Hamiltonian Monte Carlo within Stan

Daniel Lee
Columbia University, Statistics Department
bearlee@alum.mit.edu

BayesComp mc-stan.org
Why MCMC?

- Have data.
- Have a rich statistical model.
- No analytic solution.
- (Point estimate not adequate.)
Review: MCMC

- Markov Chain Monte Carlo. The samples form a Markov Chain.

- Markov property:
  \[ \Pr(\theta_{n+1} | \theta_1, \ldots, \theta_n) = \Pr(\theta_{n+1} | \theta_n) \]

- Invariant distribution:
  \[ \pi \times \Pr = \pi \]

- Detailed balance: sufficient condition:
  \[ \Pr(\theta_{n+1}, A) = \int_A q(\theta_{n+1}, \theta_n) dy \]
  \[ \pi(\theta_{n+1}) q(\theta_{n+1}, \theta_n) = \pi(\theta_n) q(\theta_n, \theta_{n+1}) \]
Review: RWMH

- Want: samples from posterior distribution:
  \( \Pr(\theta|x) \)

- Need: some function proportional to joint model.
  \( f(x, \theta) \propto \Pr(x, \theta) \)

- Algorithm:
  Given \( f(x, \theta), x, N, \Pr(\theta_{n+1} | \theta_n) \)
  For \( n = 1 \) to \( N \) do
    Sample \( \hat{\theta} \sim q(\hat{\theta} | \theta_{n-1}) \)
    With probability \( \alpha = \min \left( 1, \frac{f(x, \hat{\theta})}{f(x, \theta_{n-1})} \right) \), set \( \theta_n \leftarrow \hat{\theta} \), else \( \theta_n \leftarrow \theta_{n-1} \)
Review: Hamiltonian Dynamics

- (Implicit: \( d = \text{dimension} \))
- \( q = \text{position (}\!d\!)\text{-vector} \)
- \( p = \text{momentum (}\!d\!)\text{-vector} \)
- \( U(q) = \text{potential energy} \)
- \( K(p) = \text{kinetic energy} \)
- Hamiltonian system: \( H(q, p) = U(q) + K(p) \)
Review: Hamiltonian Dynamics

- for $i = 1, \ldots, d$
  \[ \frac{dq_i}{dt} = \frac{\partial H}{\partial p_i} \]
  \[ \frac{dp_i}{dt} = -\frac{\partial H}{\partial q_i} \]

- kinetic energy usually defined as $K(p) = p^T M^{-1} p / 2$

- for $i = 1, \ldots, d$
  \[ \frac{dq_i}{dt} = [M^{-1} p]_i \]
  \[ \frac{dp_i}{dt} = -\frac{\partial U}{\partial q_i} \]
Connection to MCMC

- $q$, position, is the vector of parameters
- $U(q)$, potential energy, is (proportional to) the minus the log probability density of the parameters
- $p$, momentum, are augmented variables
- $K(p)$, kinetic energy, is calculated
- Hamiltonian dynamics used to update $q$.
- Goal: create a Markov Chain such that $q_1, \ldots, q_n$ is drawn from the correct distribution
Hamiltonian Monte Carlo

- Algorithm:
  Given \( U(q) \propto -\log(Pr(q, x)) \), \( q_0 \), \( N \), time \((\epsilon, L)\)

For \( n = 1 \) to \( N \) do
  Sample \( p \sim N(0, 1) \)
  \( q_{\text{start}} \leftarrow q_{n-1}, \ p_{\text{start}} \leftarrow p \)
  Get \( q \) and \( p \) at time using Hamiltonian dynamics
  \( p \leftarrow -p \)
  With probability \( \alpha = \min(1, \exp(H(q,p) - H(q_{\text{start}}, p_{\text{start}}))) \),
    set \( q_n \leftarrow q \), else \( q_n \leftarrow q_{n-1} \).
Figure 2: Example of a trajectory generated during one iteration of NUTS. The blue ellipse is a contour of the target distribution, the black open circles are the positions $\theta$ traced out by the leapfrog integrator and associated with elements of the set of visited states $B$, the black solid circle is the starting position, the red solid circles are positions associated with states that must be excluded from the set $C$ of possible next samples because their joint probability is below the slice variable $u$, and the positions with a red "x" through them correspond to states that must be excluded from $C$ to satisfy detailed balance. The blue arrow is the vector from the positions associated with the leftmost to the rightmost leaf nodes in the rightmost height-3 subtree, and the magenta arrow is the (normalized) momentum vector at the final state in the trajectory. The doubling process stops here, since the blue and magenta arrows make an angle of more than 90 degrees. The crossed-out nodes with a red "x" are in the right half-tree, and must be ignored when choosing the next sample.

Being more complicated, the analogous algorithm that eliminates the slice variable seems empirically to be slightly less efficient than the algorithm presented in this paper.

- Blue ellipse is contour of target distribution
- Initial position at black solid circle
- Arrows indicate a U-turn in momentum
HMC Review

- Correct MCMC algorithm; satisfies detailed balance
- Use Hamiltonian dynamics to propose new point
- Metropolis adjustment accounts for simulation error
- Explores space more effectively than RMWH
- Difficulties in implementation
Nuances of HMC

• Simulating over discrete time steps: Error requires accept / reject step.
• leapfrog integrator. $\epsilon, L$.
• Need to tune amount of time.
• Negative momentum at end of trajectory for symmetric proposal.
• Need derivatives of $U(q)$ with respect to each $q_i$.
• Samples efficiently over unconstrained spaces. Needs continuity of $U(q)$. 
Stan’s solutions

- Autodiff: derivatives of $U(q) \propto -\log(\Pr(q, x))$.
- Transforms: taking constrained variables to unconstrained.
- Tuning parameters: No-U-Turn Sampler.
Different MCMC algorithms

Figure 7: Samples generated by random-walk Metropolis, Gibbs sampling, and NUTS. The plots compare 1,000 independent draws from a highly correlated 250-dimensional distribution (right) with 1,000,000 samples (thinned to 1,000 samples for display) generated by random-walk Metropolis (left), 1,000,000 samples (thinned to 1,000 samples for display) generated by Gibbs sampling (second from left), and 1,000 samples generated by NUTS (second from right). Only the first two dimensions are shown here.

4.4 Comparing the Efficiency of HMC and NUTS

Figure 6 compares the efficiency of HMC (with various simulation lengths $\lambda \approx \approx L$) and NUTS (which chooses simulation lengths automatically). The x-axis in each plot is the target $\delta$ used by the dual averaging algorithm from section 3.2 to automatically tune the step size $\gamma$. The y-axis is the effective sample size (ESS) generated by each sampler, normalized by the number of gradient evaluations used in generating the samples. HMC's best performance seems to occur around $\delta = 0.65$, suggesting that this is indeed a reasonable default value for a variety of problems. NUTS's best performance seems to occur around $\delta = 0.6$, but does not seem to depend strongly on $\delta$ within the range $\delta \in [0.45, 0.65]$. $\delta = 0.65$ therefore seems like a reasonable default value for NUTS.

On the two logistic regression problems NUTS is able to produce effectively independent samples about as efficiently as HMC can. On the multivariate normal and stochastic volatility problems, NUTS with $\delta = 0.6$ outperforms HMC's best ESS by about a factor of three.

As expected, HMC's performance degrades if an inappropriate simulation length is chosen. Across the four target distributions we tested, the best simulation lengths $\lambda$ for HMC varied by about a factor of 100, with the longest optimal $\lambda$ being 17.62 (for the multivariate normal) and the shortest optimal $\lambda$ being 0.17 (for the simple logistic regression). In practice, finding a good simulation length for HMC will usually require some number of preliminary runs. The results in Figure 6 suggest that NUTS can generate samples at least as efficiently as HMC, even discounting the cost of any preliminary runs needed to tune HMC's simulation length.
What is Stan?

1. Language for specifying statistical models
   \[ \Pr(\theta, X) \]

2. Fast implementation of statistical algorithms;
   interfaces from command line, R, Python, Matlab
What is Stan trying to solve?

- Stan: model fitting on arbitrary user model
- Stan: speed, speed, speed
- Team: easily implement statistics research
- Team: roll out stats algorithms
- User: easy specification of model
- User: trivial to change model
- User: latest and greatest algorithms available
Language: applied Bayesian modeling

1. Design joint probability model for all quantities of interest including:
   - observable quantities (measurements)
   - unobservable quantities (model parameters or potentially observable quantities)

2. Calculate the posterior distribution of all unobserved quantities conditional on the observed quantities

3. Evaluate model fit
Language: features

- high level language; looks like stats; inspired by BUGS language
- Imperative declaration of log joint probability distribution
  \[ \log(\Pr(\theta, X)) \]
- Statically typed language
- Constrained data types
- Easy to change models!
- Vectorization, lots of functions, many distributions
Coin flip example

data {
  int<lower=0> N;
  int<lower=0,upper=1> y[N];
}
parameters {
  real<lower=0,upper=1> theta;
}
model {
  theta ~ beta(1,1);
  y ~ bernoulli(theta);
}
Language

- Discuss more later
- Data types
- Blocks
- Constraints and transforms
Implementation

- Stan v2.2.0. 2/14/2014.
- Stan v2.3.0. Will be out within a week.
- Stan written in templated C++. Model translated to C++.
- Algorithms:
  - MCMC: auto-tuned Hamiltonian Monte Carlo, no-U-turn Sampler (NUTS)
  - Optimization: BFGS
Stan Stats

- Team: ~12 members, distributed
- 4 Interfaces: CmdStan, RStan, PyStan, MStan
- 700+ on stan-users mailing list
- Actual number of users unknown
  - User manual: 6658 downloads since 2/14
  - PyStan: 1299 downloads in the last month
  - CmdStan / RStan / MStan: ?
- 75+ citations over 2 years
  - stats, astrophysics, political science
  - ecological forecasting: phycology, fishery
  - genetics, medical informatics