Hamiltonian Monte Carlo

Tin Huynh and Romain Drai

April 24, 2019

Introduction

Outline

Review

- **Markov Chain Monte Carlo (MCMC)** is a method for approximately sampling from a distribution p by defining a Markov chain which has p as a stationary distribution.
- **Metropolis Hastings:** is a very general recipe for finding such a Markov chain
  - Choose a proposal distribution.
  - Correct for the bias by stochastically accepting or rejecting the proposal.

Shorthcomings of Gibbs and MH

- Due to its random walk behavior, MH can't fully **explore** complex and high-dimensional distributions
  - It will get stuck to a small region
  - It might not even converge
- Example:
  - Exploring Complex distribution: MH v. Hamiltonian MC

Hamiltonian Monte Carlo

What is Hamiltonian Monte Carlo?

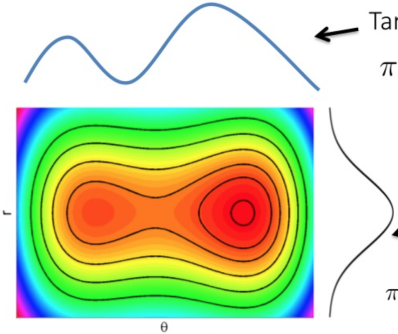
- **Hamiltonian Monte Carlo (HMC):**
  - MCMC algorithm which makes use of gradient information in order to avoid random walks and move more quickly toward regions of high probability.
  - HMC based on a discretization of Hamiltonian dynamics, with a Metropolis-Hastings accept/reject step to ensure that it has the right stationary distribution.
- Hamiltion Monte Carlo is also called **hybrid Monte Carlo** because it combines MCMC and deterministic simulation methods

Hamiltonian MC Algorithm

- **HMC**
  - borrows an idea from physics to suppress the local random walk behavior in the Metropolis algorithm
  - Allowing HMC to move much more rapidly through the target distribution.
  - For θ_i in target space, HMC adds momentum variable ϕ_i.
  - Both θ and ϕ are updated together in a new Metropolis algorithm, in which the jumping distribution for θ is determined largely by ϕ

HMC: Kinetic and Potential Energy

Example: Hamiltonian Monte Carlo (HMC)



Target posterior of θ

$$\pi(\theta) \triangleq p(\theta|\mathcal{D}) \propto \exp(-U(\theta))$$

Focus on $\theta \in \mathbb{R}^d$

Potential Energy

$U = -\log\left[p(\mathcal{D}|\theta)p(\theta)\right]$

Add auxiliary "momentum" variable r

$$\pi(\theta, r) \propto \exp\left(-U(\theta) - \underbrace{\frac{1}{2} r^T M^{-1} r}_{\text{Kinetic Energy}}\right)$$

$$= \exp(-H(\theta, r))$$

Hamiltonian

(total energy H)

Hamiltonian MC Algorithm

Notation

$p(\theta|y)$: posterior density

$p(\phi)$: Auxiliary momentum distribution

$$p(\theta, \phi|y) = p(\phi)p(\theta|y)$$

$$\phi \sim \text{MVN}(0, \Sigma)$$

$$\phi_i \sim N(0, \Sigma_{ii})$$

Σ : Usually set to be identity matrix(unit diagonal)

$$\frac{\partial \log p(\theta|y)}{\partial\theta} = \left(\frac{\partial \log p(\theta|y)}{\partial\theta_1}, ..., \frac{\partial \log p(\theta|y)}{\partial\theta_p}\right) \text{ (gradient)}$$

Hamiltonian MC Algorithm

Notation

$$\begin{aligned}
p(\phi,\theta) &: \text{defined a Hamiltonian}\\
H(\phi,\theta) &= -log(p(\theta))\\
&= -log(\phi|\theta) - log(p(\theta))\\
&= T(\phi|\theta) + V(\theta)\\
T(\phi|\theta) &= -log(\phi|\theta) \text{ ("Kinetic energy" term)}\\
V(\theta) &= -log(p(\theta)) \text{ ("potential energy" term)}
\end{aligned}$$

Hamiltonian MC Algorithm

- Properties of Hamiltonian Dynamics
    - Reversibility
    - Invariant
    - Volume preservation in (θ, ϕ) space
    - Symplecticness

Those properties are crucial to use in constructing MCMC updates

Hamiltonian MC Algorithm

Generating transitions

Start at current θ
First draw ϕ where: $\phi \sim MVN(0, \Sigma)$
Second update (θ, ϕ) via Hamilton's equation:

$$\frac{d\theta}{dt} = \frac{dH}{d\theta} = \frac{dT}{d\phi}$$
$$\frac{d\phi}{dt} = -\frac{dH}{d\theta} = -\frac{dT}{d\theta} - \frac{dV}{d\theta}$$

Hamiltonian MC Algorithm

Generating transitions

Since ϕ is independent to $p(\theta|y)$ so $p(\phi|\theta) = p(\phi)$, then:

$$\frac{d\theta}{dt} = \frac{dT}{d\theta}$$
$$\frac{d\phi}{dt} = -\frac{dV}{d\theta}$$

Discretizing Hamilton equation:

Euler's rule:

$$\phi_{t+\epsilon} = \phi_t + \epsilon \frac{d\phi}{dt}(t) = \phi_t - \epsilon \frac{dV}{d\theta}(\theta_t)$$
$$\theta_{t+\epsilon} = \theta_t + \epsilon \frac{d\theta}{dt}(t) = \theta_t + \epsilon \frac{\theta_t}{\sigma_{ii}}$$

Modification of Euler's rule:

$$\phi_{t+\epsilon} = \phi_t - \epsilon \frac{dV}{d\theta}(\theta_t)$$
$$\theta_{t+\epsilon} = \theta_t + \epsilon \frac{\theta_{t+\epsilon}}{\sigma_{ii}}$$

Hamiltonian MC Algorithm

HMC iteration (leapfrog step)

Step 1: updated ϕ with random draw from p(ϕ). Which is the same as prior distribution ϕ~ N(0,Σ)

Step 2: simultaneous update (θ, ϕ). This updated involved L (leapfrog steps)

- **(a)** $\phi = \phi + \frac{1}{2}\epsilon\frac{\partial \log p(\theta|y)}{\partial\theta}$
- **(b)** $\theta = \theta + \epsilon\Sigma^{-1}\phi$
- **(c)** $\phi = \phi + \frac{1}{2}\epsilon\frac{\partial \log p(\theta|y)}{\partial\theta}$

Hamiltonian MC Algorithm

HMC iteration (leapfrog step)

Step 3:
Compute

$$r = \frac{p(\theta^*|y)p(\phi^*)}{p(\theta^{t-1}|y)p(\phi^{t-1})}$$

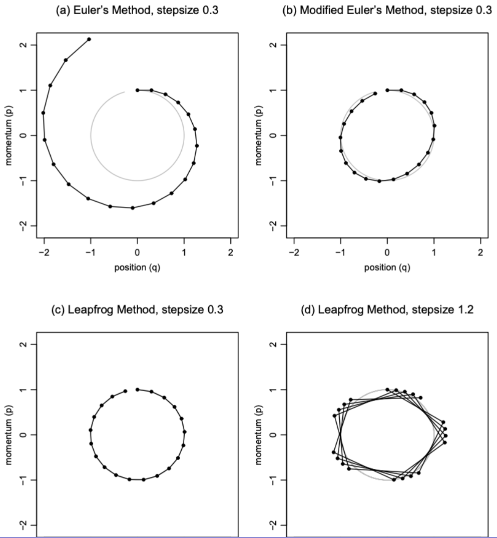
Then θ equals:

$$\theta^t = \theta^* \text{ with probability min(r,1)}$$
$$\theta^t = \theta^{t-1} \text{ o.w.}$$

HMC Algorithm Parameters

- HMC has 3 parameter:
  - **Discretization time ϵ:**
    - ϵ **is large:** Leapfrog interator will be inaccurate
    - ϵ **is small:** long simulation times
  - **number of step taken L:**
    - **L is large:** Algorithm will work too much for each iteration
    - **L is small:** the trajectory traced out in each iteration will be too short and sampling will devolve to a random walk
  - Σ **is poorly suited to the covariance:**
    - step size ϵ will have to decrease
    - number of step L will have to increase

Parameter Tuning



STAN

What is STAN?

- STAN is probabilistic progamming language used for Bayesian inference
  - Modeling
  - High Performance Computation
- Algorithms:
  - **MCMC sampler**
    - **NUTS, HMC**
  - Variational inference
    - ADVI
  - Optimization
    - L-BFGS

STAN file

A STAN file is composed of 7 programming blocks:

- data
- tranformed data
- parameters (required)
- transformed parameters
- functions
- model (required)
- generated quantities

How STAN implements HMC - Automation

Even a simplistic implementation of HMC requires extensise user input:

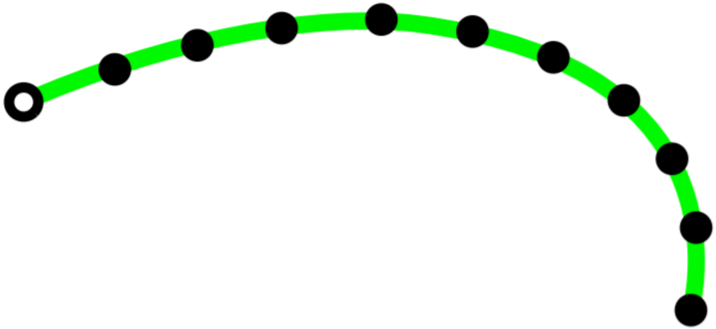
- Tuning the parameters:
    - **Discretization time ϵ:**
    - **number of step taken L:**
    - Σ
- Computing gradients:
    - $\frac{dV}{dt}$

STAN automates all those tasks

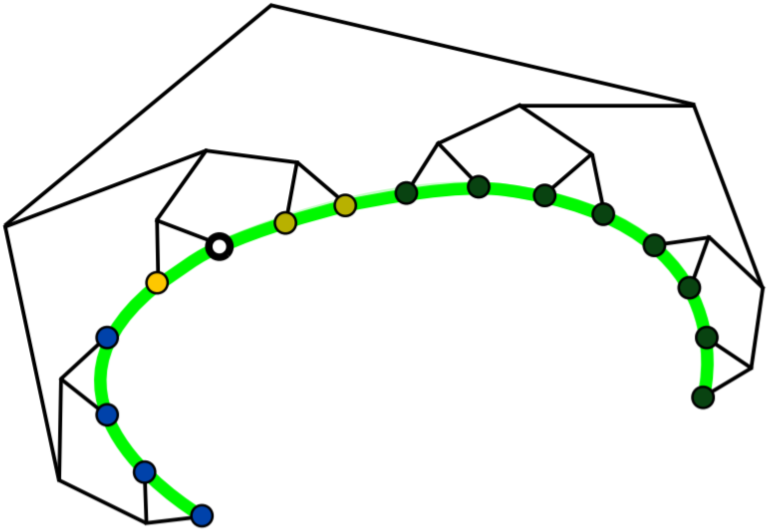
How STAN implements HMC - Locally adaptive

- Riemannian adaption:
  - allows sampler to explore the distribution more efficiently
  - mass matrix M adapts to the local curvature of $V(\theta)$
- NUTS - No U-Turn Sampler
  - increases computing efficiency
  - L is no longer fixed
  - Run trajectory until it turns around

NUTS



NUTS



Data example

Rats data

Figure 2: Rats

Rats Data

The data consists of:

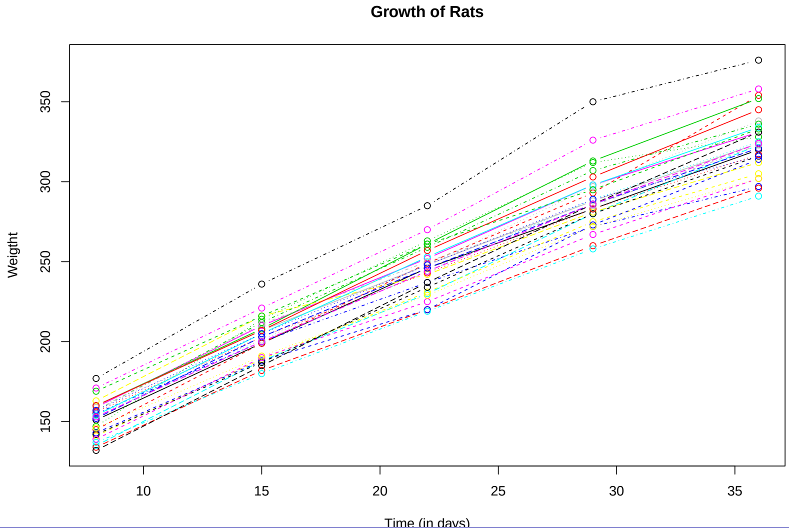
- 30 rats
- weighted weekly for 5 weeks

Notation:

- x_j age (in days)
- Y_{ij}: weight of i^{th} rat at time x_j

day8	day15	day22	day29	day36
151	199	246	283	320
145	199	249	293	354
147	214	263	312	328
155	200	237	272	297
135	188	230	280	323

Visualizing Data



Normal Hierarchical Model

- The model:
  - $Y_{ij} \sim N(\alpha_i + \beta_i(x_j - \bar{x}), \sigma^2)$
  - Priors:
    - $\alpha_i \sim N(\mu_\alpha, \sigma_\alpha^2)$
    - $\beta_i \sim N(\mu_\beta, \sigma_\beta^2)$
  - Hyperpriors:
    - μ_α, μ_β, σ_α^2, σ_β^2 and σ^2: independent 'non-informative' priors
  - $\bar{x} = 22$
- Variable of interest α_0
  - intercept when $x_j = 0$ (i.e birth)
  - $\alpha_0 = \mu_\alpha - \mu_\beta \bar{x}$

STAN: data block

The data block reads external information

```
data {
  int<lower=0> N;       // 2 primitive types
  int<lower=0> T;       // int and real
  real x[T];
  real y[N,T];
  real xbar;
}
```

STAN: parameter block

The parameters block defines the sampling space

```
parameters {
  real alpha[N];
  real beta[N];

  real mu_alpha;    // Prior
  real mu_beta;

  real<lower=0> sigmasq_y;
  real<lower=0> sigmasq_alpha;
  real<lower=0> sigmasq_beta;
}
```

STAN: transformed parameter block

Allows for parameter processing before the posterior is computed

```
transformed parameters {
  real<lower=0> sigma_y;
  real<lower=0> sigma_alpha;
  real<lower=0> sigma_beta;

  sigma_y = sqrt(sigmasq_y);
  sigma_alpha = sqrt(sigmasq_alpha);
  sigma_beta = sqrt(sigmasq_beta);
}
```

STAN: Model block

```
model {
  mu_alpha ~ normal(0, 100);
  mu_beta ~ normal(0, 100);
  sigmasq_y ~ inv_gamma(0.001, 0.001);
  sigmasq_alpha ~ inv_gamma(0.001, 0.001);
  sigmasq_beta ~ inv_gamma(0.001, 0.001);
  alpha ~ normal(mu_alpha, sigma_alpha); // vectorized
  beta ~ normal(mu_beta, sigma_beta);  // vectorized

  for (n in 1:N)
    for (t in 1:T)
      y[n,t] ~ normal(alpha[n] + beta[n] * (x[t] - xbar), s

}
```

STAN: Generated Quantities block

The generated quantity block allows for postprocessing

```
generated quantities {
  real alpha0;
  real y1_pred[T];

  alpha0 = mu_alpha - xbar * mu_beta;
  for (t in 1:T)
      y1_pred[t] = normal_rng(alpha[1] + beta[1] * (x[t] -
}
```

R file

```r
# Input data
y <- as.matrix(read.table('https://raw.github.com/wiki/star
x <- c(8, 15, 22, 29, 36)

rats_dat = list(y = y, x = x, xbar = mean(x), N <- nrow(y)

# Fit model using STAN
library(rstan)
options(mc.cores = parallel::detectCores())
rstan_options(auto_write = TRUE)
rats_fit <- stan('rats.stan', data = rats_dat,
                 chains = 4,
                 iter = 2000,
                 warmup = 1000,
                 thin = 1)
```

HMC speed

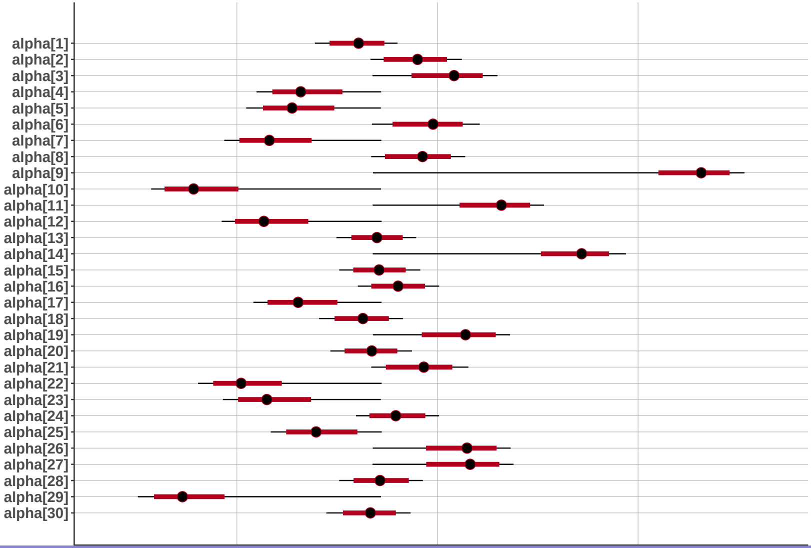
```r
print(get_elapsed_time(rats_fit))
```

```
##         warmup sample
## chain:1  1.000  1.067
## chain:2 16.234 12.202
## chain:3  2.144  0.513
## chain:4  1.439  0.398
```

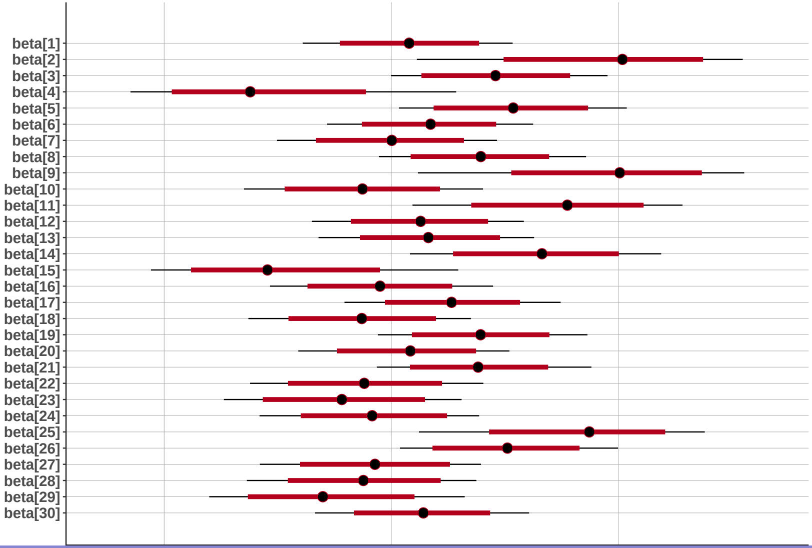
Summary Statistics

```
## Inference for Stan model: rats.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##                  mean se_mean    sd     10%     50%     90% n_eff Rhat
## mu_alpha       242.46    0.04  2.59  239.07  242.42  245.75  3799 1.00
## mu_beta          6.19    0.00  0.11    6.05    6.18    6.34   882 1.00
## sigmasq_y       55.55   20.37 61.60   30.90   37.47   51.77     9 1.66
## sigmasq_alpha  198.32   19.86 82.65  121.97  200.95  291.15    17 1.24
## sigmasq_beta     0.25    0.03  0.13    0.11    0.25    0.40    24 1.16
## sigma_y          6.92    0.92  2.78    5.56    6.12    7.20     9 1.66
## sigma_alpha     13.39    1.32  4.36   11.04   14.18   17.06    11 1.47
## sigma_beta       0.48    0.04  0.16    0.34    0.50    0.64    13 1.36
## alpha0         106.35    0.10  3.61  101.63  106.44  110.87  1230 1.00
## y1_pred[1]     154.90    0.14  8.37  144.89  154.75  164.62  3772 1.00
## y1_pred[2]     197.52    0.14  8.17  188.15  197.46  206.90  3482 1.00
## y1_pred[3]     240.05    0.13  7.95  230.89  239.91  249.26  3528 1.00
## y1_pred[4]     282.57    0.13  7.86  273.17  282.48  291.73  3710 1.00
## y1_pred[5]     325.05    0.17  8.65  314.79  325.00  335.25  2578 1.00
## lp__          -433.37    5.96 19.71 -447.32 -437.62 -425.17    11 1.46
##
## Samples were drawn using NUTS(diag_e) at Wed Apr 24 16:29:49 2019.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

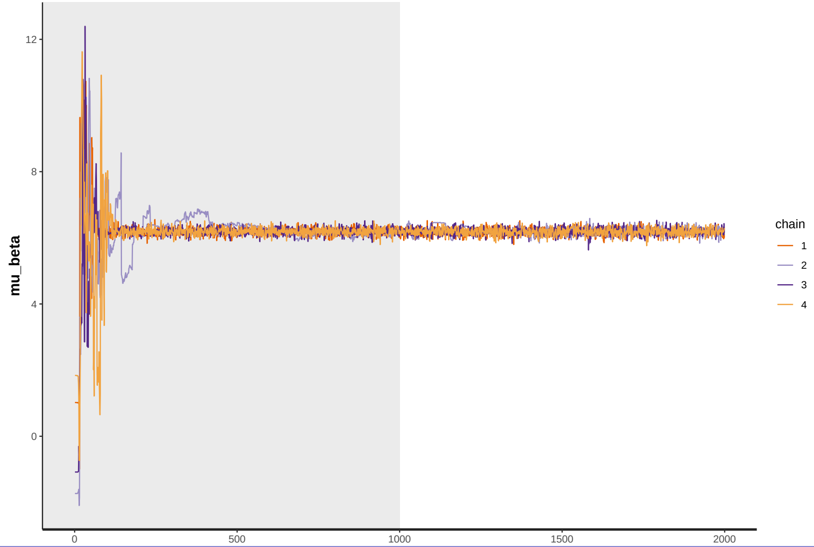
Alpha estimates



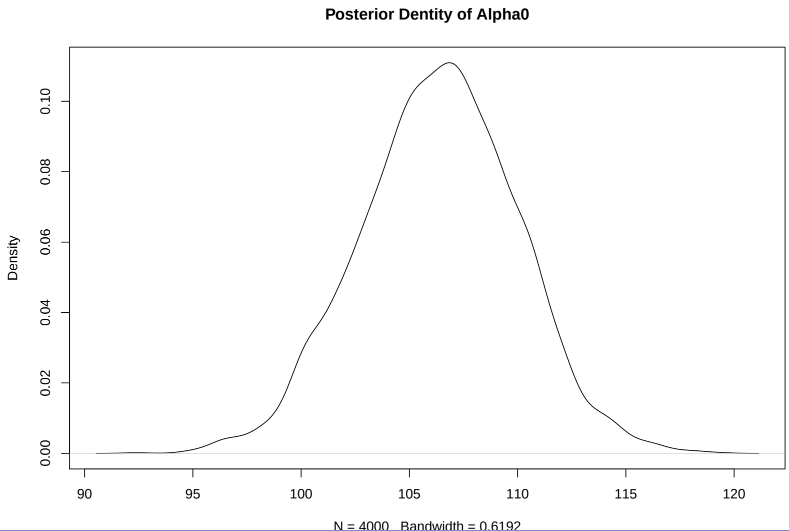
Beta estimates



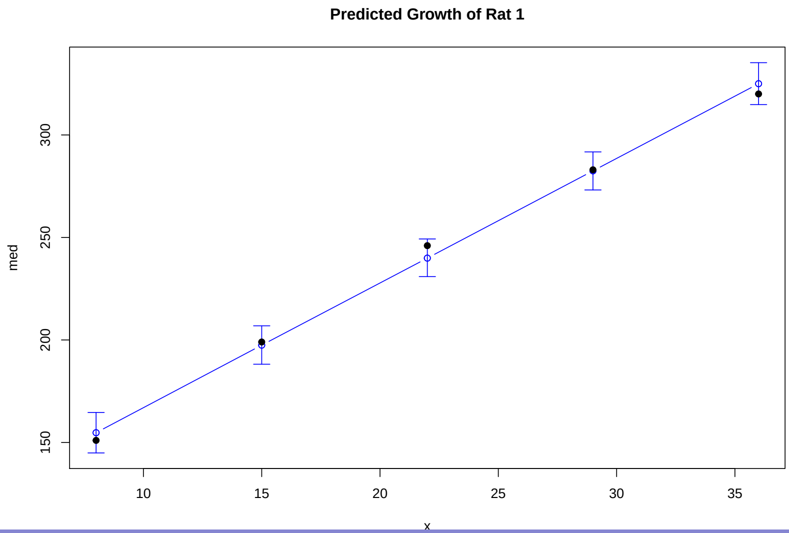
One trace plots among many



Weight at Birth



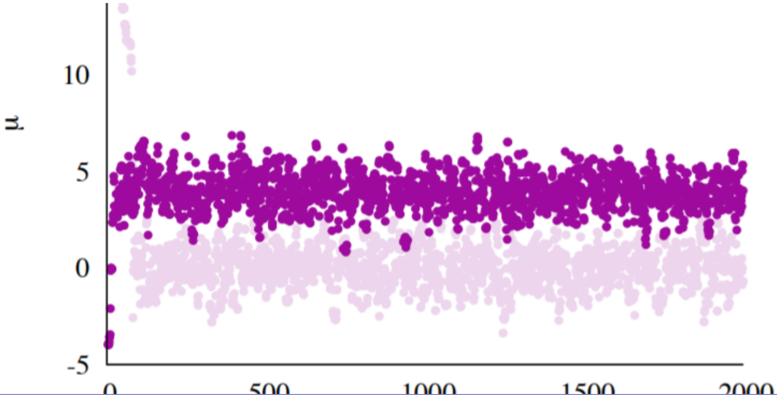
Model Checking: Predictive Sample



Conclusion

Why run Multiple Chains?

- Improve speed
  - Using multiple cores leads linear speed-up
- Add Robustness
  - Check for bad mixing or bad convergence



Multimodality

A sampler might get ‘stuck’ at one mode

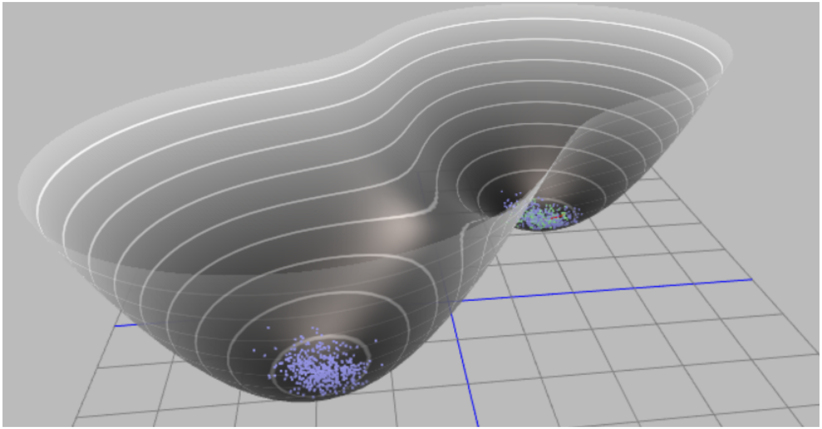


Figure 3: Multimodal distribution

Conclusion

- Compared to MH, Hamiltonian MC is better at:
    - **Exploring** complex and high-dimensional distributions
        - momentum function v. random walk
    - **Computating** efficiently:
        - less iterations
        - single iteration is more expensive
- HMC still has **difficulties** with:
    - distribution with **isolated modes**
    - distribution with very **short** or **long tails**
- HMC is an active research area
    - STAN is looking for developers!

THANK YOU!

Questions?