Firefly Monte Carlo: Exact MC with data subsets

Feynman Liang, Max Chamberlin, Kai Xu

April 30, 2016

Contents

1 Introduction

Many methods in Bayesian inference involve computing the likelihood $L(\theta)$ of parameters θ over iid observations $X = \{x_n\}_{n=1}^N$, defined as

$$L(\theta) = P(X|\theta) = \prod_{n=1}^{N} P(x_n|\theta) = \prod_{n=1}^{N} L_n(\theta) \tag{1}$$

For example, Monte Carlo Markov Chain (MCMC) are a class of methods often used to sample posterior distributions $P(\theta|X)$ with intractable partition functions $Z = \int P(X|\theta)P(\theta)d\theta$. When performing Metropolis-Hastings MCMC, at each iteration we need to compute the acceptance probability

$$A(\theta'|\theta) = \min\left\{1, \frac{q(\theta|\theta')L(\theta')P(\theta')}{q(\theta'|\theta)L(\theta)P(\theta)}\right\}$$

This means that each iteration requires at least one evaluation $L(\theta')$ of the likelihood over all N points ($L(\theta)$ can be cached from the prior iteration). Unfortunately, in the era of "Big Data" N may be on the order of petabytes and computing $L(\theta)$ at each iteration may be prohibitively expensive.

Many methods have been proposed to tractably estimate Equation 1. Firefly MC (FlyMC) belongs to a class of methods which approximate $L(\theta)$ using smaller tractable subsets of the total dataset. In FlyMC, auxiliary "brightness" variables are first introduced to decrease the number of likelihood evaluations per iteration. The Markov chain state and auxiliary variables are then successively Gibbs sampled. For certain inference problems, FlyMC can provide a significant speedup (up to 12× in our experiments, Table 1) over traditional MCMC even after accounting for differences in expected sample size.

1.1 Rationale for selecting the paper

We chose the Firefly MC paper [MA14] because of the ubiquity of MCMC sampling methods which see use not just in the fields of machine learning and Bayesian statistics, but also in the fields of computational physics, biology, and computational linguistics. Indeed, MCMC can be used a universal inference method that is agnostic to the underlying likelihood function and prior distribution. As a result, it already forms the basis of many state-of-the-art probabilistic programming frameworks [Pré03].

Furthermore, Firefly MC addresses the issue of tractable computations at scale. With increasingly larger dataset sizes across all research and industry domains, scalability has become an increasingly central concern and proven tools such as MCMC have recently begun to receive much attention [KCW13].

2 Theory

Firefly Monte Carly (FlyMC) is a method which addresses the computational problem of evaluating $L(\theta)$ for a large number of data points N by using exact Monte Carlo estimates of $L(\theta)$ computed over smaller, tractable subsets. It works by augmenting the joint distribution with hidden *brightness* variables $Z = \{z_n \in \{0,1\}\}_{n=1}^{N}$to form a *complete data likelihood*

$$P(X, Z|\theta) = \prod_{n=1}^{N} P(x_n|\theta)P(z_n|x_n, \theta)$$

Like other auxiliary variable methods (e.g. Hamiltonian MC [BGJM11]), we can recover the desired likelihoods simply through marginalization

$$\sum_{Z} P(X, Z|\theta) = \sum_{z_1} \cdots \sum_{z_N} \prod_{n=1}^{N} P(|x_n|\theta)P(z_n|x_n, \theta) = \prod_{n=1}^{N} P(|x_n|\theta) \sum_{z_n} P(z_n|x_n, \theta) = L(\theta)$$

In FlyMC, the brightness variables are chosen to have Bernoulli distribution

$$P(z_n|x_n, \theta) = \left(\frac{L_n(\theta) - B_n(\theta)}{L_n(\theta)}\right)^{z_n} \left(\frac{B_n(\theta)}{L_n(\theta)}\right)^{1-z_n} \tag{2}$$

where$B_n(\theta)$ is a lower bound on the likelihood satisfying

$$0 \leq B_n(\theta) \leq L_n(\theta) \tag{3}$$

$$\prod_{n \in A} B_n(\theta) \text{ can be efficiently computed for any data subset } A \subset X \tag{4}$$

Equation 3 is required for Equation 2 to be a valid probability distribution. To see why Equation 4 is necessary, consider computing the complete data likelihood

$$\begin{aligned} P(X, Z|\theta) &= \prod_{n=1}^{N} P(x_n|\theta)P(z_n|x_n, \theta) \\ &= \prod_{n=1}^{N} L_n(\theta) \left(\frac{L_n(\theta) - B_n(\theta)}{L_n(\theta)}\right)^{z_n} \left(\frac{B_n(\theta)}{L_n(\theta)}\right)^{1-z_n} \\ &= \prod_{x_n \in X_{bright}} (L_n(\theta) - B_n(\theta)) \prod_{x_n \in X_{dim}} (B_n(\theta)) \\ &= \prod_{n=1}^{N} B_n(\theta) \prod_{x_n \in X_{bright}} \frac{L_n(\theta) - B_n(\theta)}{B_n(\theta)} \end{aligned} \tag{5}$$

where we have defined the bright points $X_{bright} = \{x_n : z_n = 1\}$ and dim points $X_{dim} = \{x_n : z_n = 0\}$. As a consequence of our choice for Equation 2, Equation 5 decomposes into two products where $\frac{L_n(\theta) - B_n(\theta)}{B_n(\theta)}$ only needs to be evaluated for $x_n \in X_{bright}$. For illustration, we plot the bright/dim decomposition of $L_n(\theta)$ in Figure 1, as well as show an example (θ, z_n) trajectory.

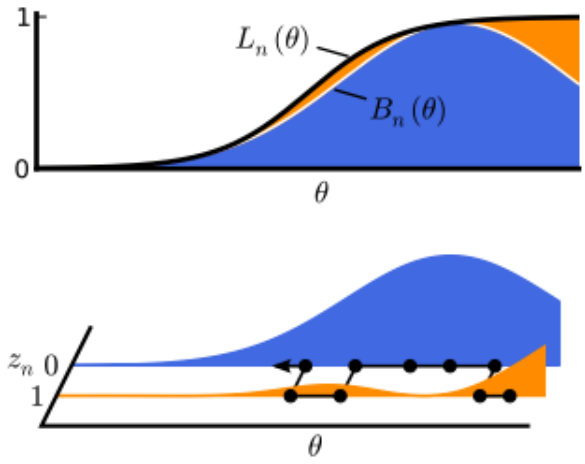


Figure 1: Graphical representation of the decomposition of L_n into a $L_n - B_n$ part associated with bright variables $z_n = 0$ and a B_n part associated with dim variables $z_n = 0$ [MA14]

Equation 5 also illustrates the payoff from assuming Equation 4: by assuming $\prod_{n=1}^{N} B_n(\theta)$ term in Equation 5 is efficient, the overall computational complexity of evaluating the complete likelihood $P(X, Z|\theta)$ is determined by $\#X_{bright}$.

2.1 Markov chain implementation

Samples $(\theta, \vec{z}) \sim (P(\theta|X), P(Z|X, \theta))$ can be generated using Gibbs sampling because of iid assumptions among the X and lack of interdependence among the Z. Given current $\theta^{(t)}$ and $\vec{z}^{(t)}$, we first fix the parameters θ and sample the brightness variables z_n with Bernoulli probability $\frac{L_n(\theta)-B_n(\theta)}{L_n(\theta)}$. Then, the brightnesses z_n are fixed and the Markov chain is stepped using Metropolis-Hastings [CG95] or more complex steppers such as Metropolis-adjusted Langevin dynamics [RS02] when the likelihood is differentiable.

The resampling of the brightness variables at each iteration cause each data point to "blink" bright and dim across iterations, inspiring the name "Firefly" given to this method.

2.1.1 Implicit sampling

One possible criticism of the method described above is that direct sampling of z_n from N Bernoulli distributions, each with probability $\frac{L_n(\theta)-B_n(\theta)}{L_n(\theta)}$, would again require N evaluations of $L_n(\theta)$. This appears to mitigate any savings obtained by FlyMC.

Fortunately, this can be mitigated through implicit sampling. Instead of recomputing Bernoulli probabilities and directly sampling the N z_ns, a second MH-MCMC is used for sampling brightnesses. The motivation is that many $z_i = 0$ ideally, hence it makes sense to only consider proposals where each dim point has a small $q_{d\to b} \in [0, 1]$ probability of becoming bright. As computing the acceptance probability requires likelihoods only need to be evaluated for the brightness variables proposed to change state, this could significantly reduce the number of likelihood evaluations if few brightness variables change state across successive Gibbs iterations.

2.2 Theoretical performance analysis

FlyMC's speedup over a traditional $O(N)$ likelihood evaluation is $\frac{N}{\#X_{bright}}$ per iteration. However, since the brightness variables $z_n$are themselves, to understand the performance improvements we consider the expected number of bright points $E[\#X_{bright}]$.

Note that for any point x_n, Equation 2 shows that the probability x_n is bright is given by $\frac{L_n(\theta)-B_n(\theta)}{L_n(\theta)}$. Hence,

the number of bright points is given by

$$E[\#X_{bright}] = \sum_{n=1}^{N} E[z_n] = \sum_{n=1}^{N} \int P(\theta|X) \frac{L_n(\theta) - B_n(\theta)}{L_n(\theta)} d\theta \tag{6}$$

This says that the number of bright points (and hence the speedup achieved by FlyMC) is related to the gap between the true likelihood $L_n(\theta)$ and the lower bound $B_n(\theta)$, weighted by the posterior $P(\theta|X)$.

This result makes intuitive sense. Consider the extreme case where for all n: $B_n(\theta) = 0$. Then $P(z_n = 1|x_n, \theta) = 1$ so every data point is bright (i.e. $X = X_{bright}$) and Equation 5 reduces to evaluating the full dataset likelihood $\prod_{n=1}^{N} L_n(\theta)$ as usual.

In the other extreme, if $B_n(\theta) = L_n(\theta)$ so all data points are dim. At first it appears that FlyMC counter-intuitively offers a $\frac{N}{\#X_{bright}} = \frac{N}{0} = \infty$speedup. However, considering Equation 5 reveals that what is computed is $\prod_{n=1}^{N} B_n(\theta) = \prod_{n=1}^{N} L_n(\theta)$, which by assumption Equation 4can be computed efficiently. But then computing $L(\theta) = \prod_{n=1}^{N} L_n(\theta)$ is efficient and hence is no longer $O(N)$, contradicting our earlier assumptions.

2.3 Discussion on the lower bound $B_n(\theta)$

The integral role played by the lower bound $B_n(\theta)$ makes it an important topic of investigation. In this section, we investigate how reasonable and justified are the assumptions required for developing FlyMC's theory.

One of our assumptions (Equation 4) was that computation of $\prod_{n=1}^{N} B_n(\theta)$ be efficient new θ. For example, choosing $B_n(\theta)$ to be exponential family will make computing $\prod_{n=1}^{N} B_n(\theta)$ for each new θ be $O(1)$ because

$$B_n(\theta) \propto \exp\left(\langle \eta_n, T(\theta) \rangle\right)$$
$$\prod_{n=1}^{N} B_n(\theta) \propto \exp\left(\left\langle \sum_{n=1}^{N} \eta_n, T(\theta) \right\rangle\right)$$

and $\sum_{n=1}^{N} \eta_n$ can be evaluated once and cached. We believe that this assumption is quite restrictive and have had some difficulty coming up with functional forms which are not exponential family that satisfy this assumption.

The other assumption (Equation 3) required $B_n(\theta)$ be a non-negative lower bound for $L_n(\theta)$. This is trivially satisfied by setting $B_n(\theta) = 0$, but to be practical we should try to find a lower bound with the smallest gap possible. Several forms of bounds exist for commonly used likelihoods; the original authors consider the Jaakola and Jordan bound on the log logistic function. However, this requirement is quite restrictive and unrealistic for more complex use cases. For example, if $L_n(\theta)$ is given by some complicated Bayes net or neural network, then a valid $B_n(\theta)$ for use with FlyMC is not immediately obvious.

2.3.1 MAP-tuning the lower bound

In certain cases (e.g. Jaakola and Jordan bound on log logistic), B_n is part of a parametric family of lower bounds $\{B_n(\cdot, \xi)\}_\xi$where ξ is a free parameter. Since the number of bright points determines the speedup offered by FlyMC, Equation 6 suggests that we should choose ξ to minimize $E\left[\frac{L_n(\theta) - B_n(\theta)}{L_n(\theta)}\right]$ for each datapoint n where the expectation is taken over the posterior distribution $P(\theta|X)$.

The authors of FlyMC propose a *MAP-tuned FlyMC* variant where the posterior expectation $E\left[\frac{L_n(\theta) - B_n(\theta)}{L_n(\theta)}\right]$ is replaced with the MAP point estimate $\frac{L_n(\theta_{MAP}) - B_n(\theta_{MAP})}{L_n(\theta_{MAP})}$ and $\xi = \text{argmin}_\xi L_n(\theta_{MAP}) - B_n(\theta_{MAP}).\theta_{MAP} = \text{argmax}_\theta P(\theta|X)$ can be found using stochastic gradient descent to handle large data-set concerns.

Our opinions about this method are mixed. While MAP tuning should provide a tighter lower bound than no tuning, approximating the posterior with a delta distribution centered at θ_{MAP} seems suspicious. Other methods which could be considered here include first generating some posterior samples using an untuned chain or MCMC on a subset of the data, then tuning ξ over the posterior samples.

3 Implementation

FlyMC augments traditional MCMC by introducing brightness variables Z which are Gibbs sampled along with the original chain's state θ. At a high level, the algorithm consists of the following steps:

1. Initialize $\theta^{(0)}$

2. For $t = 1, \cdots, N_{iter}$

   (a) For $n = 1, \cdots, N$: sample $z_n^{(t)} \sim \text{Bernoulli}\left(\frac{L_n(\theta^{(t)}) - B_n(\theta^{(t)})}{L_n(\theta^{(t)})}\right)$ with $\theta = \theta^{(t)}$fixed

3. Propose $\theta^{(t+1)} \sim q(\theta^{(t+1)}|\theta^{(t)})$

4. Compute acceptance probability $A(\theta^{(t+1)}|\theta^{(t)}) = \min\left\{1, \frac{q(\theta^{(t)}|\theta^{(t+1)})P(X,Z|\theta^{(t+1)})P(\theta^{(t+1)})}{q(\theta^{(t+1)}|\theta^{(t)})P(X,Z|\theta^{(t)})P(\theta^{(t)})}\right\}$

5. Repeat from step 2 until chain has mixed, yielding a single sample $z_n^{(\infty)}$

In step 4, the complete data likelihood is computed using Equation 5 and hence only requires evaluating the likelihood over X_{bright}.

3.1 Provided reference software

Reference code is provided by the authors[1], which provides implementations FlyMC for logistic regression using MH-MCMC, softmax using Langevin dynamics, and sparse linear regression using slice sampling. The reference implementation includes an implicit sampling scheme for sampling brightness variables z_i.

3.2 Our contributions to the project

However, the only full example provided is a logistic regression example on toy data and is insufficient for reproducing the results reported in the paper. We forked he authors' code [2] and added our own modifications, most importantly:

1. Support for MAP-tuning of the lower bounds, using scipy.optimize for MAP estimation
2. Implementation of regular full-dataset MCMC, which is achieved by fixing $\forall i : z_i = 1$
3. Improved numerical overflow handling in `LogisticModel`
4. Instrumentation for counting log-likelihood evaluations and outputting traces of chain state and log likelihood per iteration
5. Integration, parameter tuning, and reproduction of MNIST Logistic Regression, CIFAR-10 Softmax Classification, and Robust Linear Regression experimental results

We also fixed a significant bug in some caching code provided with the reference implementation and submitted our bugfix back upstream [3]. The problem was that computations of $\left\{\frac{L_n(\theta) - B_n(\theta)}{B_n(\theta)} : n \in idx\right\}$were being cached for vectors θ and idx, but that caching was performed over the entire idx vector rather than elementwise. For example, if $idx2 \subsetneq idx$, then even though $\left\{\frac{L_n(\theta) - B_n(\theta)}{B_n(\theta)} : n \in idx2\right\} \subset \left\{\frac{L_n(\theta) - B_n(\theta)}{B_n(\theta)} : n \in idx\right\}$ the reference implementation would still result in a cache miss because $idx2 \neq idx$. This bug causes the number of likelihood evaluations to be twice the number of proposed bright points $\#X_{bright}$at each iteration, which means that FlyMC will only offer speedups if $\#X_{bright} < \frac{N}{2}$.

4 Reproduction experiments

We conducted the following three experiments:

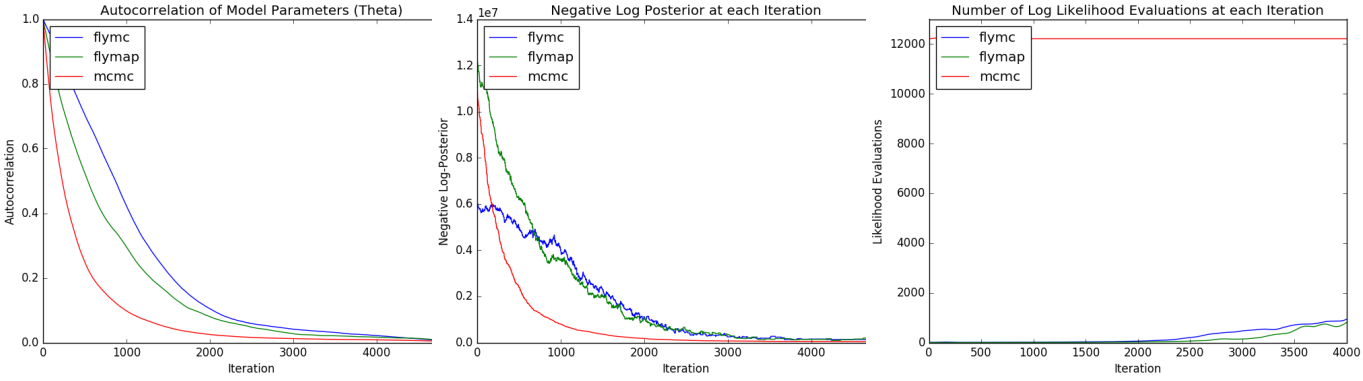
- Logistic regression: For this particular task, we tested we tested FlyMC's performance for the task of classifying 7s and 9s on MNIST, with prior $P(\theta) \sim \mathcal{N}(0, I)$.
- Softmax classification: For this task, we tested FlyMC's ability to fit a softmax classification model on three CIFAR-10 classes (airplane, automobile and bird) using Langevin dynamics
- Robust linear regression: For logistic regression task, we generated synthetic data and evaluated the performance of FlyMC in comparison to regular MCMC.

The first two of these three experiments were data-sets used in the paper. However, the OPV dataset used in the robust linear regression was not publicly available in a convenient format. This also gave us an opportunity to explore FlyMC's performance fitting a model against data where the true model is known.

[1] https://github.com/HIPS/firefly-monte-carlo
[2] https://github.com/feynmanliang/firefly-monte-carlo
[3] https://github.com/HIPS/firefly-monte-carlo/pull/1

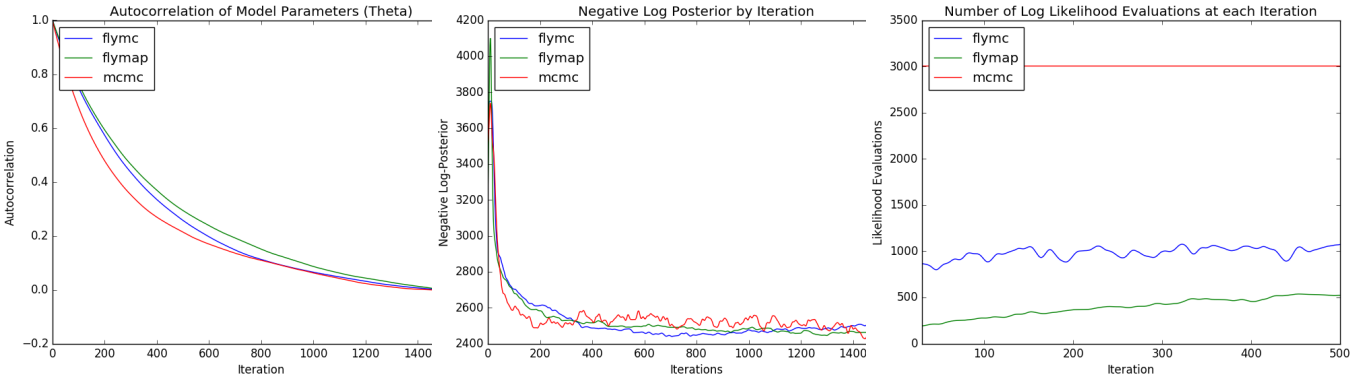
4.1 Results of the three methods

In this section we present the results of our experiments running MCMC (regular full-dataset MCMC), FlyMC (untuned Firefly), and FlyMAP (MAP-tuned Firefly) on the tasks described above.



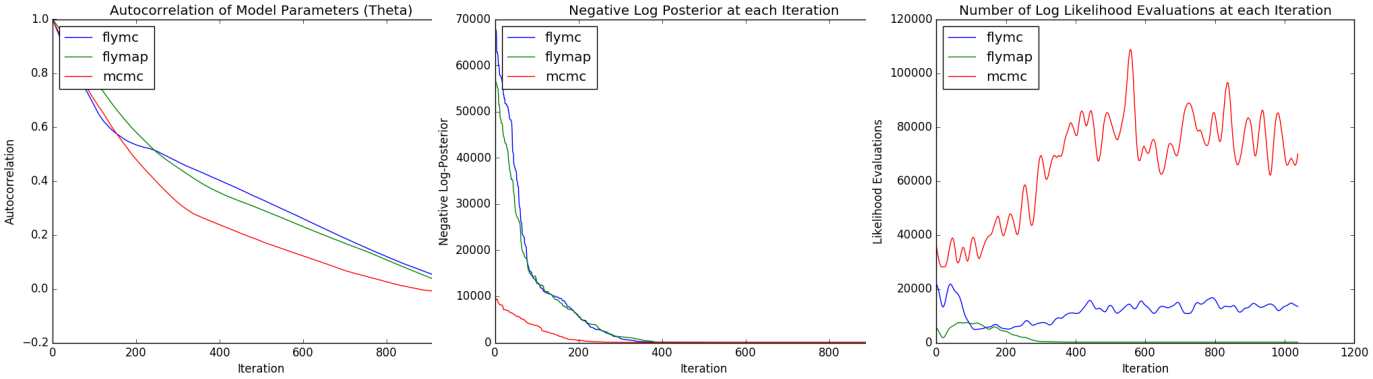
(a) Autocorrelation (b) Negative log posterior (c) Number likelihood evaluations

Figure 2: Logistic regression binary classification on MNIST with MH-MCMC



(a) Autocorrelation (b) Negative log posterior (c) Number likelihood evaluations

Figure 3: Softmax 3-way classification on CIFAR-10 with Langevin dynamics



(a) Autocorrelation (b) Negative log posterior (c) Number likelihood evaluations

Figure 4: Robust regression on synthesized data with slice sampler

From figure (c), we can see that the number of log-likelihood evaluations at a given iteration is constant under the MH-MCMC sampler for Logistic regression on MNIST. Approximately, 12,000 evaluations are computed at each iteration. For soft-max classification, figure (c) , this number of evaluations is also constant at 3000. In contrast, far fewer likelihood evaluations are required under both the FlyMC and FlyMAP models. In figure (c) we can see

that both FlyMAP and FlyMC compute less than a 12th of the likelihood computations evaluated by the MCMC sampler. The trend is less marked for both soft-max classification and robust regression as can be seen in the respective figures (c), where FlyMC and FlyMAP seem to return more modest reductions of around 1/3 and 1/6 of the likelihoods respectively.

If we had reached the stationary distribution of a Markov Chain, this would represent a significant reduction in the likelihoods that one would need to compute. However, to an extent FlyMAP's benefits from reduced likelihood computations will be mitigated by a slower burn in period and also more dependent samples. This means that the sampling chains for FlyMC and FlyMAP would have to be run for longer to generate accurate samples.

The plots for the negative log posterior (b) gives some indication of the burn-in period, since log-likelihoods for the sampled parameters will only start to stop dropping, when the stationary distribution of the Markov Chain has been reached. For logistic regression, it seems as though these likelihoods that are reached at the 2000th iteration are only reached at the 4000th iteration for both FlyMC and FlyMAP, which seems to suggest a burn in that is around twice as slow as regular MCMC. For logistic regression, a reasonable burn in period would be 200 iterations, whereas it seems as though 400 iterations would do better justice to the Firefly Monte Carlo methods. For robust regression, these figures seem to be 200 and 350 respectively for MCMC and FlyMC/FlyMAP.

Plots of the auto-correlations can further complement the picture presented by the log-likelihood plots, since they can give an idea of the spacing between dependent samples, which can also approximate for a particular burn in period. The auto-correlation plots confirm our suspicion that fire-fly MC methods produce samples that tend to be more dependent, however they do little to arbitrate int he dispute of what term will dominate when performing FlyMC: fewer likelihood evaluations per iteration or greater burn-in periods. So far, it seems as though the likelihood reductions are the dominant term in these computations.

To summarize, we found that Firefly reduced likelihood evaluations but led to slower mixing times as indicated by autocorrelations across longer time scales. Furthermore, the impact of MAP-tuning is strongly evident with FlyMAP oftentimes requiring far less evaluations. These trends were also found in the paper [MA14] we are trying to replicate.

4.2 Quantifying mixing rates and assessing relative speedup

Our results have shown that FlyMC decreases the number of likelihood evaluations required, but also leads to more correlated samples and hence slower mixing times. To determine whether FlyMC offers any speedup, it is necessary to quantify the slowdown in mixing due to using data subsets.

Following [CC96], one method for measuring mixing rates is effective sample size. Table 1 reproduces Table 1 from [MA14], comparing the average likelihood queries per iteration and the effective sample size (ESS) per 1000 iterations to determine relative speedups.

Table 1: Reproduction of the experiments reported in [MA14]. We did not reproduce the robust regression with slice sampling experiments because the OPV dataset used by the authors is not available in an easily consumable format.

Data Set	Algorithm	Average likelihood queries per iteration	ESS per 1000 iterations	Relative Speedup
Logistic Regression	Regular MCMC	12214	3.6	(1)
	Untuned FlyMC	6252	0.5	0.271
	MAP-Tuned FlyMC	207	0.6	9.834
Softmax classification	Regular MCMC	18000	7.1	(1)
	Untuned FlyMC	7997	4.0	1.268
	MAP-Tuned FlyMC	662	3.3	12.638
Robust Regression	Regular MCMC	NA	NA	NA
	Untuned FlyMC	NA	NA	NA
	MAP-Tuned FlyMC	NA	NA	NA

Our results show that MAP-tuned FlyMC performs well and achieves roughly an order of magnitude relative speedup in all cases. Untuned FlyMC, however, can be as much as 5× slower. As evidenced by the increased number of likelihood queries, this slowdown is caused by increased likelihood evaluations while attaining the same ESS. We suspect the underlying cause is due to the larger gap in the untuned lower bound, which following Equation 6 leads to more bright points.

5 Conclusion

From this practical, we saw how Firefly MC introduces auxiliary "brightness" variables to control the computational complexity of each MC iteration while still maintaining an exact estimator for the full likelihood. We saw that the auxiliary variables were defined using a lower bound $B_n(\theta)$ who's tightness to $L_n(\theta)$ is inversely proportional to the probability of a point being bright. We saw MAP tuning as one way to choose lower bounds minimize this gap, which in turn minimizes the number of bright points and hence maximizes the speedups offered by FlyMC.

Our experimental results from section 4 yielded similar trends on all data sets, tasks, and Markov chain steppers (e.g. MH, Langevin, slice sampler) considered. In all cases, using FlyMC results in a decrease in the average number of likelihood evaluations due to how the brightness probabilities are defined in Equation 2. At the same time, ESS also decreases because subsets rather than the full dataset are used for computing acceptance probabilities.

Comparing the ratio of ESS over average likelihood queries can give us a sense of relative speedup over regular MCMC accounting for the longer burn in time. According to Table 1, we see that untuned FlyMC may perform slightly better than regular MCMC but may also be relatively slower. In contrast, the MAP tuned FlyMC algorithm consistently achieves speedups $\approx 10\times$. This highlights the significant impact of the lower bound's gap on performance and emphasizes the importance of choosing the lower bound properly. Our results in Table 1 are very close to those reported in Table 1 of [MA14], indicating that we were successful in replicating the results reported in the original paper.

Through completing this practical, we have made a number of contributions (Github pull requests) back upstream to Harvard Intelligent Probabilistic Systems Group's code repository which address several important bugs and optimizations. In particular, we optimize the likelihood caches to work elementwise, fix overflow prevention in various areas of the code, and provide integration for MAP-tuning using `scipy.optimize`.

Many extensions of the work we have presented are possible. One particularly important experiment would be to tune the lower bounds over multiple parameter values sampled from the posterior rather than the single MAP estimate. This would ensure a low number of expected bright points, even if we are possibly far away from the MAP estimate. Given the significant performance difference between MAP-tuned and untuned FlyMC, we expect tuning lower bounds in this manner to yield even lower numbers of bright points and hence greater speedups over traditional MCMC.

Another direction for investigation is on the lower bounds B_n. The assumptions made in assumptions 1 and 2 on B_n are required for formulating FlyMC, and the authors do a good first pass discussion about using exponential families for B_n. However, it may be interesting to consider other functional forms for B_n such that $\prod_{n=1}^{N} B_n(\theta)$ is cheap to compute. Furthermore, it is often the case that we are working with likelihoods $L(\theta)$ which do not have a well-known lower bound (e.g. $L(\theta)$ could be parameterized by a deep neural network). In these situations, one would like any form of lower bound which is non-zero. Alternatively, one could investigate what happens when approximate bounds are used for $B_n(\theta)$ where cite the 0 <= Bn <= 1 does not hold for all θ.

Finally, we would like to mention that although the motivation for this work is computing on big datasets with large N, most experiments presented here and in the paper were small and run on a single machine. For FlyMC to truly be impactful in big data applications, an implementation on a distributed computing platform (e.g. Spark, Hadoop) which demonstrates tractable MCMC on large datacenter-scale datasets would be a convincing and immediately useful line of work.

References

[BGJM11] Steve Brooks, Andrew Gelman, Galin Jones, and Xiao-Li Meng. *Handbook of Markov Chain Monte Carlo*. CRC press, 2011.

[BR98] Stephen P Brooks and Gareth O Roberts. Convergence assessment techniques for markov chain monte carlo. *Statistics and Computing*, 8(4):319–335, 1998.

[CC96] Mary Kathryn Cowles and Bradley P Carlin. Markov chain monte carlo convergence diagnostics: a comparative review. *Journal of the American Statistical Association*, 91(434):883–904, 1996.

[CG95] Siddhartha Chib and Edward Greenberg. Understanding the metropolis-hastings algorithm. *The american statistician*, 49(4):327–335, 1995.

[Dia09] Persi Diaconis. The markov chain monte carlo revolution. *Bulletin of the American Mathematical Society*, 46(2):179–205, 2009.

[KCW13] Anoop Korattikara, Yutian Chen, and Max Welling. Austerity in mcmc land: Cutting the metropolis-hastings budget. *arXiv preprint arXiv:1304.5299*, 2013.

[MA14] Dougal Maclaurin and Ryan P Adams. Firefly monte carlo: Exact mcmc with subsets of data. *arXiv preprint arXiv:1403.5693*, 2014.

[Nea03] Radford M Neal. Slice sampling. *Annals of statistics*, pages 705–741, 2003.

[Pré03] András Prékopa. Probabilistic programming. *Handbooks in operations research and management science*, 10:267–351, 2003.

[RS02] Gareth O Roberts and Osnat Stramer. Langevin diffusions and metropolis-hastings algorithms. *Methodology and computing in applied probability*, 4(4):337–357, 2002.