Parallel Tempered MCMC (PT) algorithm is designed to overcome exactly this challenge.



(a) Example Distribution, t = 1 (b) Example Distribution, t = 10

Figure 1: a) shows an example distribution with different high probability regions that would be difficult to sample using the basic random walk MH algorithm. b) shows the same distribution that has been tempered with t = 10

1.1 Parallel Tempered MCMC

PTMCMC is a method for generating candidate samples from all over a distribution, overcoming low probability regions between areas of importance.

The inspiration for the parallel tempering MCMC algorithm comes from the idea that a temperature parameter could be used to flatten out the target distribution, see Figure 1(b) . As the temperature of the distribution is raised the distribution flattens out, making the random walk chain for that temp more likely to mix quickly. Using the exponential in terms of negative energy $p(x) = Exp(-E(x))$, expressing higher temperature distributions is also relatively easy as $p(x, t_k) = p_k(x) = Exp(-E(x)/t_k)$, which is related to the effect that temperatures have on some physically based distributions, like the Boltzmann distribution.

Algorithm 2: Parallel Tempered MCMC

Input : $\{t_k\}$: Temperature set
M: Number of temperatures
N_{iter}: Number of iterations per sweep
N_{sweep}: Number of sweeps of PT
Output: $\{x_k^i\}$: Chains of samples
begin
  for $i = 1 \dots N_{sweep}$ **do**
    for $k = 1 \dots M$ **do**
      $\{x_k^i\}$ =Metropolis-Hastings($p_k(\cdot)$, $q(x^*|x^{(i)}) = x^{(i)} + N(0, I/t_k)$, N_{iter})
    for $i = 1 \dots M - 1$ **do**
      Swap x_k^i with x_{k+1}^i with probability $\alpha_k = \min\left(1, \frac{p_k(x_{k+1})p_{k+1}(x_k)}{p_k(x_k)p_{k+1}(x_{k+1})}\right)$

Once we have an expression for the target distribution for a given temperature parameter $p_k(x)$, which we will call replicas, we run the basic MCMC algorithm on each distribution. After a number of iterations on each replica, called a sweep, the current samples are considered probabilistically for exchanges between different temperature levels with probability α_k, which is analogous to the α acceptance probability within one chain. In practice only pairs between neighboring temperature values are considered for swapping, where the chances of accepting a trade are more likely to be higher.

The lowest temperature setting is the target distribution $t_1 = 1$, with several distributions of higher temperatures to accommodate the full exploration of the distribution. A high temperature sample can travel anywhere the target distribution has high probability, 'cool' as it is swapped to lower temperatures, then be considered for acceptance in the target distribution chain. This can effectively produce candidate samples in regions that are not otherwise reachable, on practical time scales, because the high temperature chains can cross regions that that have relatively low probability in the cooler chains[4].

This technique very effectively overcomes much of the shortcomings of basic random walk MCMC for disconnected distributions, where here disconnected means there are high probability regions in the distribution that are very unlikely to be sampled in the same chain, with steps yielding good acceptance rates. Without this technique the random walk step would have to be made very large to ensure mixing, but this would increase computation time because of the low acceptance rate that would likely result. The simulation of multiple chains does incur a linear increase of time complexity, however in practice the algorithm is often still more effective than the non-tempered approach, including the linear penalty [5].

Parallel Tempered MCMC does however have an additional set of parameters that need to be tuned before performance becomes optimal, which are the number of replicas M, and their temperatures $\{t_k\}$. Poor spacing of the temperatures can cause the replica systems to be too far apart, limiting the exchange chancesα_k , or too close, limiting the improvement in sample variety. It is often difficult even for experts to provide good temperature spacings, and adaptive approaches have been introduced to automatically tune the number of replica systems, and the temperature spacings. For more in depth about Parallel Tempered MCMC see [4].

1.2 Adaptive PTMCMC

The benefit of having multiple tempered replicas of the target system is that the samples from one area can be exchanged between temperatures, and are able to eventually wonder everywhere over the range of $p(\cdot)$. In order to make this a reality we need to make sure the replicas are at the right temperature spacings, to allow for a good average swapping rate, α_k, over all sweeps.

However, making sure that there are exchanges between neighboring replicas is not enough to guarantee that the samples are mobile beyond just one transition[5]. For example, the samples might only be exchanged in cycles between a few neighboring temperature levels without ever traveling to the highest or lowest levels, making that replica, $p_k(\cdot)$, and its neighbors useless for sampling the distribution we are interested in. What we really want is for the samples to be exchanged between the lowest, $p_1(\cdot)$, and highest, $p_M(\cdot)$, replicas after wondering through the intermediate replicas.

We measure this as $F(t_k)$ which is the fraction of upward traveling samples to visit replica system k [5]. An upward traveling sample has visited the lowest temperature replica more recently than the highest temperature replica, the total number of upward traveling samples to visit replica k is $n_{up}(t_k)$. Similarly a downward traveling sample has visited the highest temperature replica more recently, and the total number of downward traveling samples to visit replica k is $n_{down}(t_k)$. Samples that have not yet reached either of the terminal replicas do not contribute to either count. With this notation:

$$F(t_k) = \frac{n_{up}(t_k)}{n_{up}(t_k) + n_{down}(t_k)} \tag{1}$$

The FOPT adaptive method changes the temperatures based on making $\Delta F = F(t_k) - F(t_{k+1})$ a constant value for any adjacent replica systems. This means that the fraction of upward moving particles increases linearly as one moves from the highest temperature replica to the target system, where $F(t_M) = 0$ and $F(t_1) = 1$. Intuitively this means that we want each replica system to contribute equally to the flow of samples between the high and low temperature systems, so that ideally no single replica system is less useful than any other. For more information on feedback optimized parallel tempering MCMC, see [5].

Algorithm 3: Adaptive PT Template

Input : $\{t_k\}$: Initial parameter set
M: Number of parameters
N_{iter}: Number of feedback iterations
N_{sweep}: Number of sweeps of PT within an iteration
Output: $\{t_k\}$: Optimized parameter set
begin
  for $i = 1 \dots N_{iter}$ **do**
    ParallelTempering(λ_k) for N_{sweep} steps, calculate F
    $\{\lambda_k\}$ = Adaptive Update(F,$\{\lambda_k\}$)

1.2.1 Feedback Optimized Parallel Tempering

This would involve replacing the Adaptive Update function with the Feedback Optimized PT Update, Algorithm 4, function[5]. The algorithm moves the replica systems away from where F has a low slope, towards where it has a high slope, in the hopes that the ΔF will be made constant between consecutive replicas.

1.2.2 Bayesian Inference Optimization

Another approach would be to use Bayesian inference to calculate a choice of parameters, using a parametric bandit interpretation of the problem. This method would be based on the method proposed in[6], which was originally tested on Hybrid Monte Carlo simulation tuning, but is fairly general. The method uses radial basis functions (BRMs) to model nonlinearities. A set of $\{t_k\}$ is

Algorithm 4: Feedback Optimized PT Update

Input : $\{t_k\}$: Initial parameter set
$F(t_k)$: The upward flow fraction
Output: $\{t_k\}$: Optimized parameter set
begin
Define $g(f)$ such that:
1. $g(f(\lambda_k)) = \lambda_k$
2. Within $(\lambda_k, \lambda_{k+1})$, $g(f)$ is a linear interpolation between $g(f(\lambda_k))$ and $g(f(\lambda_{k+1}))$

for $k = 2 \dots M-1$ **do**
$\lambda_k \leftarrow g((k-1)/(M-1))$

the action taken, and by expanding the action in terms of the RBMs a linear model can be used. The reward function would use a measure of the average deviation of the ΔF between consecutive replicas. For low variations in the slope of F, ie F is close to linear, the reward would need to be high , and for high variations in slope, or highly nonlinear F, the reward should be low. This method would also steer the spacing of the replica systems so that each would contribute roughly the same amount to the global exchange between the target system and the highest temperature replica. Using Bayes Rule we can derive the update for our model[6], and use Algorithm 5 to update $\{t_k\}$ [6].

Algorithm 5: Bayesian Inference Update

Input : $\{t_k\}$: Initial parameter set
$F(t_k)$: The upward flow fraction
Output: $\{t_k\}$: Optimized parameter set
begin
Compute the standard deviation of ΔF , as σ
Augment the data $D_{1:i} = \{D_{1:i-1}, (\{t_k\}_i, \sigma)\}$
Update the Linear Model
Find $\{t_k\}^*$ by optimizing the acquisition function $argmax_{\{t_k\}^*}[u(\{t_k\}^*|D_{1:i})]$
Draw $u \in U[0,1]$
if $u < p$ **then**
$\{t_k\}_{i+1} = \{t_k\}^*$
else
$\{t_k\}_{i+1} = \{t_k\}_i$
end if

2 Experiment Proposal

The experiment I propose here would compare the FOPT, Algorithm 4, method of adjusting the temperature parameters, to the Bayesian inference method, Algorithm 5. As PT is a fairly widely used algorithm, finding an efficient and general method for automatically tuning $\{t_k\}$ may increase the variety of distributions that can be effectively approached with this method. The experiment would test the two methods on at least one procedurally generated distribution, somewhat like that in image 1, but likely in higher dimensions. Ideally it would also be tested using a few distributions used in different fields, so that the results would be as close to real practice as possible. If the Bayesian inference method turns out to be competitive with FOPT, it could then be compared to the Robust Feedback Optimized PT algorithm proposed in []. Robust FOPT is more stable as it also allows for the introduction of more replicas into the system, in order to maintain a minimum swap rate between neighboring replicas. Data collected for the experiment would be in the form of the time of execution, and the relative accuracy of the results as a function of run time, and the time taken to do the optimization.

3 Conclusion

Parallel Tempering MCMC is a popular method for simulating and integrating high dimensional probability distributions, and these adaptive techniques discussed here show promise of expanding the applicability of the technique. It could be beneficial to carry out the proposed experiment, and is unfortunate I was unable to do so in time.

References

[1] Andrieu, Christophe. de Freitas, Nando. Doucet, Arnaud. Jordan, Michael I. (2003) *An Introduction to MCMC for Machine Learning* . Machine Learning 50, pp. 5-43. Kluwer Academic Publishers. http://homepage.psy.utexas.edu/homepage/group/loveLAB/love/classes/CompCogsci/mcmc-ml.pdf

[2] Roberts, Gareth O. Rosenthal, Jeffrey S. (2008) *Examples of Adaptive MCMC* http://probability.ca/jeff/ftpdir/adaptex.pdf

[3] Atchade, Yves. Fort, Gersende. Moulines, Eric. Priouret, Pierre. (2011) *Bayesian Time Series Models, Ch1*. Cambridge University Press. www.stat.lsa.umich.edu/ yvesa/afmp.pdf

[4] Earl, David J. Deema, Michael W. (2008) *Parallel Tempering: Theory, Applications, and New Perspectives* http://arxiv.org/abs/physics/0508111v2

[5] Hamze, Firas Dickson, Neil. Karimi, Kamran. (2010) *Robust Parameter Selection for Parallel Tempering* D-Wave Systems. http://arxiv.org/abs/1004.2840v1

[6] Wang, Ziyu. de Freitas, Nando. (2011) *Adaptive Hybrid Monte Carlo with Bayesian Parametric Bandits and Predictive Adaptation Measures* http://www.cs.ubc.ca/ nando/publications.php