Chapter 12

- 12.1 Efficient Gibbs samplers (not part of the course)
- 12.2 Efficient Metropolis jump rules (not part of the course)
- 12.3 Further extensions to Gibbs and Metropolis (not part of the course)
- 12.4 Hamiltonian Monte Carlo (important)
- 12.5 Hamiltonian dynamics for a simple hierarchical model (useful example)
- 12.6 Stan: developing a computing environment (useful intro)

Extra material for HMC / NUTS

- An introduction for applied users with good visualizations:
  Cole C. Monnahan, James T. Thorson, and Trevor A. Branch (2016) Faster estimation of Bayesian models in ecology using Hamiltonian Monte Carlo. https://dx.doi.org/10.1111/2041-210X.12681
- A review of why HMC works:
  Radford Neal (2012). MCMC using Hamiltonian dynamics. https://arxiv.org/abs/1206.1901
- The No-U-Turn Sampler:
  Matthew D. Hoffman and Andrew Gelman (2014). The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo. https://jmlr.csail.mit.edu/papers/v15/hoffman14a.html
- Multinomial variant of NUTS:
  Michael Betancourt (2018). A Conceptual Introduction to Hamiltonian Monte Carlo. https://arxiv.org/abs/1701.02434

Extra material for Stan

- Andrew Gelman, Daniel Lee, and Jiqiang Guo (2015) Stan: A probabilistic programming language for Bayesian inference and optimization. http://www.stat.columbia.edu/~gelman/research/published/stan_jebs_2.pdf
- Carpenter et al (2017). Stan: A probabilistic programming language. Journal of Statistical Software 76(1). https://dox.doi.org/10.18637/jss.v076.i01
- Stan User's Guide, Language Reference Manual, and Language Function Reference (in html and pdf) https://mc-stan.org/users/documentation/
  - easiest to start from Example Models in User's guide
- Basics of Bayesian inference and Stan, part 1 Jonah Gabry & Lauren Kennedy (StanCon 2019 Helsinki tutorial)
  - https://www.youtube.com/watch?v=ZRpo41l02KQ&index=6&list=PLuwyh42iHquU4hUBQs20hkBsKSMrp6H0J
  - https://www.youtube.com/watch?v=6cc4N1vT8pk&index=7&list=PLuwyh42iHquU4hUBQs20hkBsKSMrp6H0J

Chapter 12 demos

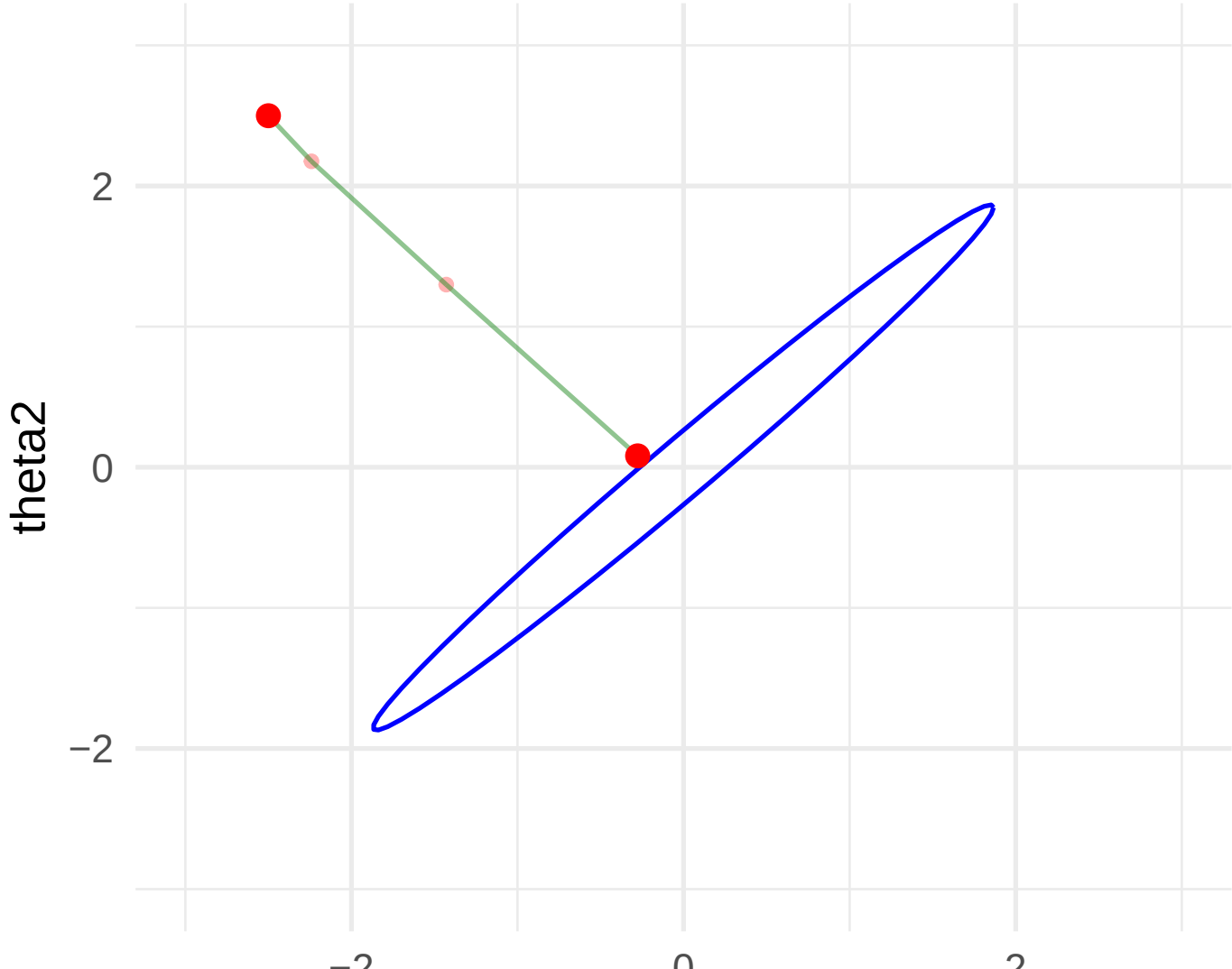
- demo12_1: HMC
- https://chi-feng.github.io/mcmc-demo/
- http://elevanth.org/blog/2017/11/28/build-a-better-markov-chain/
- cmdstanr_demo, rstan_demo
- http://sumsar.net/blog/2017/01/bayesian-computation-with-stan-and-farmer-jons/
- http://mc-stan.org/documentation/case-studies.html
- https://mc-stan.org/cmdstanr/
- https://mc-stan.org/rstan/

Hamiltonian Monte Carlo

- Uses log density (negative log density is called energy)
- Uses gradient of log density for more efficient sampling
- Augments parameter space with momentum variables

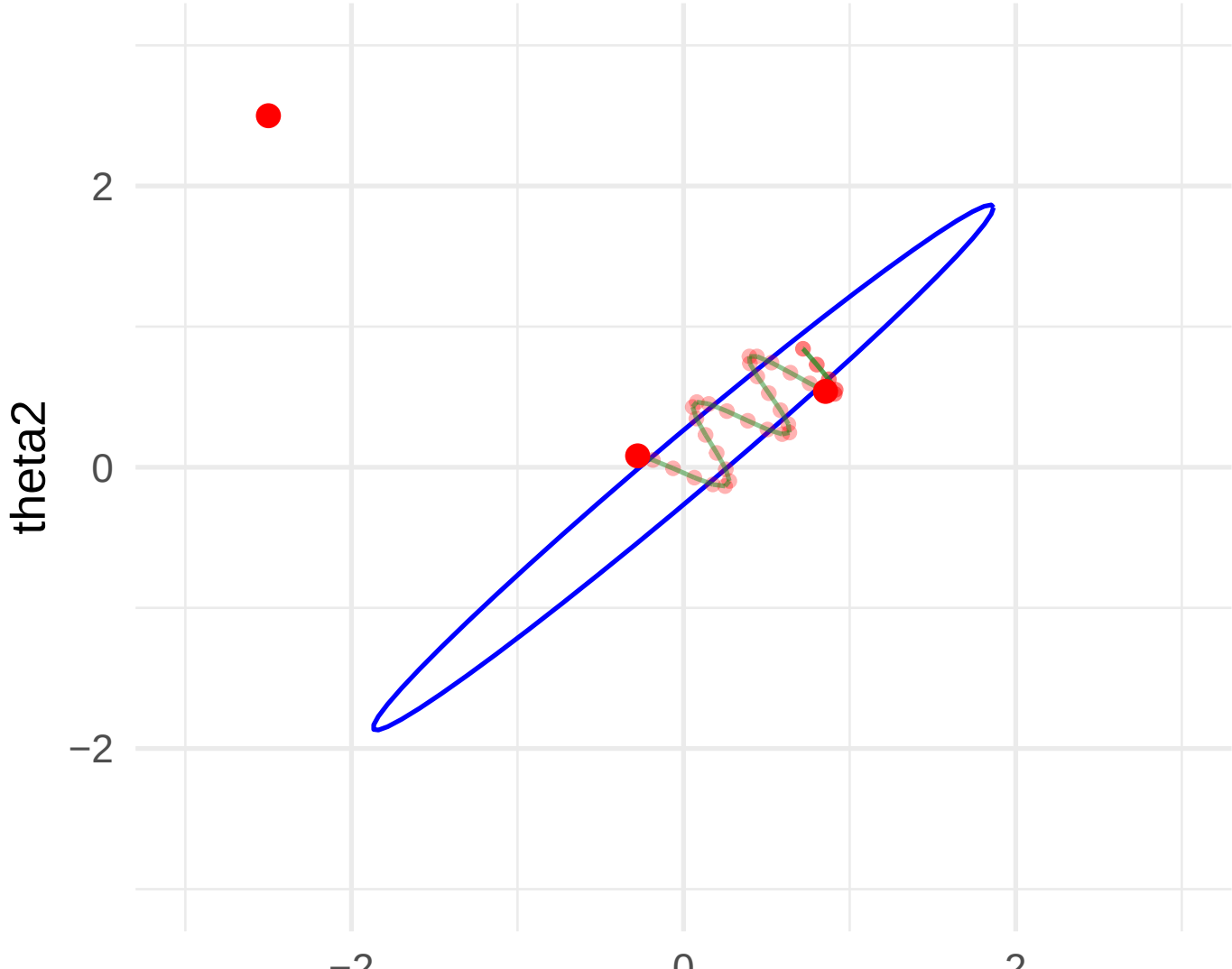
Hamiltonian Monte Carlo

- Uses log density (negative log density is called energy)
- Uses gradient of log density for more efficient sampling
- Augments parameter space with momentum variables



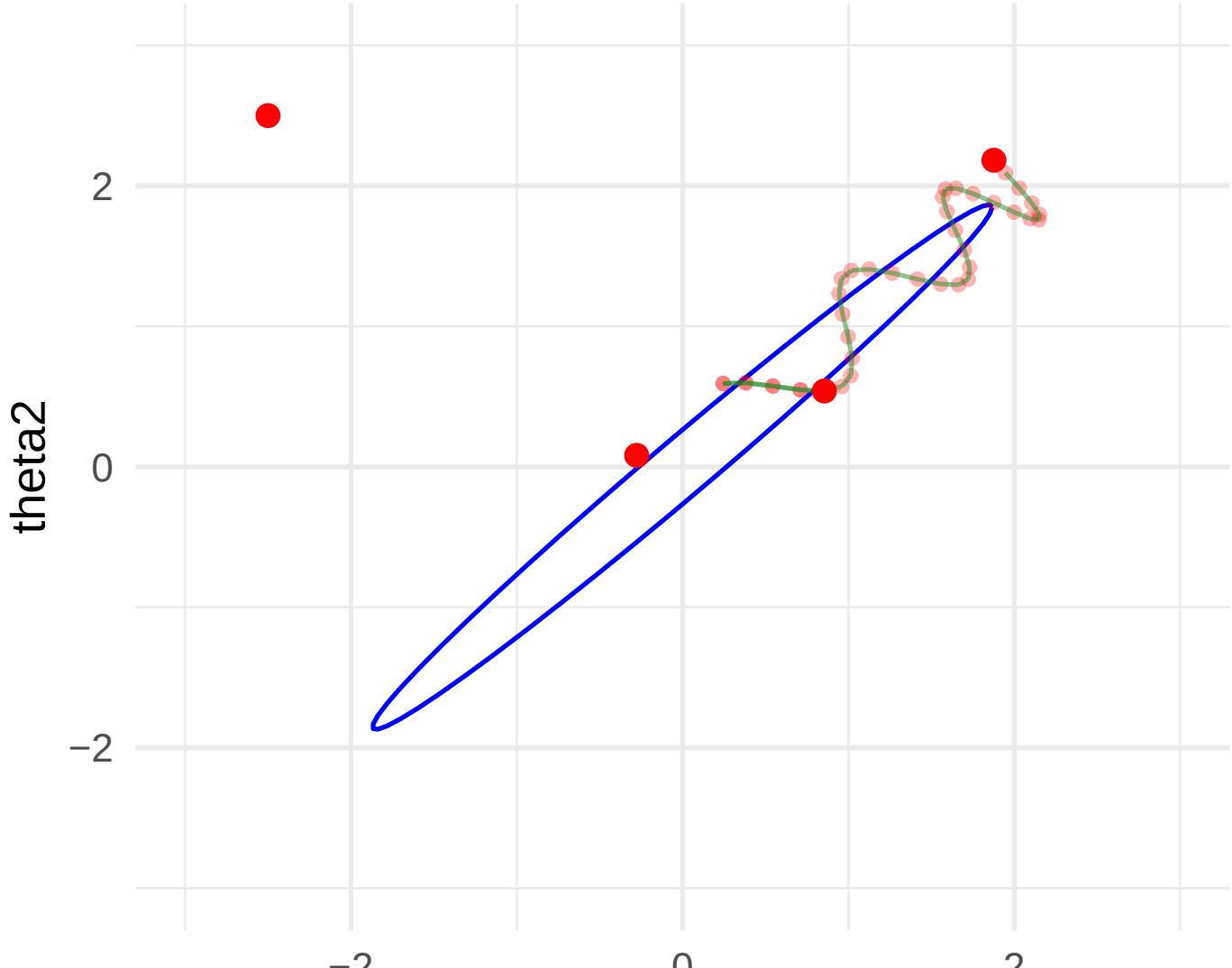
Hamiltonian Monte Carlo

- Uses log density (negative log density is called energy)
- Uses gradient of log density for more efficient sampling
- Augments parameter space with momentum variables



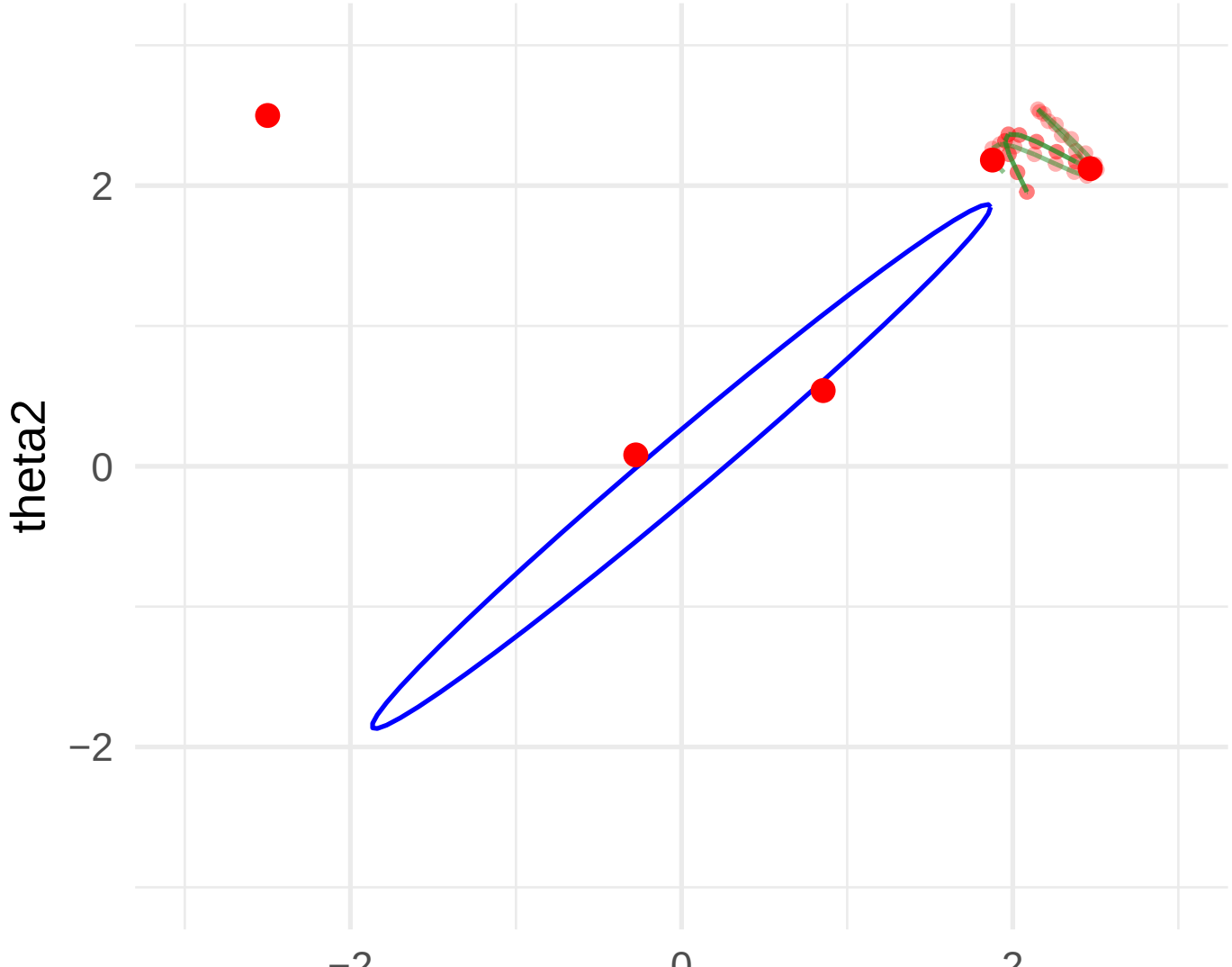
Hamiltonian Monte Carlo

- Uses log density (negative log density is called energy)
- Uses gradient of log density for more efficient sampling
- Augments parameter space with momentum variables



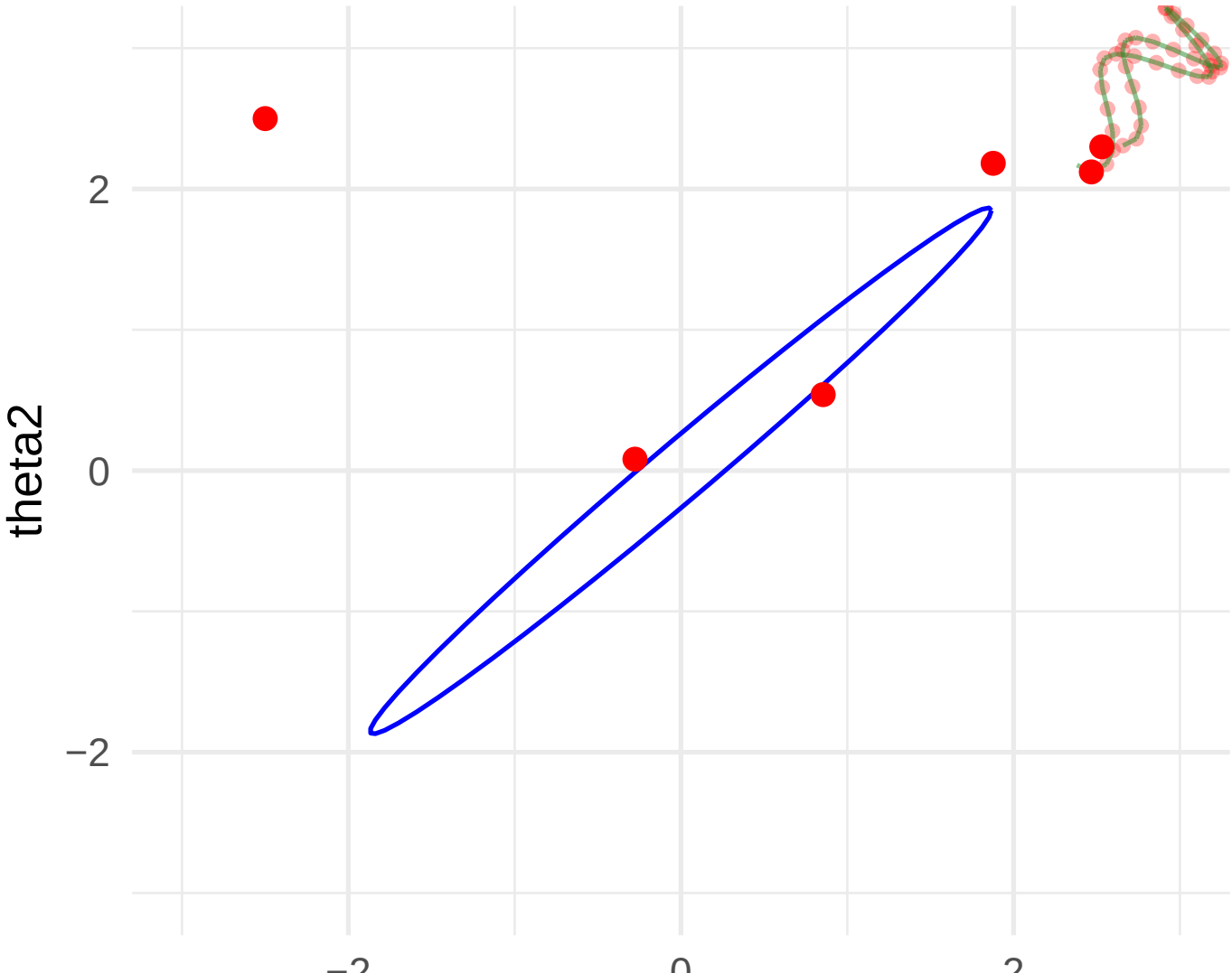
Hamiltonian Monte Carlo

- Uses log density (negative log density is called energy)
- Uses gradient of log density for more efficient sampling
- Augments parameter space with momentum variables



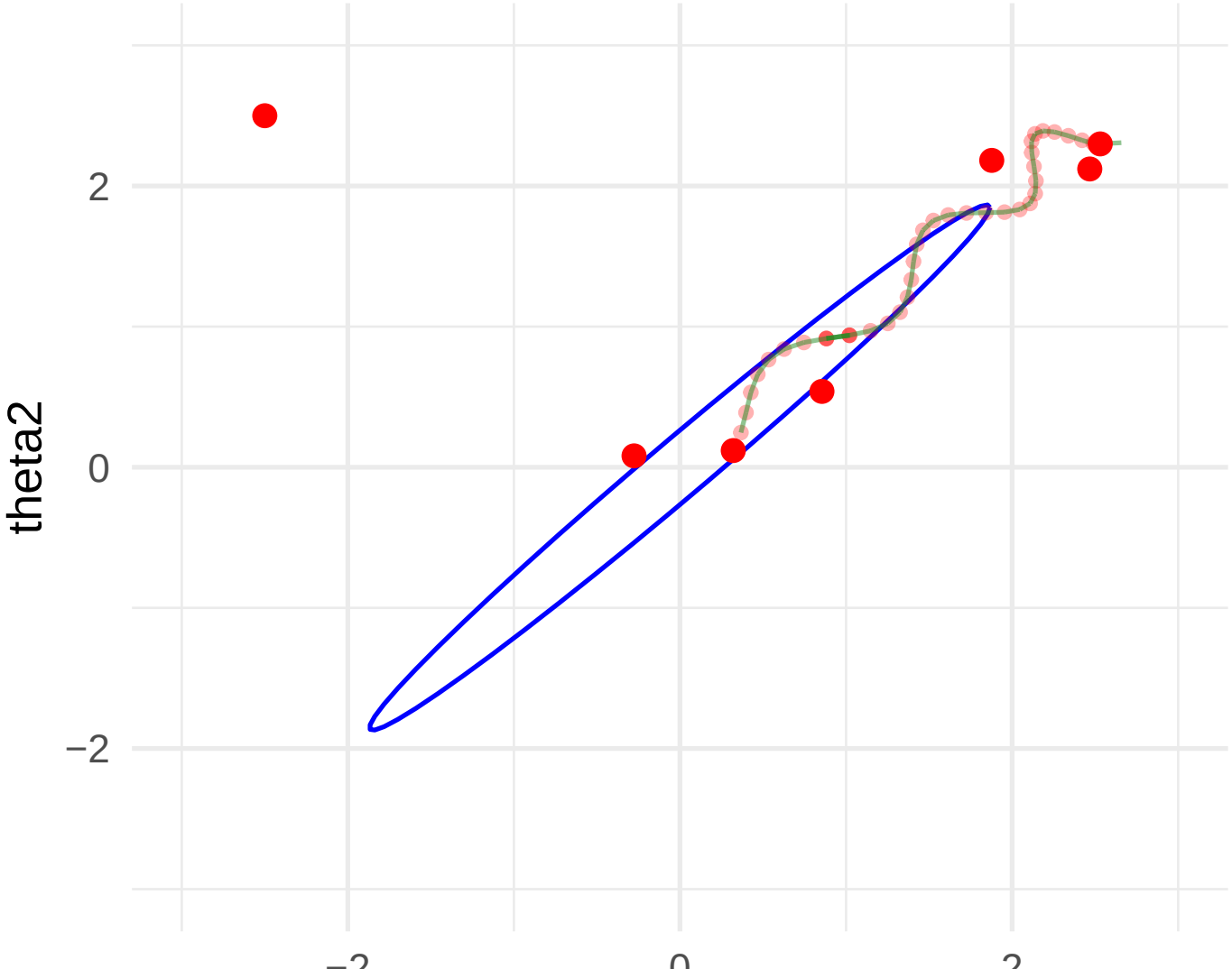
Hamiltonian Monte Carlo

- Uses log density (negative log density is called energy)
- Uses gradient of log density for more efficient sampling
- Augments parameter space with momentum variables



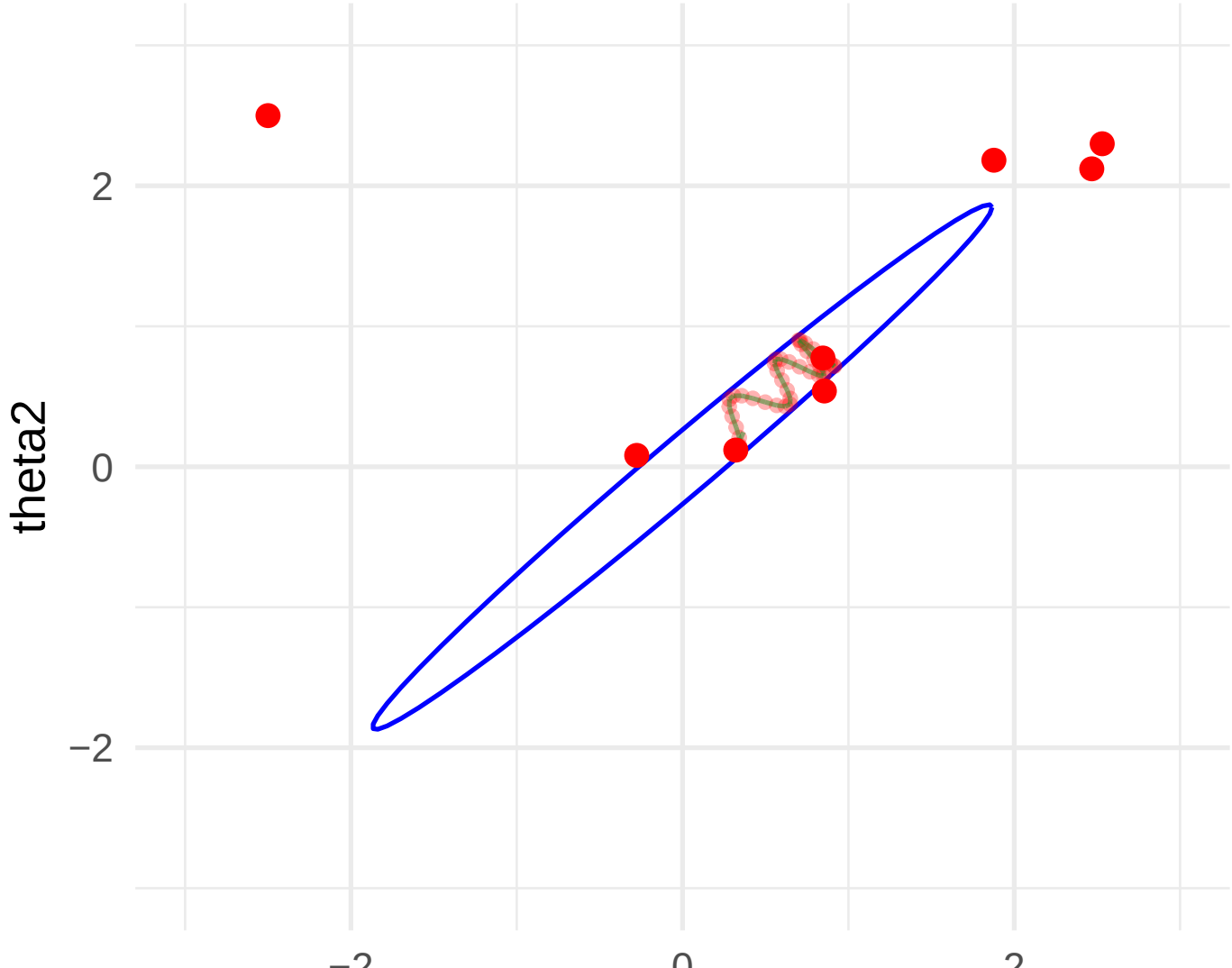
Hamiltonian Monte Carlo

- Uses log density (negative log density is called energy)
- Uses gradient of log density for more efficient sampling
- Augments parameter space with momentum variables



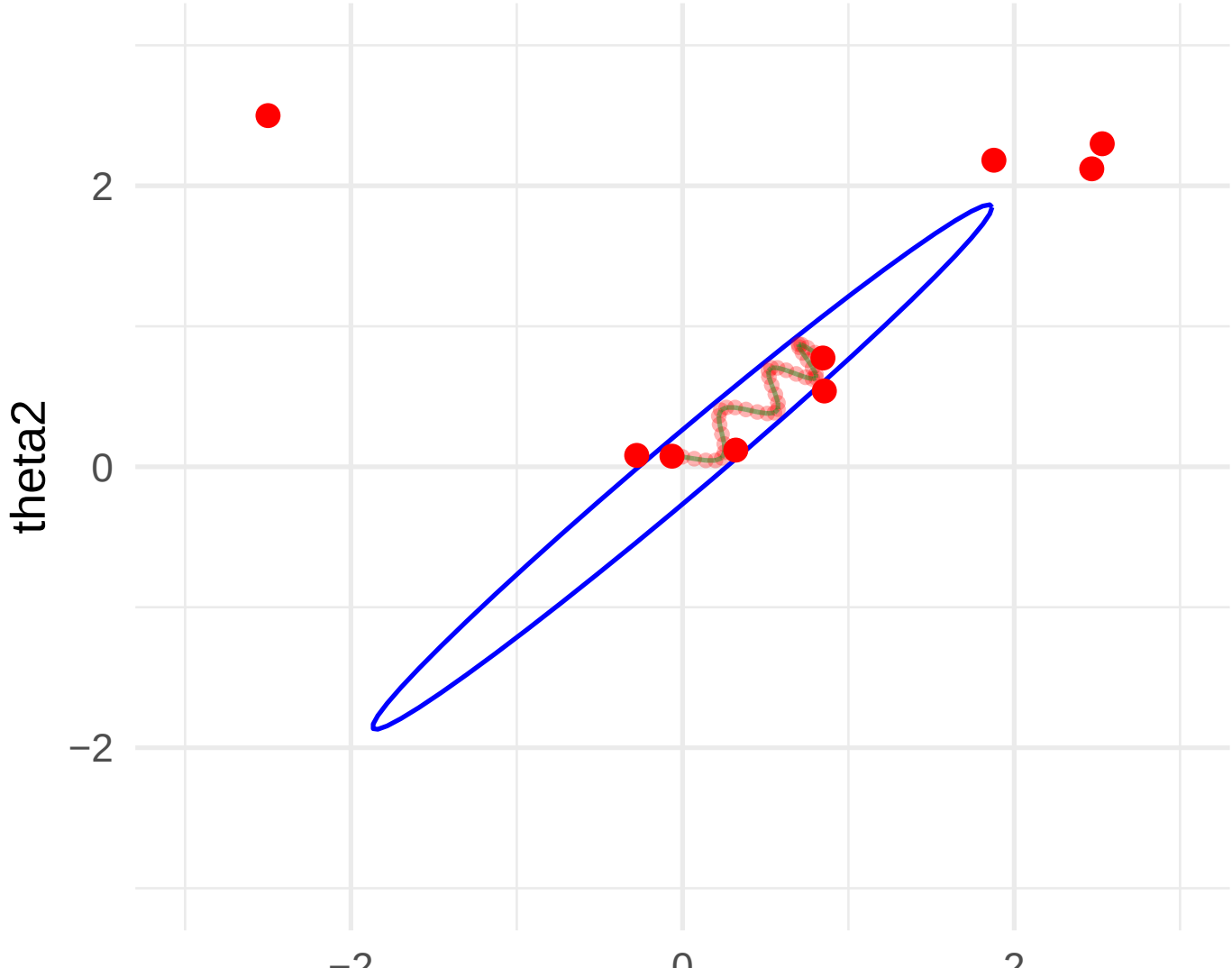
Hamiltonian Monte Carlo

- Uses log density (negative log density is called energy)
- Uses gradient of log density for more efficient sampling
- Augments parameter space with momentum variables



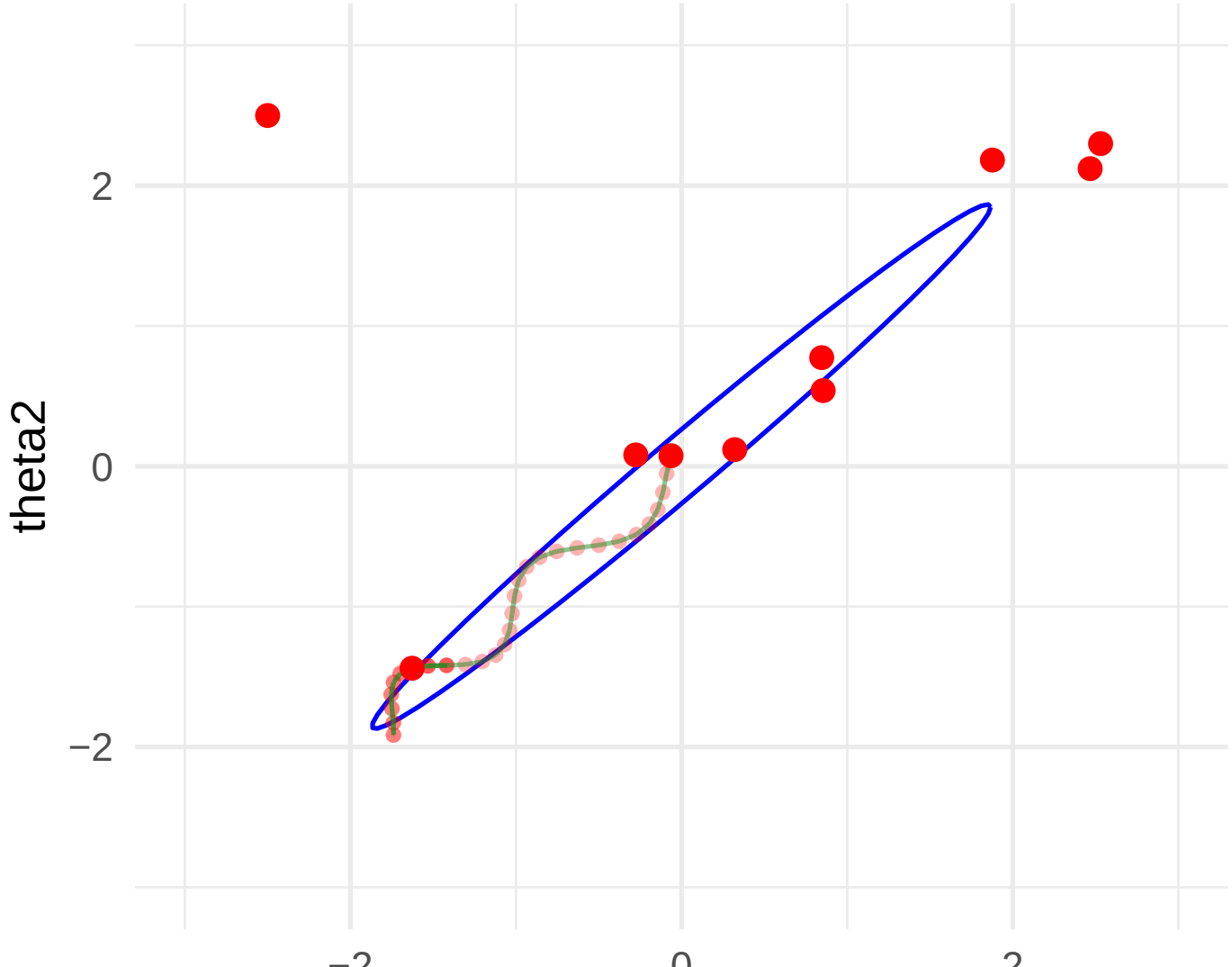
Hamiltonian Monte Carlo

- Uses log density (negative log density is called energy)
- Uses gradient of log density for more efficient sampling
- Augments parameter space with momentum variables



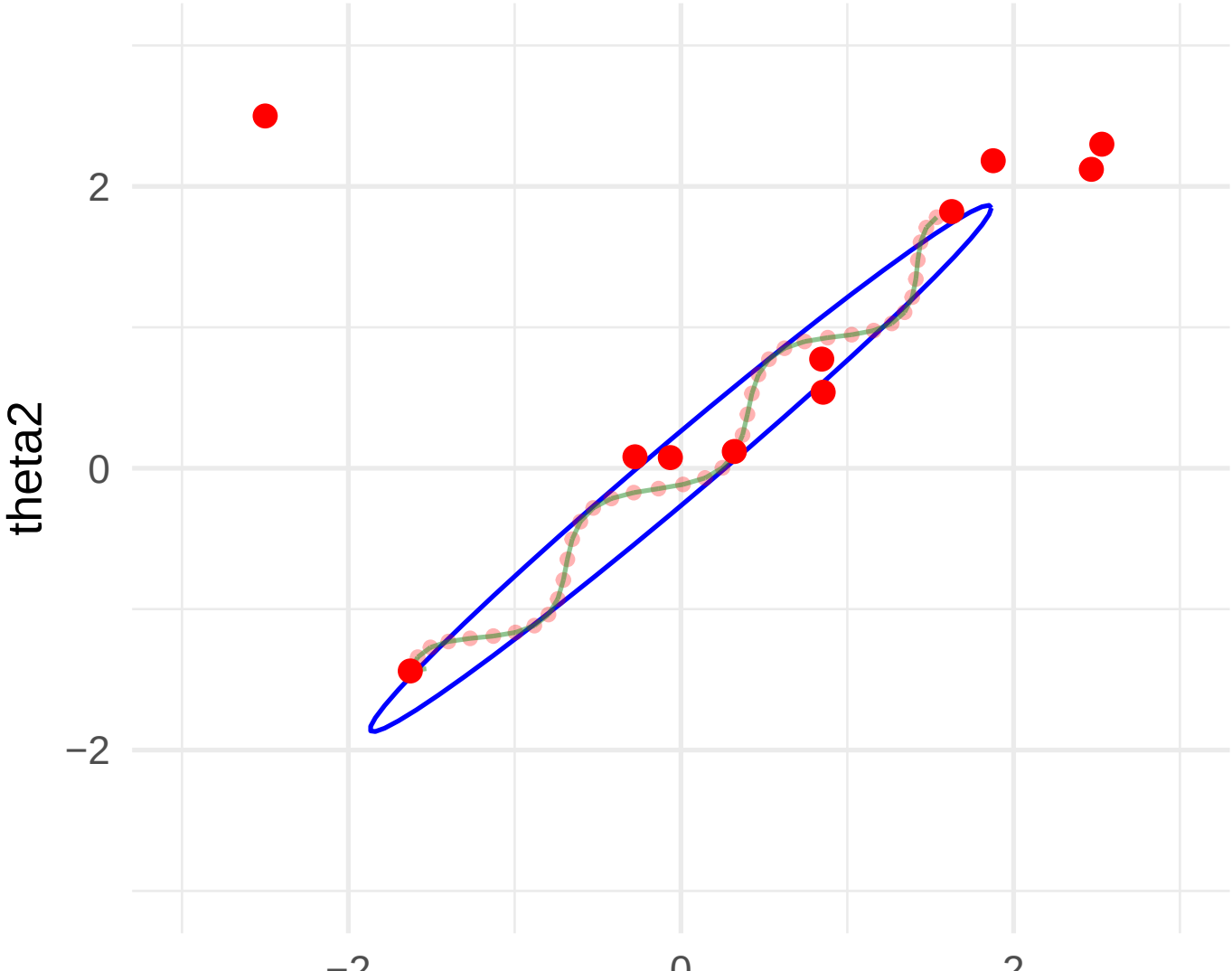
Hamiltonian Monte Carlo

- Uses log density (negative log density is called energy)
- Uses gradient of log density for more efficient sampling
- Augments parameter space with momentum variables



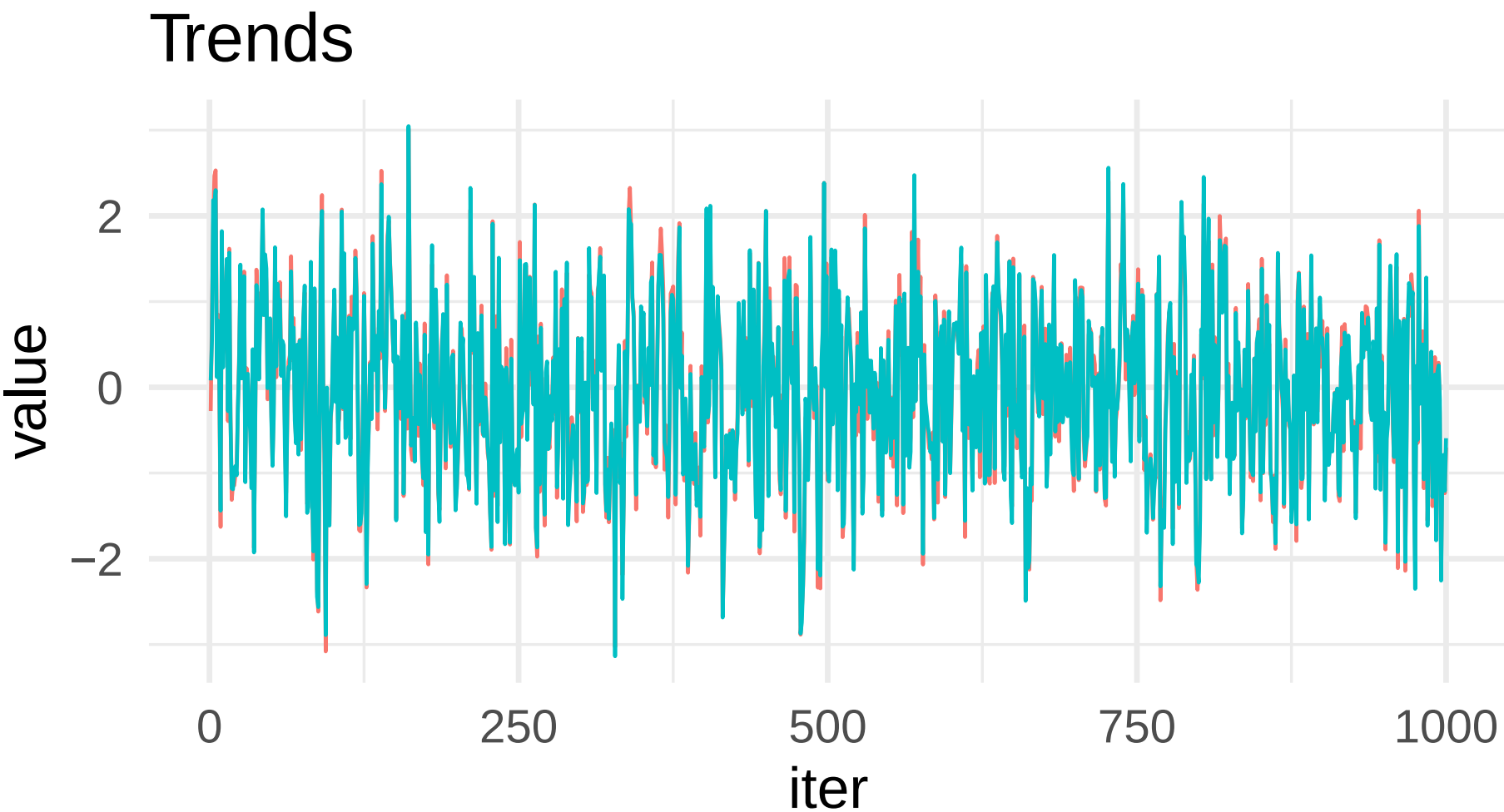
Hamiltonian Monte Carlo

- Uses log density (negative log density is called energy)
- Uses gradient of log density for more efficient sampling
- Augments parameter space with momentum variables



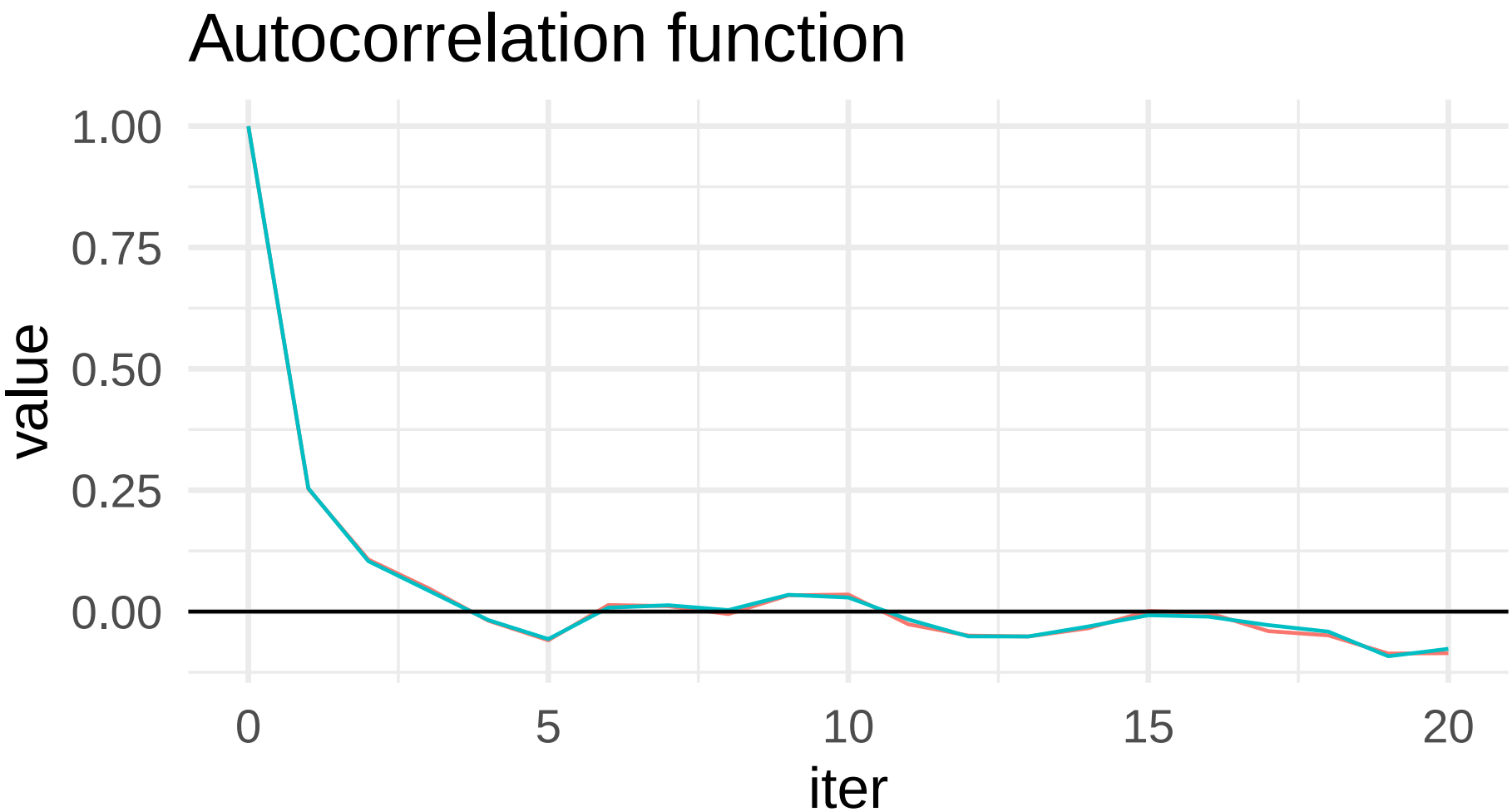
Hamiltonian Monte Carlo

- Uses log density (negative log density is called energy)
- Uses gradient of log density for more efficient sampling
- Augments parameter space with momentum variables



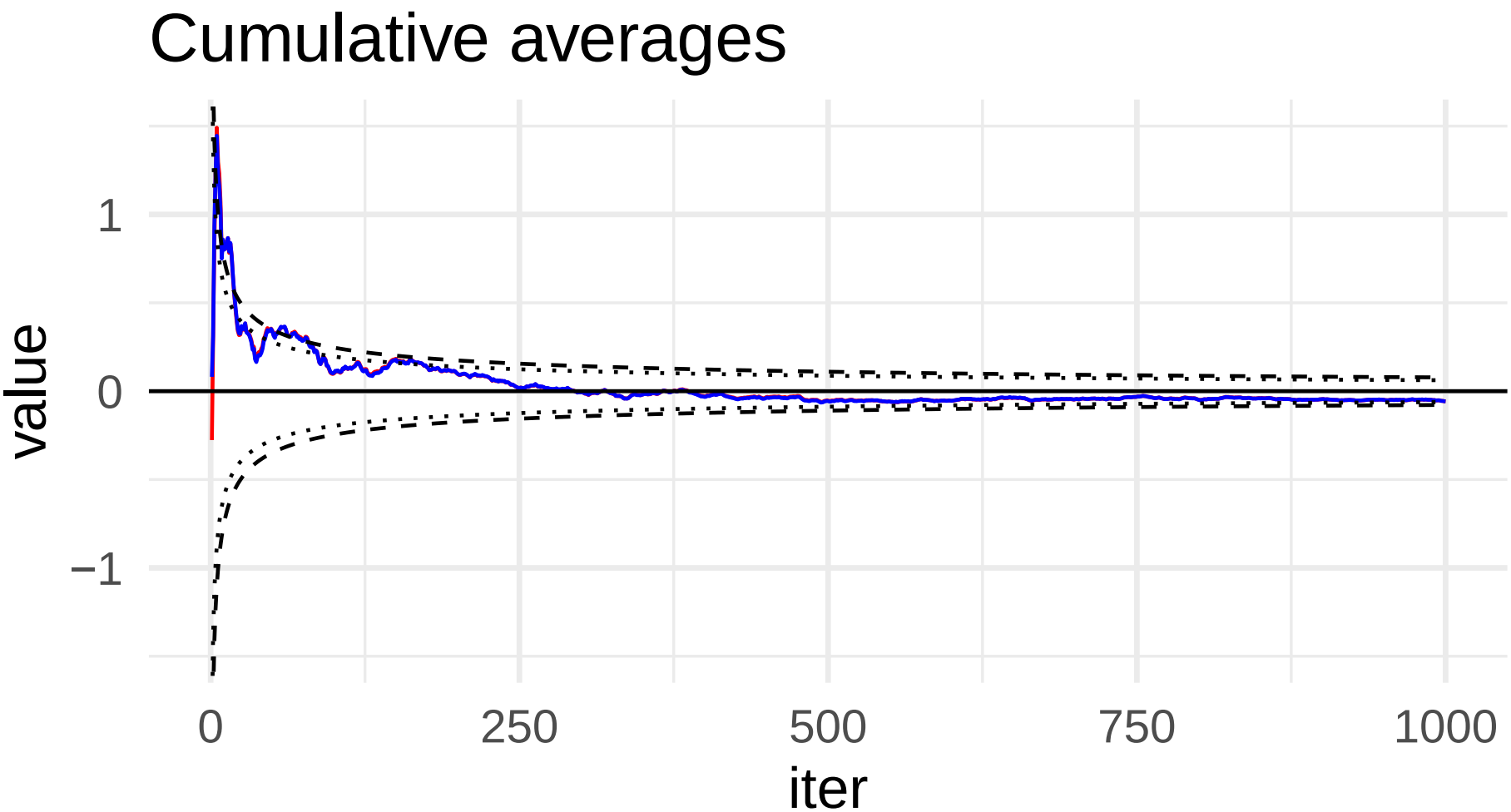
Hamiltonian Monte Carlo

- Uses log density (negative log density is called energy)
- Uses gradient of log density for more efficient sampling
- Augments parameter space with momentum variables



Hamiltonian Monte Carlo

- Uses log density (negative log density is called energy)
- Uses gradient of log density for more efficient sampling
- Augments parameter space with momentum variables



Hamiltonian Monte Carlo / No-U-Turn sampling

1. HMC basics (static HMC)
2. HMC + leapfrog discretization + Metropolis (static HMC)
3. NUTS + slice sampling + Metropolis (dynamic HMC)
4. NUTS + multinomial (dynamic HMC)

Hamiltonian Monte Carlo

- Previously
  - Factorizing: sample from 1) $p(\sigma^2)$, 2) $p(\mu \mid \sigma^2)$
  - Gibbs: sample from 1) $p(\sigma^2 \mid \mu)$, 2) $p(\mu \mid \sigma^2)$

Hamiltonian Monte Carlo

- Previously
  - Factorizing: sample from 1) $p(\sigma^2)$, 2) $p(\mu \mid \sigma^2)$
  - Gibbs: sample from 1) $p(\sigma^2 \mid \mu)$, 2) $p(\mu \mid \sigma^2)$
- HMC
  - Augment with ϕ (the same dimensionality as θ)
  - Sample from 1) $p(\phi)$, 2) $p(\theta, \phi)$

Hamiltonian Monte Carlo

1) Sample from $p(\phi)$
   - define $p(\phi) = \text{normal}(0, 1)$
2) Sample from $p(\theta, \phi)$
   - Hamiltonian dynamic simulation

Hamiltonian dynamic simulation

- Statistical mechanics and canonical distribution

$$\begin{aligned} p(\theta, \phi) &= p(\theta)p(\phi) \\ &= \frac{1}{Z} \exp(-H(\theta, \phi)) \\ &= \frac{1}{Z} \exp(-(U(\theta) + K(\phi))) \end{aligned}$$

where H is Hamiltonian energy function, U is potential energy function, K is kinetic energy function, and ϕ can now be considered to be a momentum variable

Hamiltonian dynamic simulation

- Statistical mechanics and canonical distribution

$$\begin{aligned} p(\theta, \phi) &= p(\theta)p(\phi) \\ &= \frac{1}{Z} \exp(-H(\theta, \phi)) \\ &= \frac{1}{Z} \exp(-(U(\theta) + K(\phi))) \end{aligned}$$

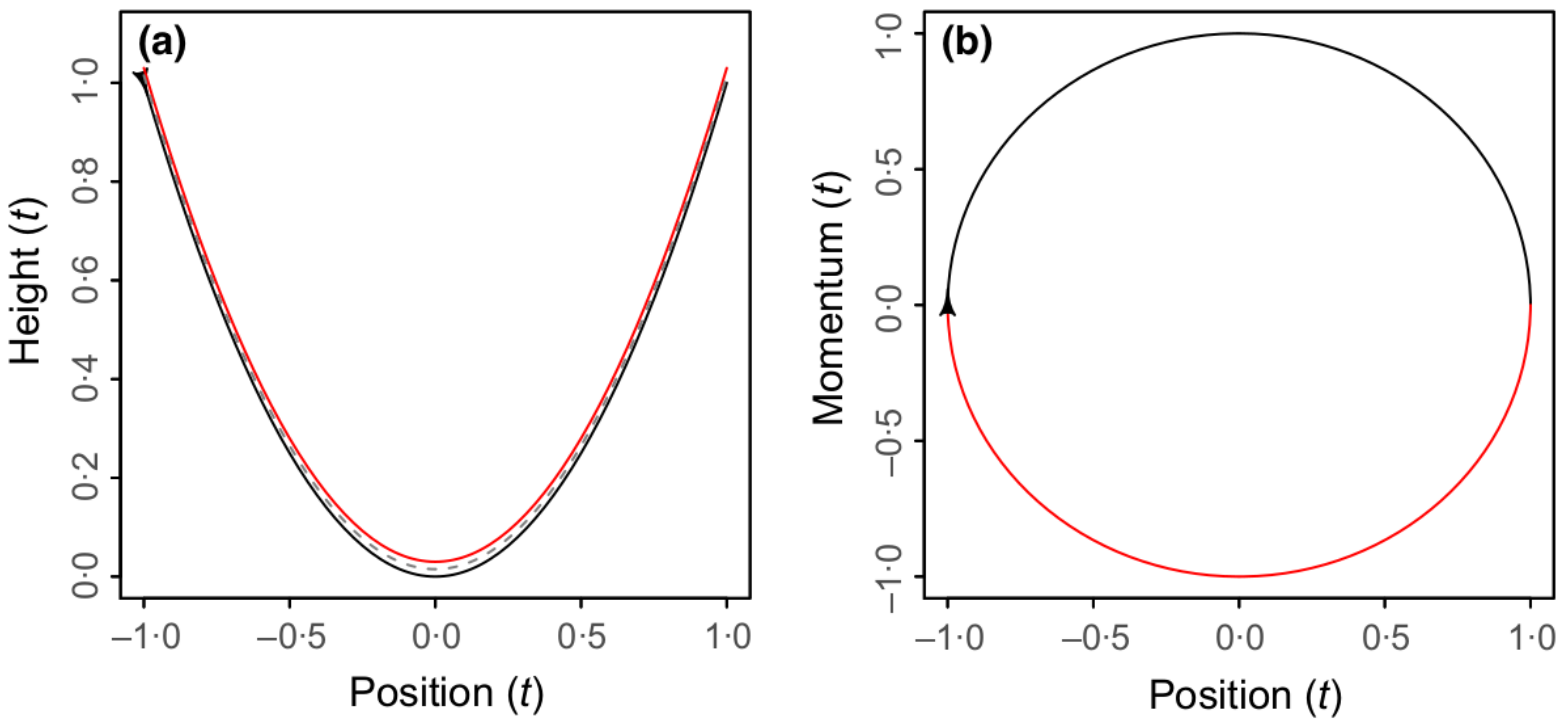
  where H is Hamiltonian energy function, U is potential energy function, K is kinetic energy function, and ϕ can now be considered to be a momentum variable

- The potential energy is the neg log density $U(\theta) = -\log(p(\theta)) + C$

Hamiltonian dynamic simulation

Equations of motion, use also the gradient

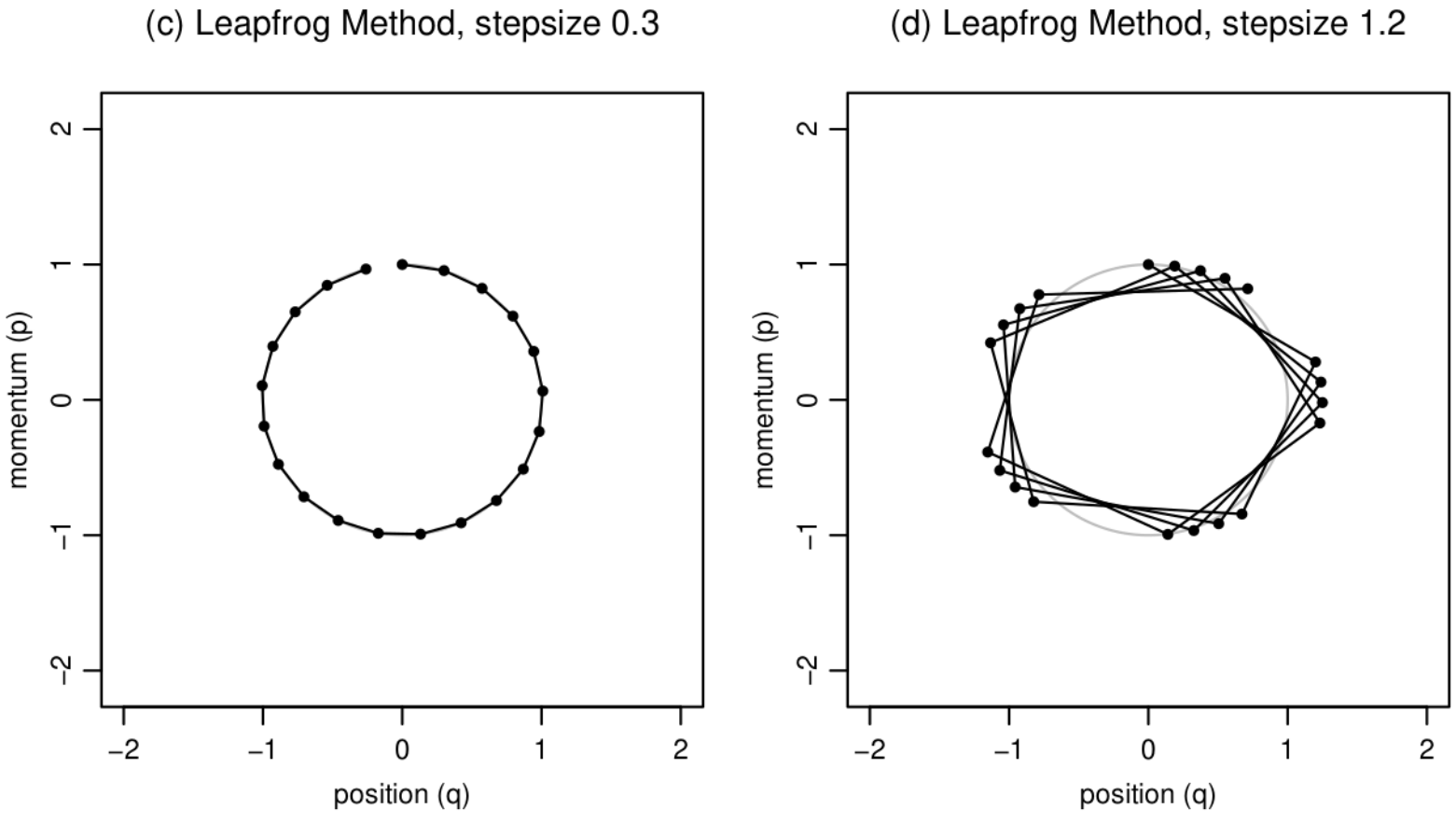
$$\frac{d\theta_i}{dt} = \frac{\partial H}{\partial p_i}$$
$$\frac{d\phi_i}{dt} = -\frac{\partial H}{\partial \theta_i}$$



From Monnahan et al (2017)

Leapfrog discretization

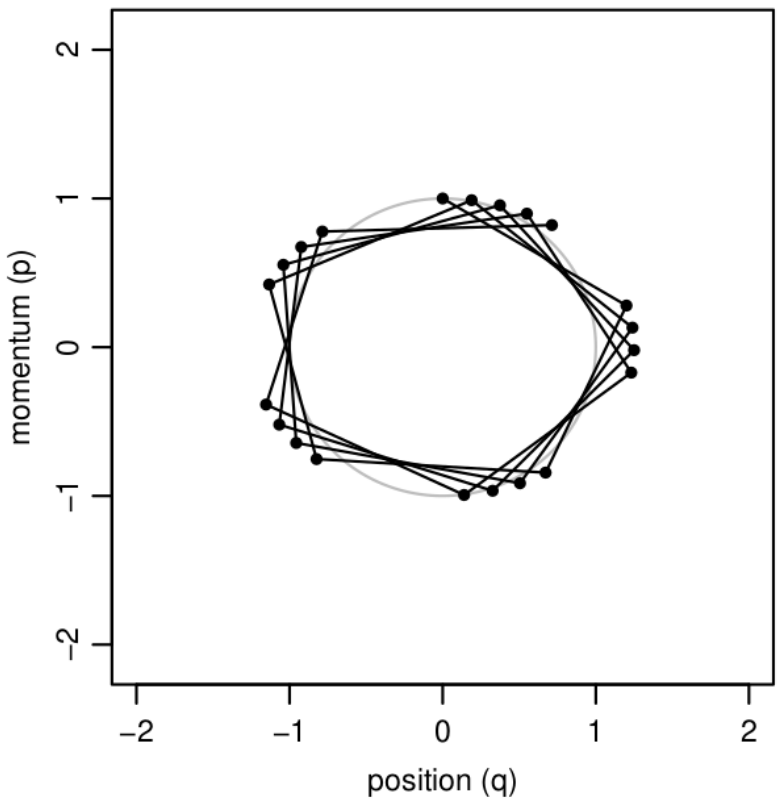
- Leapfrog discretization
  - preserves volume
  - reversible
  - discretization error does not usually grow in time



From Neal (2012)

Leapfrog discretization + Metropolis

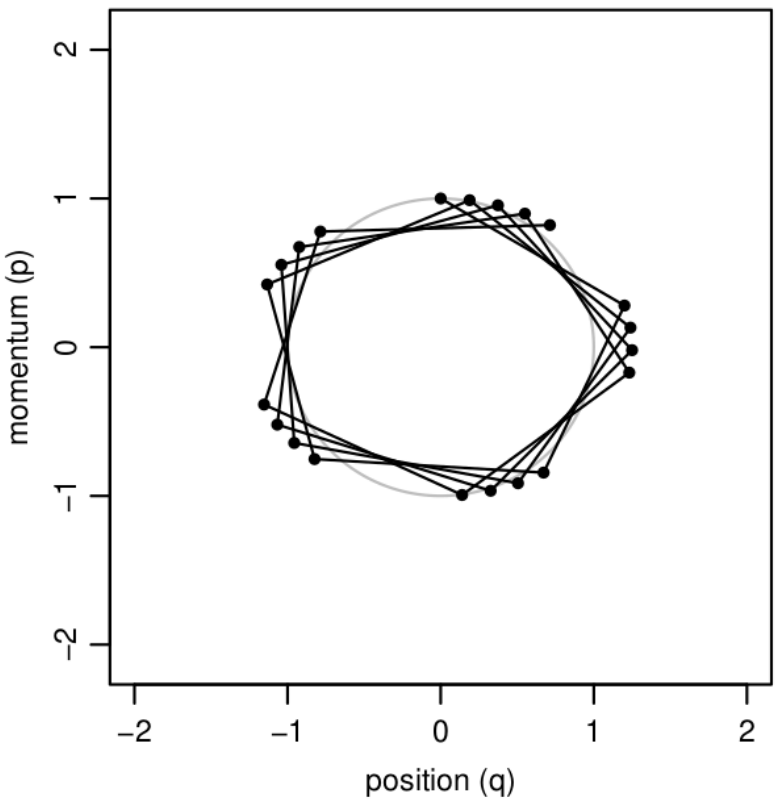
- Leapfrog discretization
  - due to the dicretization error the simulation steps away from the constant contour



From Neal (2012)

Leapfrog discretization + Metropolis

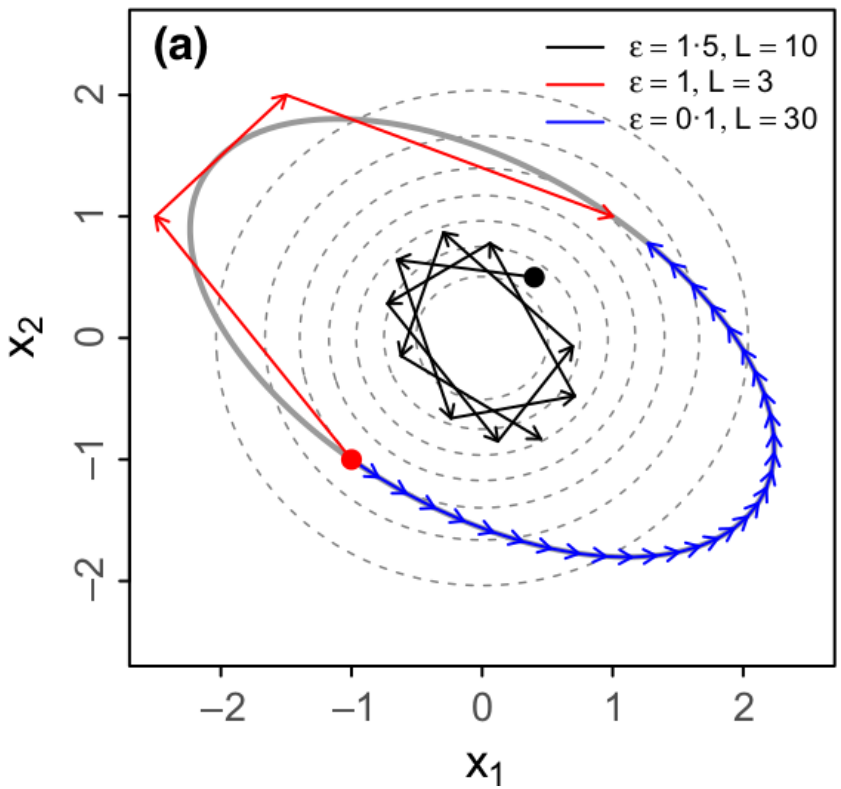
- Leapfrog discretization
  - due to the dicretization error the simulation steps away from the constant contour
- Metropolis step with $r = \exp\left(H(\theta^*, \phi^*) - H(\theta^{(t-1)}, \phi^{(t-1)})\right)$
  - accept if the Hamiltonian energy in the end is higher
  - accept with some probability if the Hamiltonian energy in the end is lower



From Neal (2012)

Leapfrog discretization, step size

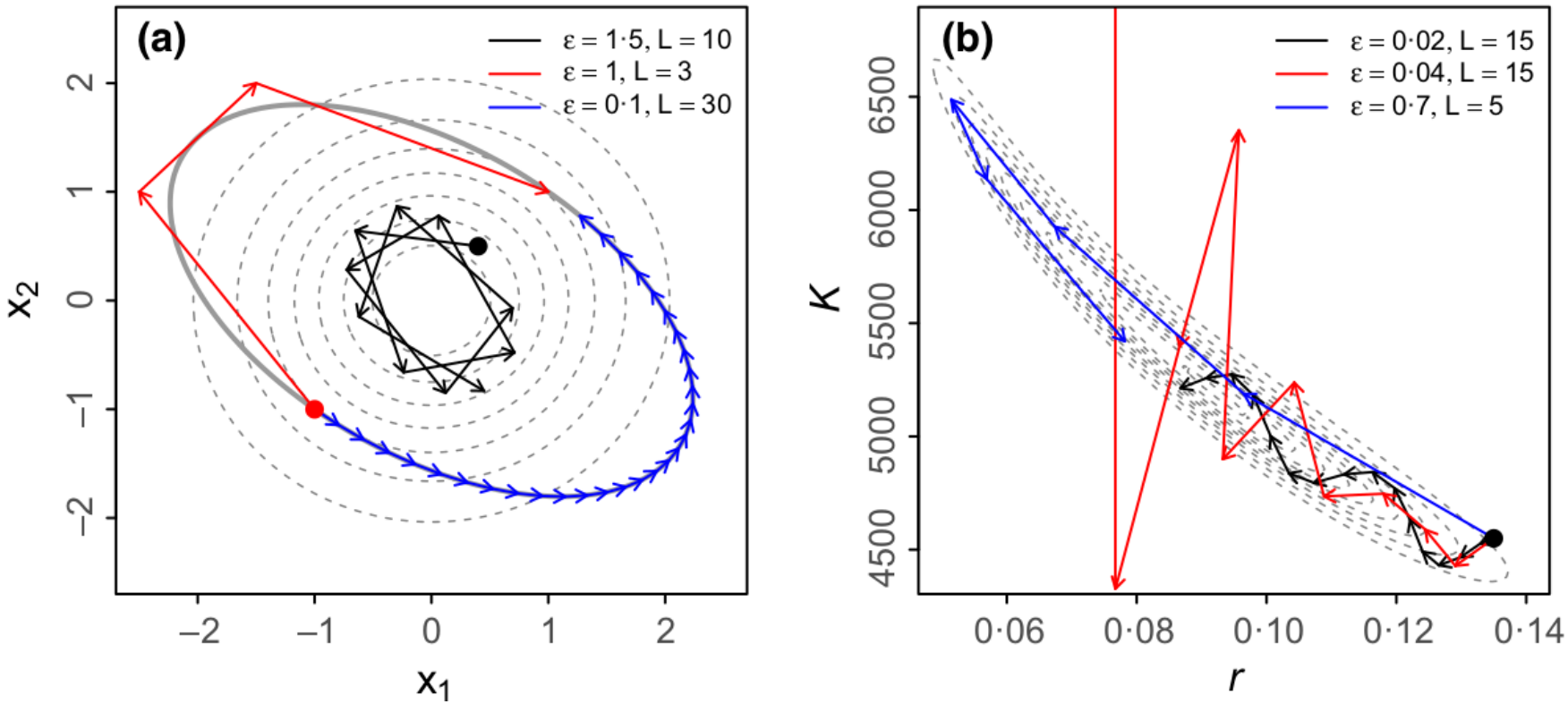
- Small step size $\rightarrow$ high acceptance rate, but many log density and gradient evaluations
- Big step size $\rightarrow$ less log density and gradient evaluations, but lower acceptance rate



From Monnahan et al (2017)

Leapfrog discretization, step size

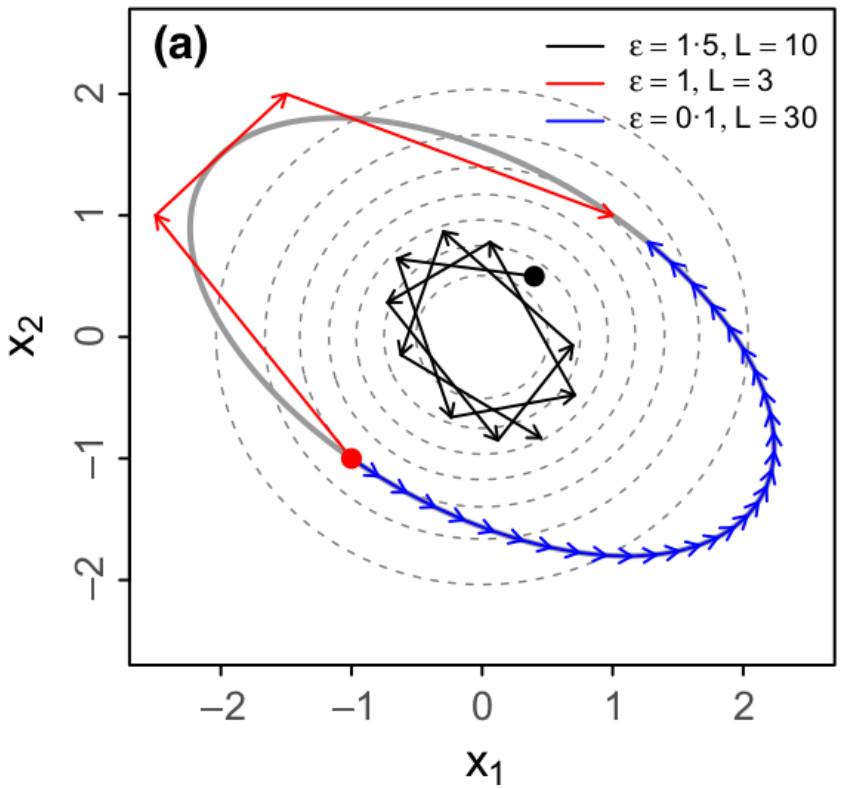
- Small step size $\rightarrow$ high acceptance rate, but many log density and gradient evaluations
- Big step size $\rightarrow$ less log density and gradient evaluations, but lower acceptance rate and the simulation may diverge



From Monnahan et al (2017)

Leapfrog discretization, the number of steps

- Many steps can reduce random walk
- Many steps require many log density and gradient evaluations



From Monnahan et al (2017)

Static Hamiltonian Monte Carlo

- Fixed number of steps
- Demo https://chi-feng.github.io/mcmc-demo/

No-U-Turn sampler

- Adaptively selects number of steps to improve robustness and efficiency
  - NUTS is a dynamic HMC algorithm, where dynamic refers to the dynamic trajectory length
  - simulate until a U-turn is detected
  - the number of simulation steps doubled if no U-turn yet

No-U-Turn sampler

- Adaptively selects number of steps to improve robustness and efficiency
  - NUTS is a dynamic HMC algorithm, where dynamic refers to the dynamic trajectory length
  - simulate until a U-turn is detected
  - the number of simulation steps doubled if no U-turn yet
- To keep reversibility of Markov chain
  - need to simulate in two directions
  - choose a point along the simulation path with slice sampling
  - Metropolis acceptance step for the selected point

No-U-Turn sampler

- Adaptively selects number of steps to improve robustness and efficiency
  - NUTS is a dynamic HMC algorithm, where dynamic refers to the dynamic trajectory length
  - simulate until a U-turn is detected
  - the number of simulation steps doubled if no U-turn yet
- To keep reversibility of Markov chain
  - need to simulate in two directions
  - choose a point along the simulation path with slice sampling
  - Metropolis acceptance step for the selected point
- For further efficiency
  - simulation path parts further away from the starting point can have higher probability
  - max treedepth to keep computation in control

No-U-Turn sampler

- Adaptively selects number of steps to improve robustness and efficiency
  - NUTS is a dynamic HMC algorithm, where dynamic refers to the dynamic trajectory length
  - simulate until a U-turn is detected
  - the number of simulation steps doubled if no U-turn yet
- To keep reversibility of Markov chain
  - need to simulate in two directions
  - choose a point along the simulation path with slice sampling
  - Metropolis acceptance step for the selected point
- For further efficiency
  - simulation path parts further away from the starting point can have higher probability
  - max treedepth to keep computation in control
- Demo https://chi-feng.github.io/mcmc-demo/

No-U-Turn sampler with multinomial sampling

- Original NUTS
  - choose a point along the simulation path with slice sampling
  - possibly with bigger weighting for further points
  - Metropolis acceptance step for the selected point
  - if the proposal is rejected the previous state is also the new state

No-U-Turn sampler with multinomial sampling

- Original NUTS
  - choose a point along the simulation path with slice sampling
  - possibly with bigger weighting for further points
  - Metropolis acceptance step for the selected point
  - if the proposal is rejected the previous state is also the new state
- NUTS with multinomial sampling
  - compute the probability of selecting a point and accepting it for all points
  - select the point with multinomial sampling
  - more likely to accept a point that is not the previous one

No-U-Turn sampler with multinomial sampling

- Original NUTS
  - choose a point along the simulation path with slice sampling
  - possibly with bigger weighting for further points
  - Metropolis acceptance step for the selected point
  - if the proposal is rejected the previous state is also the new state
- NUTS with multinomial sampling
  - compute the probability of selecting a point and accepting it for all points
  - select the point with multinomial sampling
  - more likely to accept a point that is not the previous one
- Demo https://chi-feng.github.io/mcmc-demo/

Mass matrix and the step size adaptation

- Mass matrix refers to having different scaling for different parameters and optionally also rotation to reduce correlations
  - mass matrix is estimated during the adaptation phase of the warm-up
  - mass matrix is estimated using the draws so far

Mass matrix and the step size adaptation

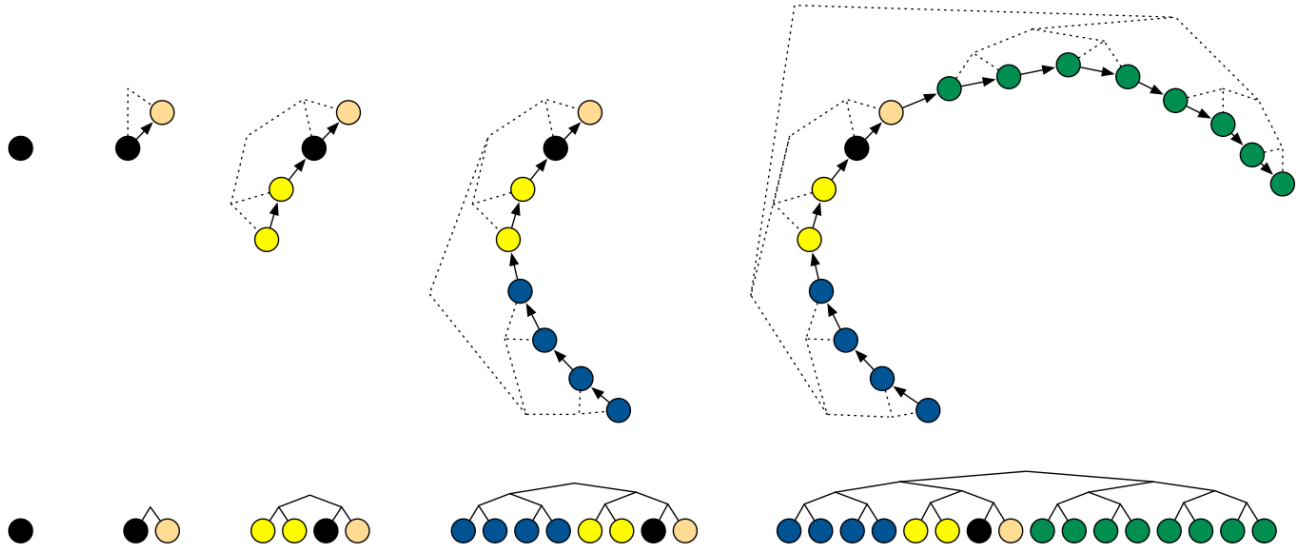
- Mass matrix refers to having different scaling for different parameters and optionally also rotation to reduce correlations
  - mass matrix is estimated during the adaptation phase of the warm-up
  - mass matrix is estimated using the draws so far
- Step size
  - adjusted to be as big as possible while keeping discretization error in control (adapt_delta)

Mass matrix and the step size adaptation

- Mass matrix refers to having different scaling for different parameters and optionally also rotation to reduce correlations
  - mass matrix is estimated during the adaptation phase of the warm-up
  - mass matrix is estimated using the draws so far
- Step size
  - adjusted to be as big as possible while keeping discretization error in control (adapt_delta)
- After adaptation the algorithm parameters are fixed and some more iterations run to finish the warmup

Max tree depth diagnostic

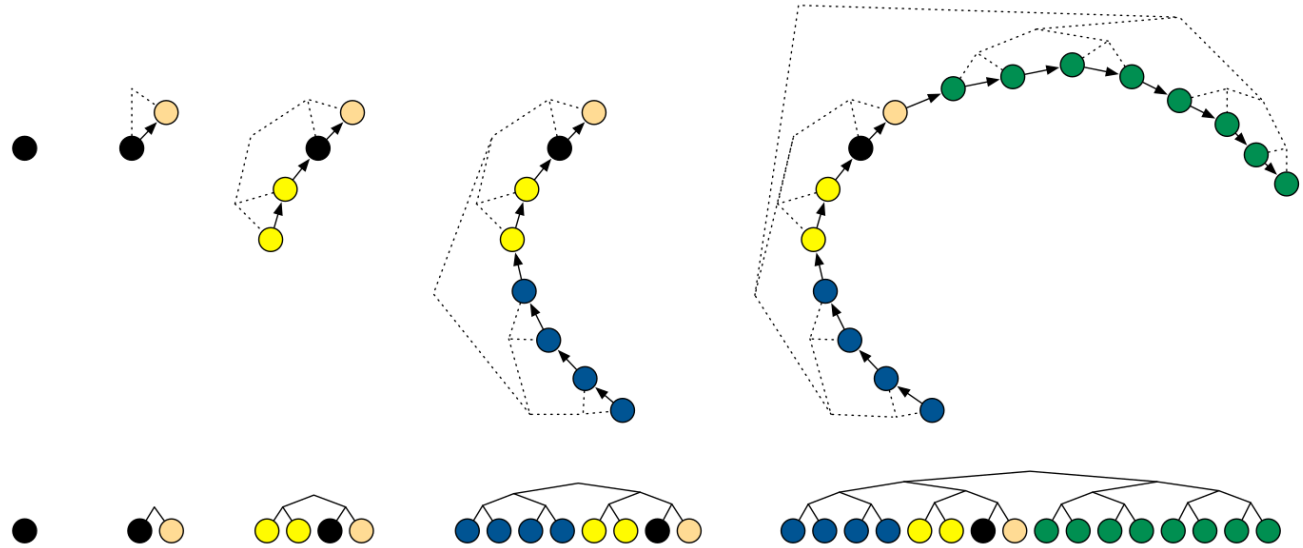
- NUTS specific diagnostic
  - the dynamic simulation is build as a binary tree



from Hoffman & Gelman (2014)

Max tree depth diagnostic

- NUTS specific diagnostic
  - the dynamic simulation is build as a binary tree

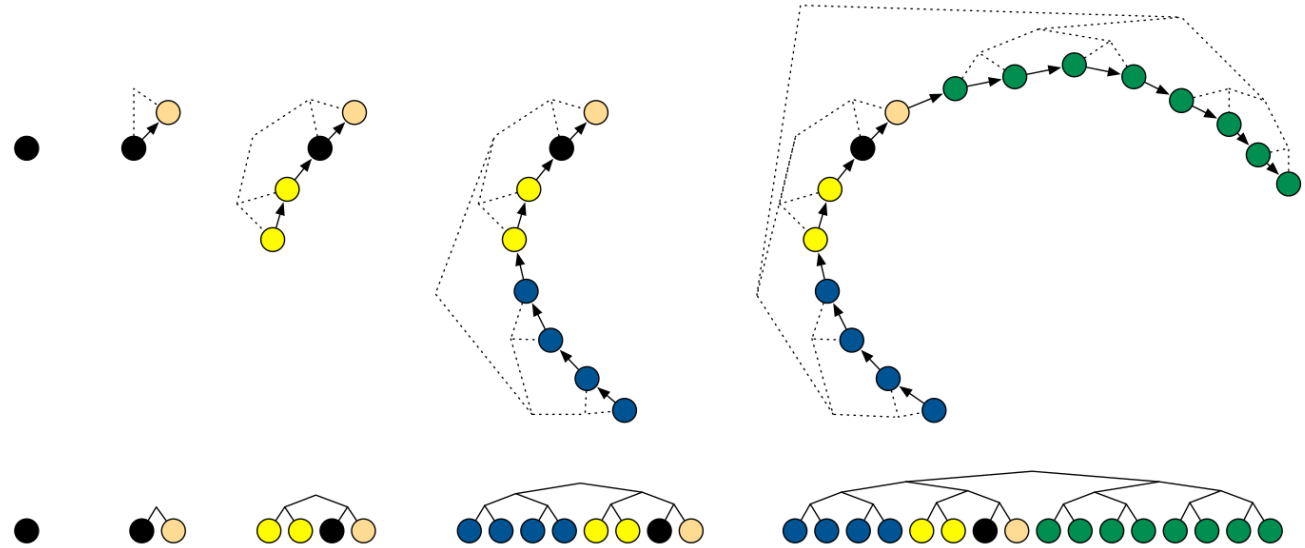


from Hoffman & Gelman (2014)

  - maximum simulation length is capped to avoid very long waiting times in case of bad behavior

Max tree depth diagnostic

- NUTS specific diagnostic
  - the dynamic simulation is build as a binary tree

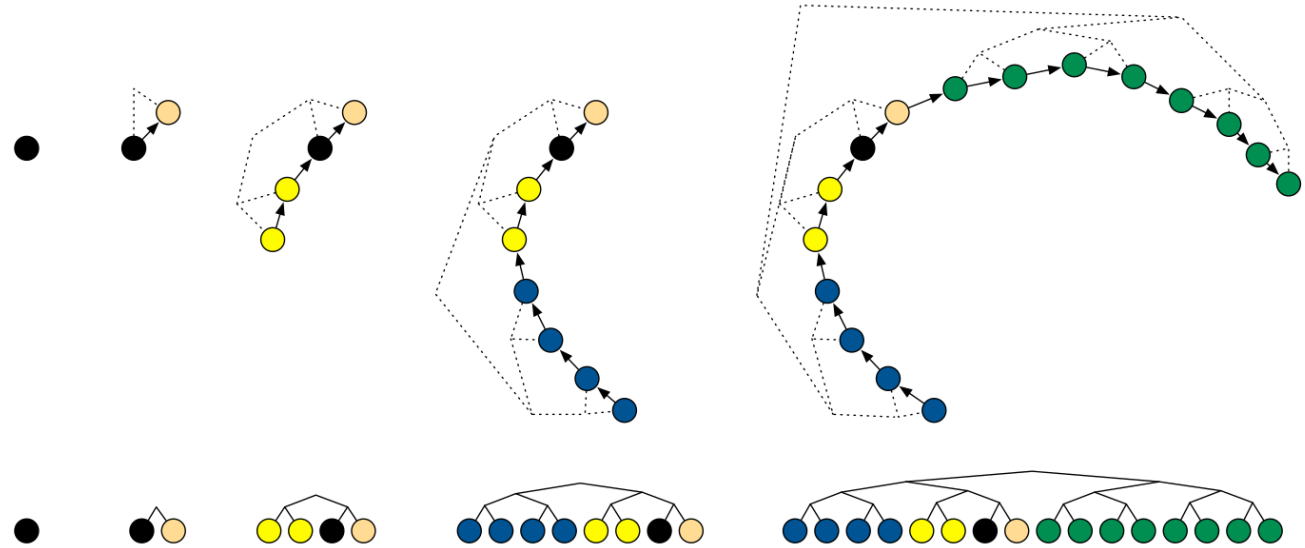


from Hoffman & Gelman (2014)

  - maximum simulation length is capped to avoid very long waiting times in case of bad behavior
- Indicates inefficiency in sampling leading to higher autocorrelations and lower ESS (S_{eff})
  - very low inefficiency can indicate problems that need to be inverse-distance
  - moderate inefficiency doesn't invalidate the result

Max tree depth diagnostic

- NUTS specific diagnostic
  - the dynamic simulation is build as a binary tree

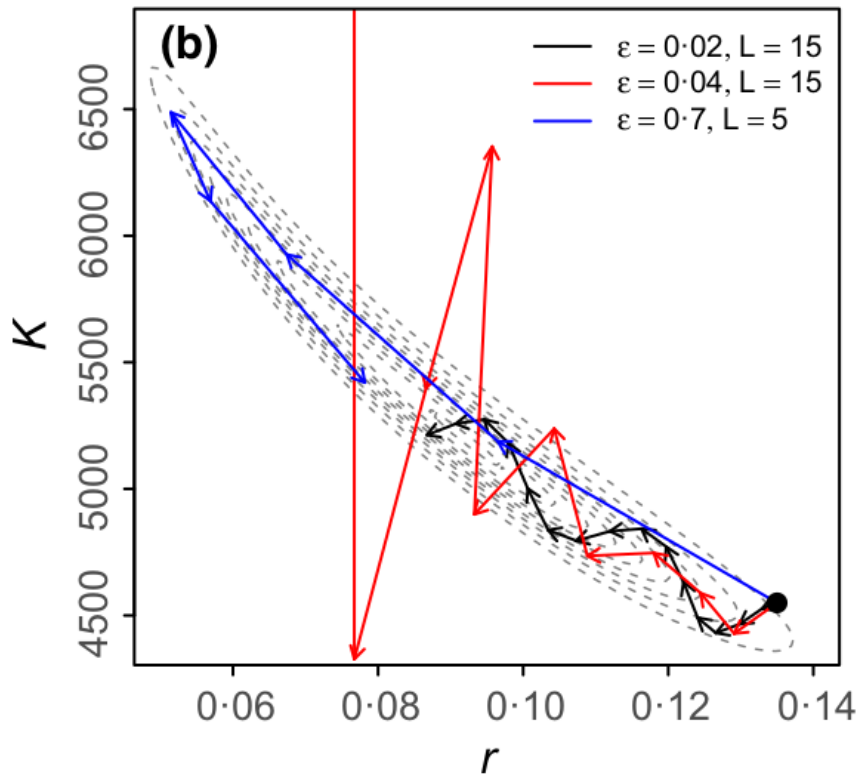


from Hoffman & Gelman (2014)

  - maximum simulation length is capped to avoid very long waiting times in case of bad behavior
- Indicates inefficiency in sampling leading to higher autocorrelations and lower ESS (S_{eff})
  - very low inefficiency can indicate problems that need to be inverse-distance
  - moderate inefficiency doesn't invalidate the result
- Different parameterizations matter

Divergences

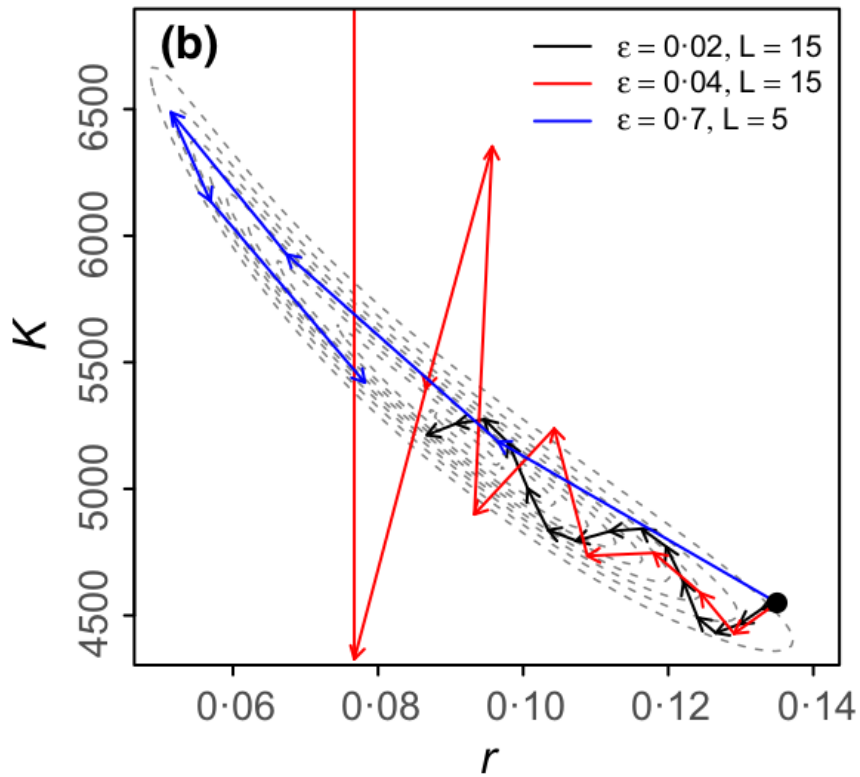
- HMC specific: indicates that Hamiltonian dynamic simulation has problems with unexpected fast changes in log-density (compared to the used step size)
  - indicates possibility of biased estimates



From Monnahan et al (2017)

Divergences

- HMC specific: indicates that Hamiltonian dynamic simulation has problems with unexpected fast changes in log-density (compared to the used step size)
  - indicates possibility of biased estimates



From Monnahan et al (2017)

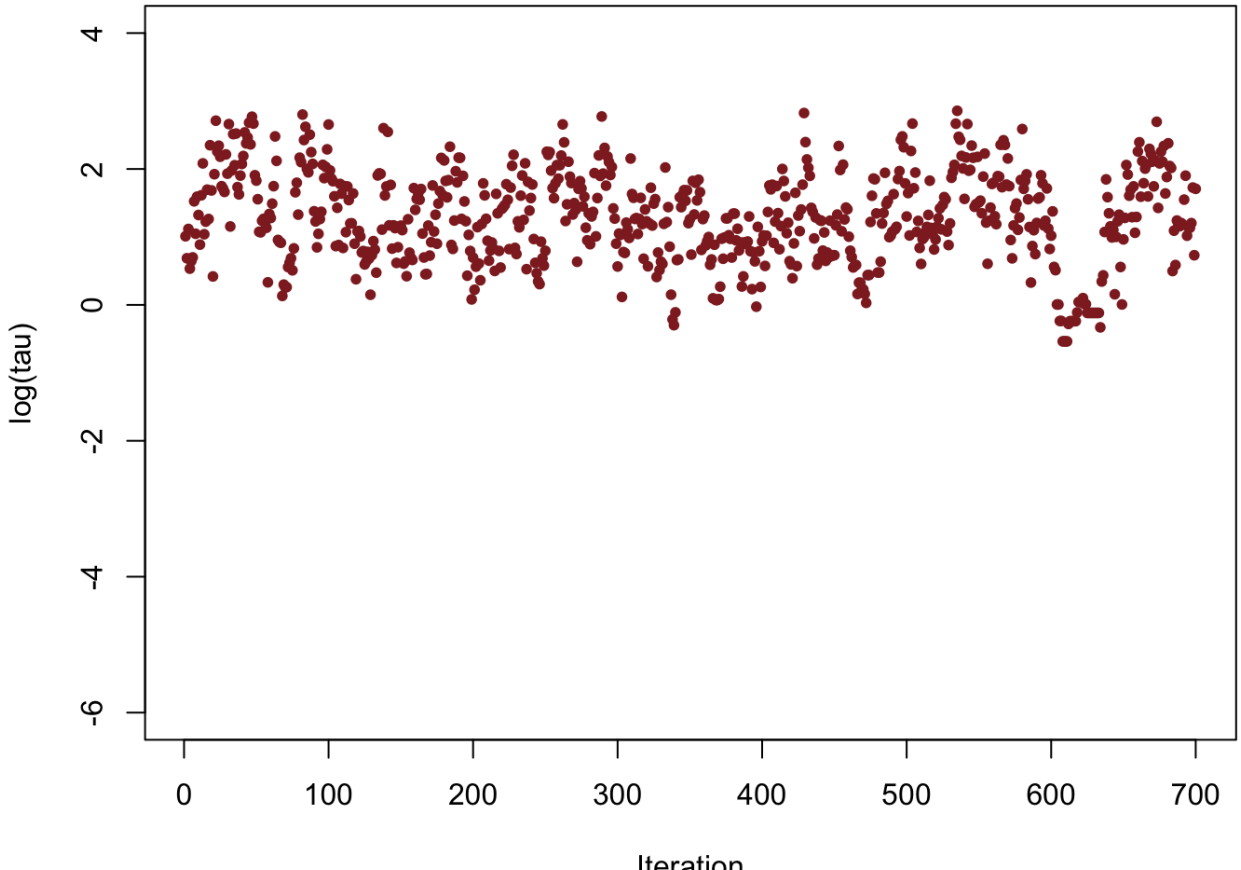
- Demo https://chi-feng.github.io/mcmc-demo/

Divergences

- HMC specific: indicates that Hamiltonian dynamic simulation has problems with unexpected fast changes in log-density
  - indicates possibility of biased estimates
- http://mc-stan.org/users/documentation/case-studies/divergences_and_bias.html

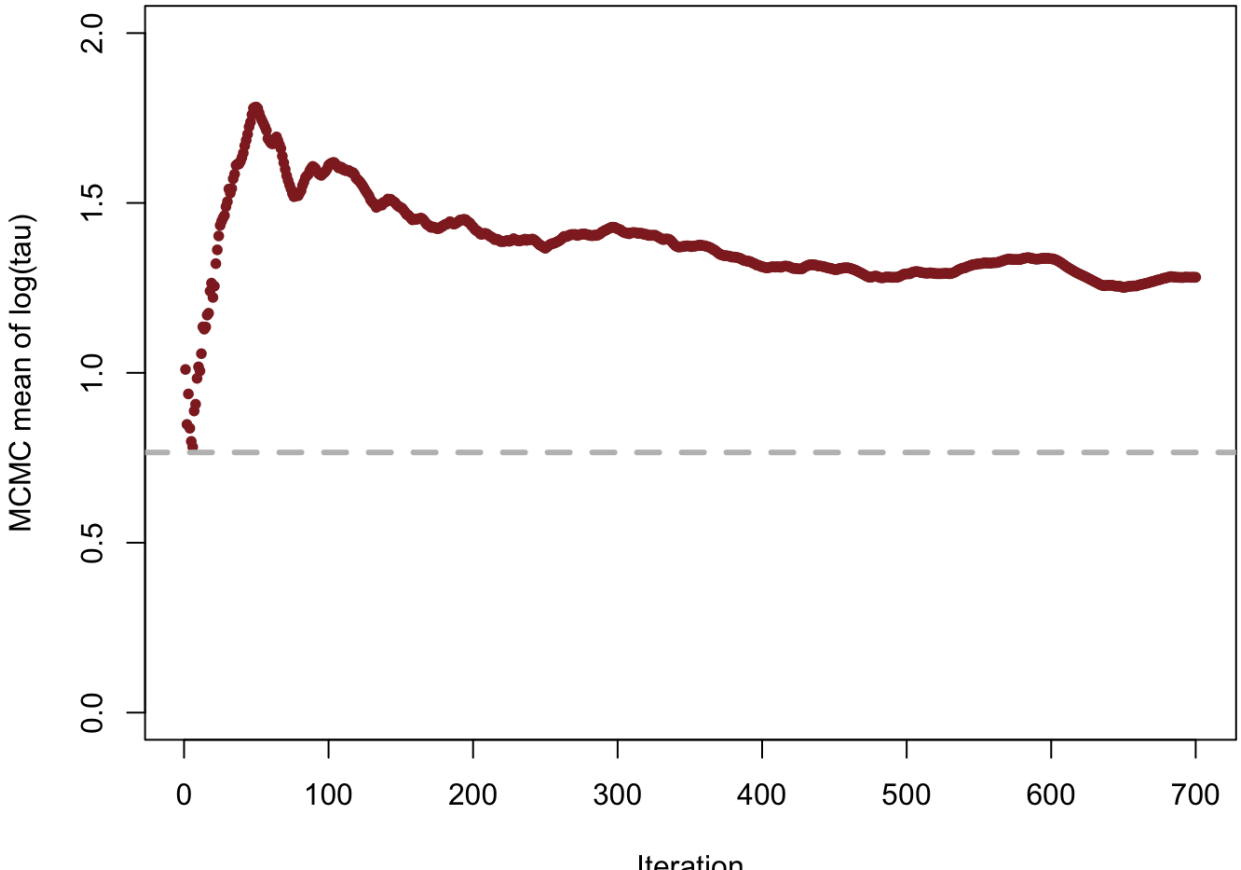
Divergences

- HMC specific: indicates that Hamiltonian dynamic simulation has problems with unexpected fast changes in log-density
  - indicates possibility of biased estimates
- http://mc-stan.org/users/documentation/case-studies/divergences_and_bias.html



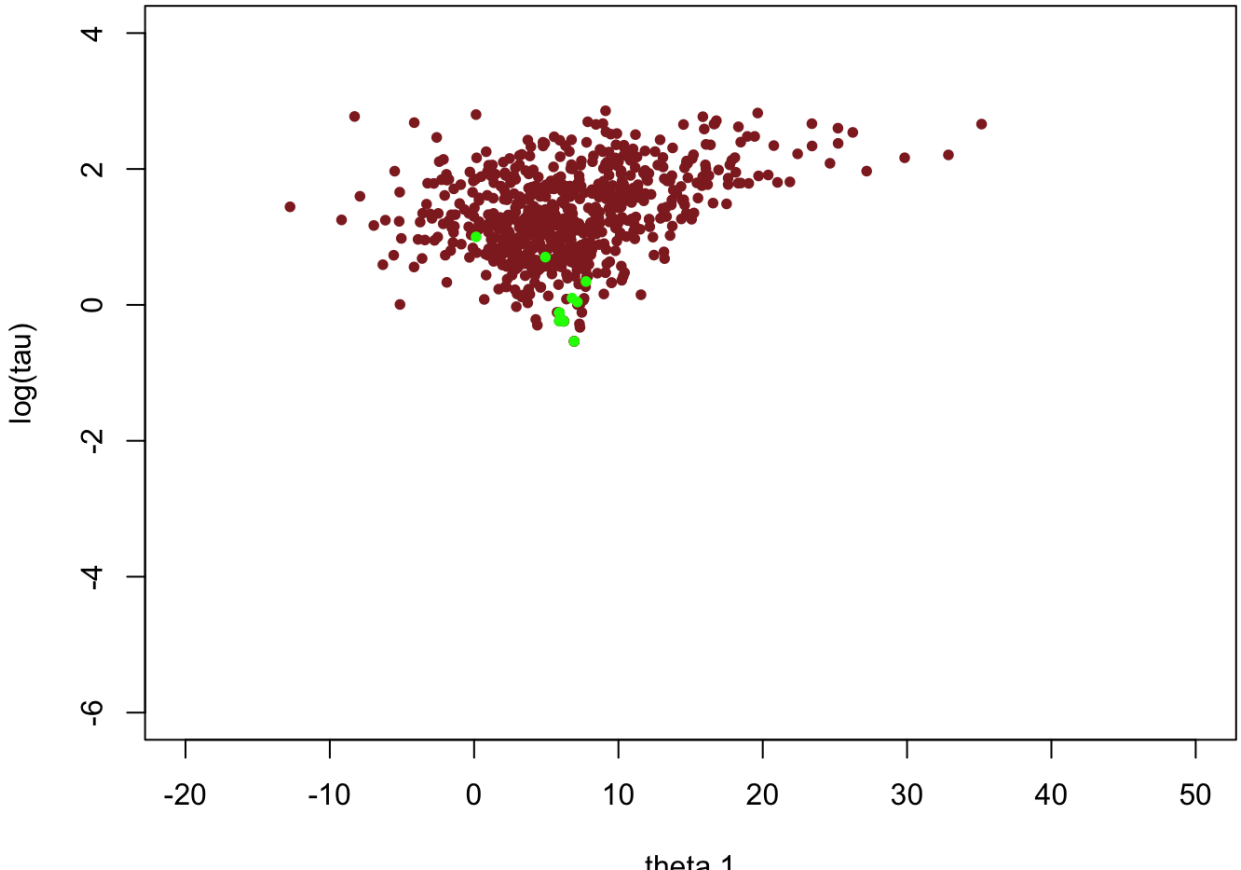
Divergences

- HMC specific: indicates that Hamiltonian dynamic simulation has problems with unexpected fast changes in log-density
  - indicates possibility of biased estimates
- http://mc-stan.org/users/documentation/case-studies/divergences_and_bias.html



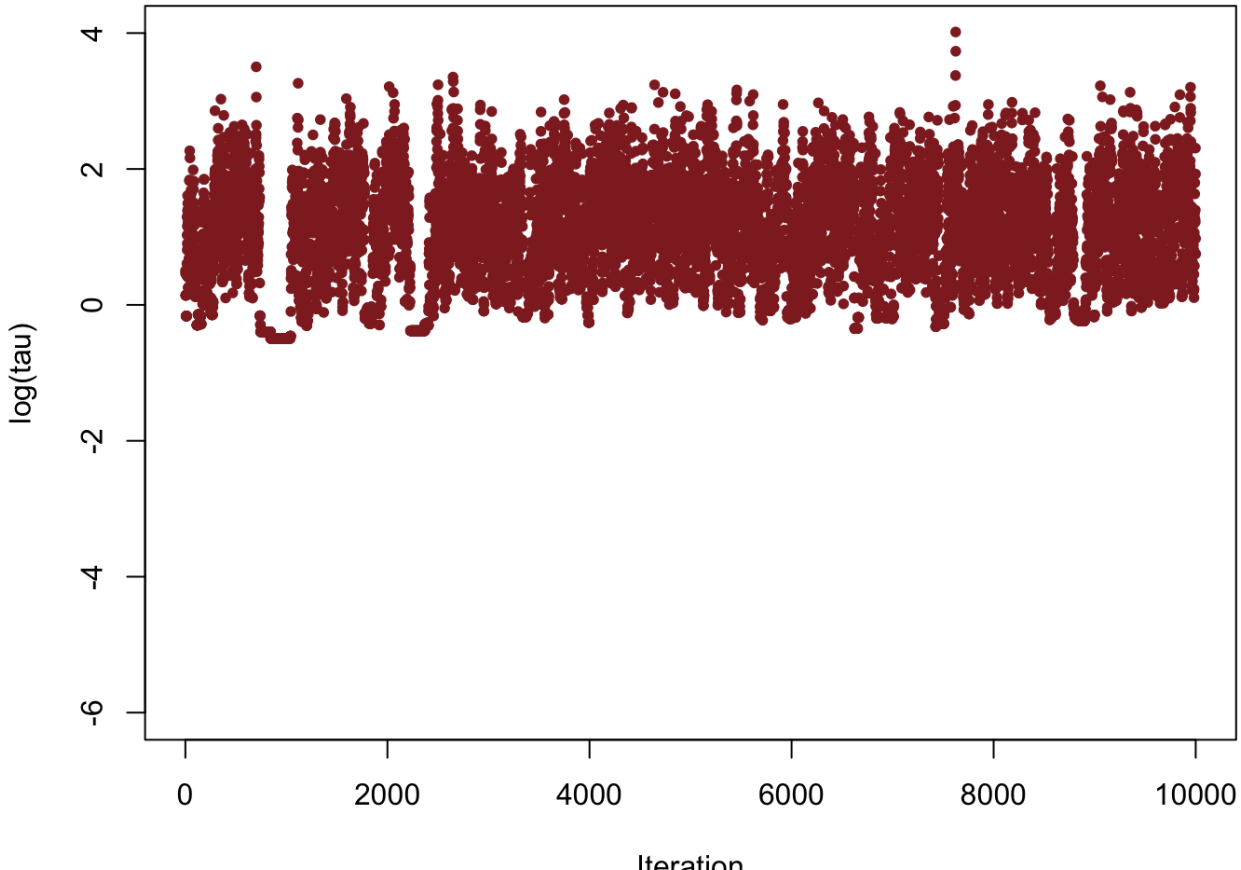
Divergences

- HMC specific: indicates that Hamiltonian dynamic simulation has problems with unexpected fast changes in log-density
  - indicates possibility of biased estimates
- http://mc-stan.org/users/documentation/case-studies/divergences_and_bias.html



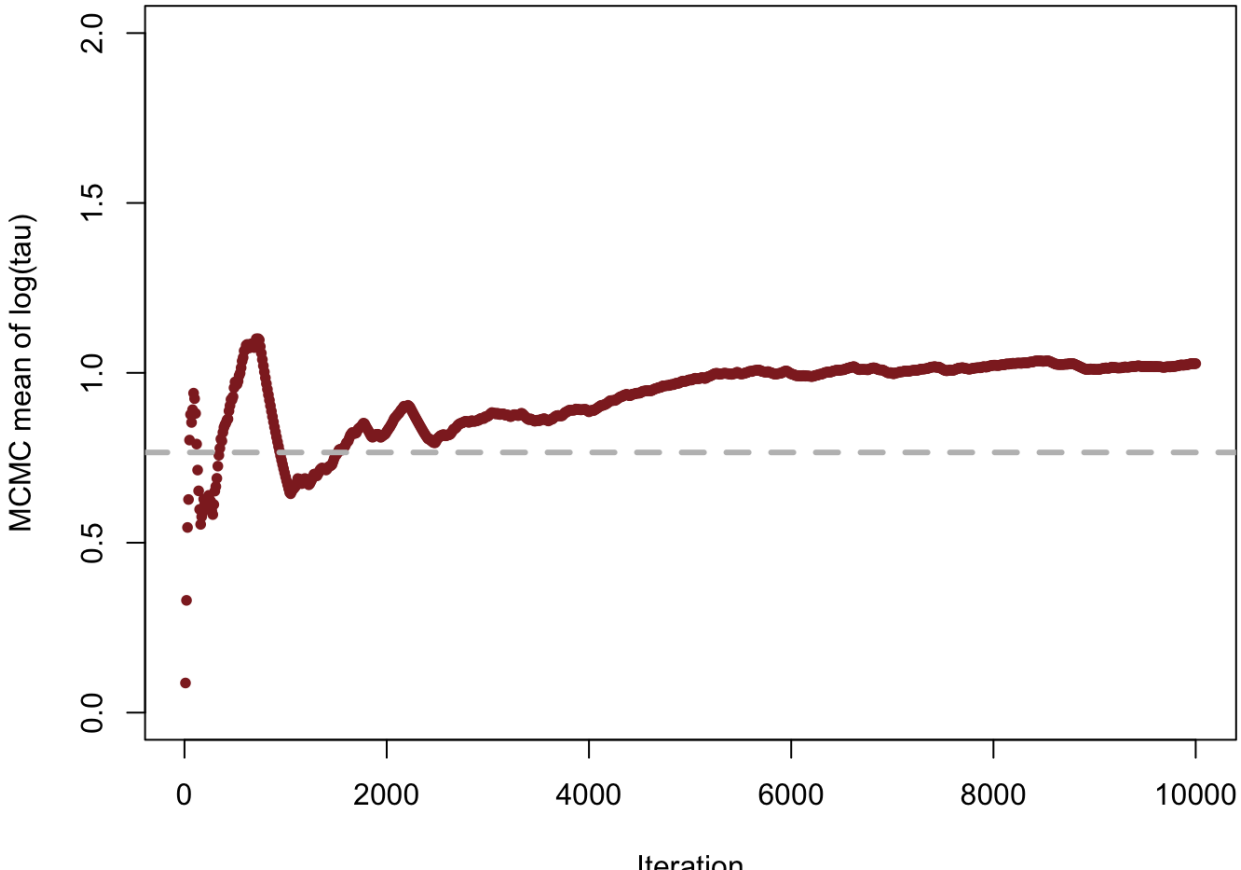
Divergences

- HMC specific: indicates that Hamiltonian dynamic simulation has problems with unexpected fast changes in log-density
  - indicates possibility of biased estimates
- http://mc-stan.org/users/documentation/case-studies/divergences_and_bias.html



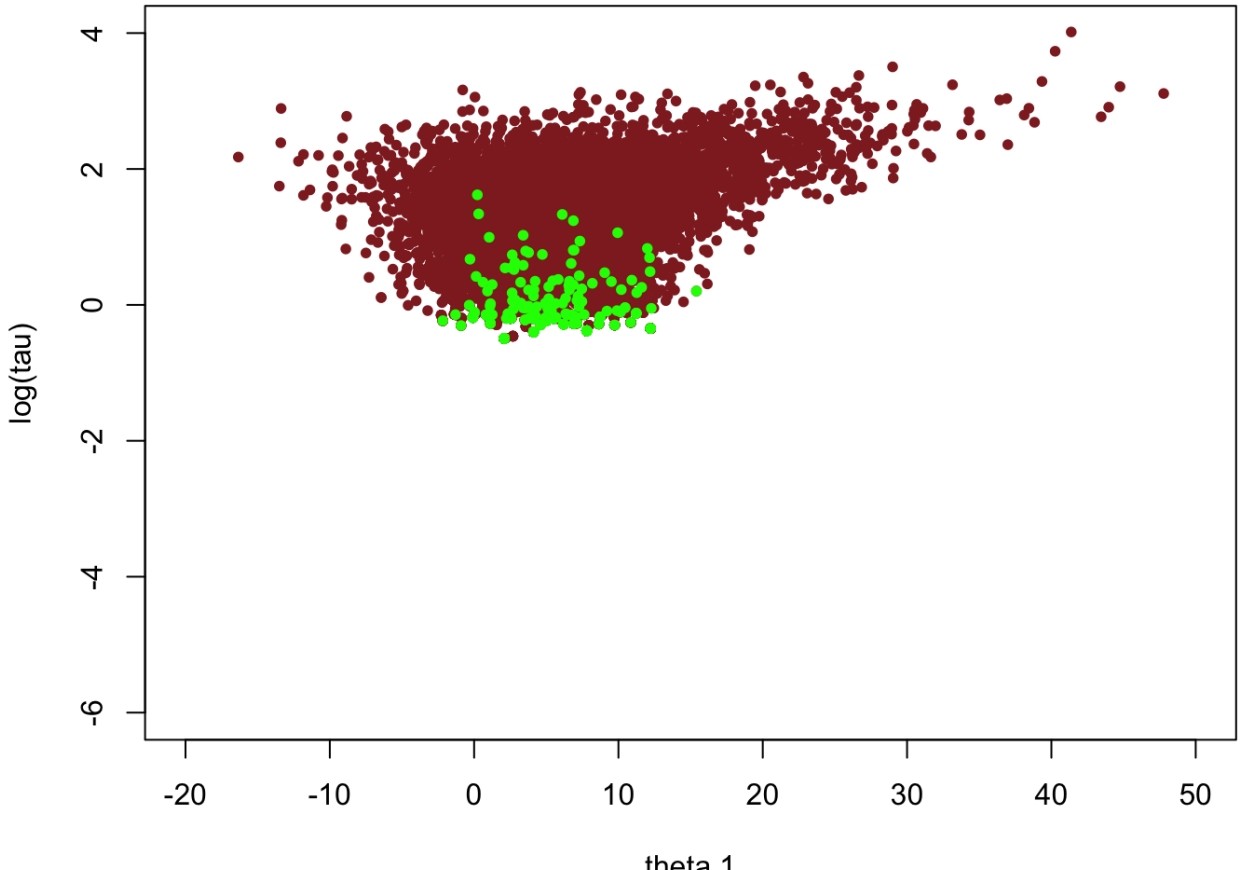
Divergences

- HMC specific: indicates that Hamiltonian dynamic simulation has problems with unexpected fast changes in log-density
  - indicates possibility of biased estimates
- http://mc-stan.org/users/documentation/case-studies/divergences_and_bias.html



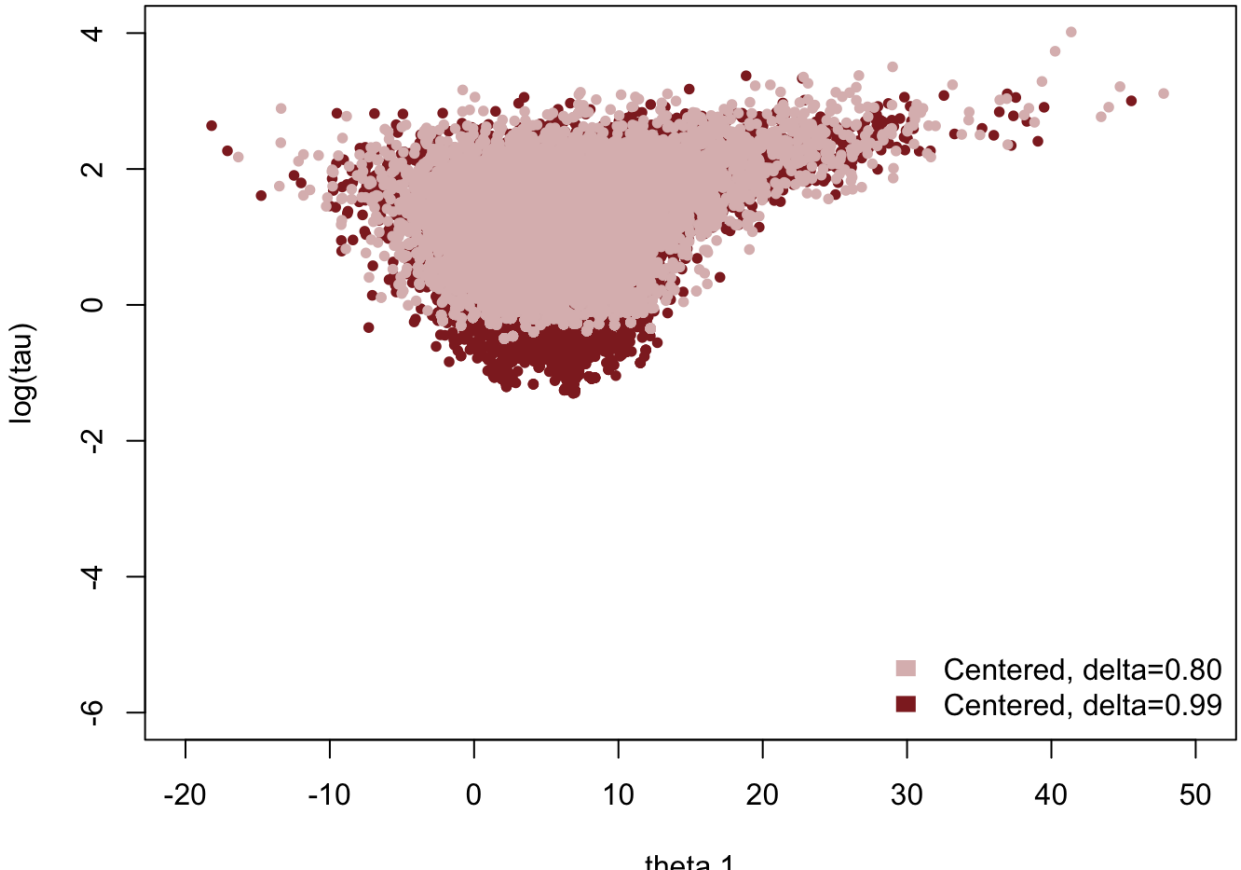
Divergences

- HMC specific: indicates that Hamiltonian dynamic simulation has problems with unexpected fast changes in log-density
  - indicates possibility of biased estimates
- http://mc-stan.org/users/documentation/case-studies/divergences_and_bias.html



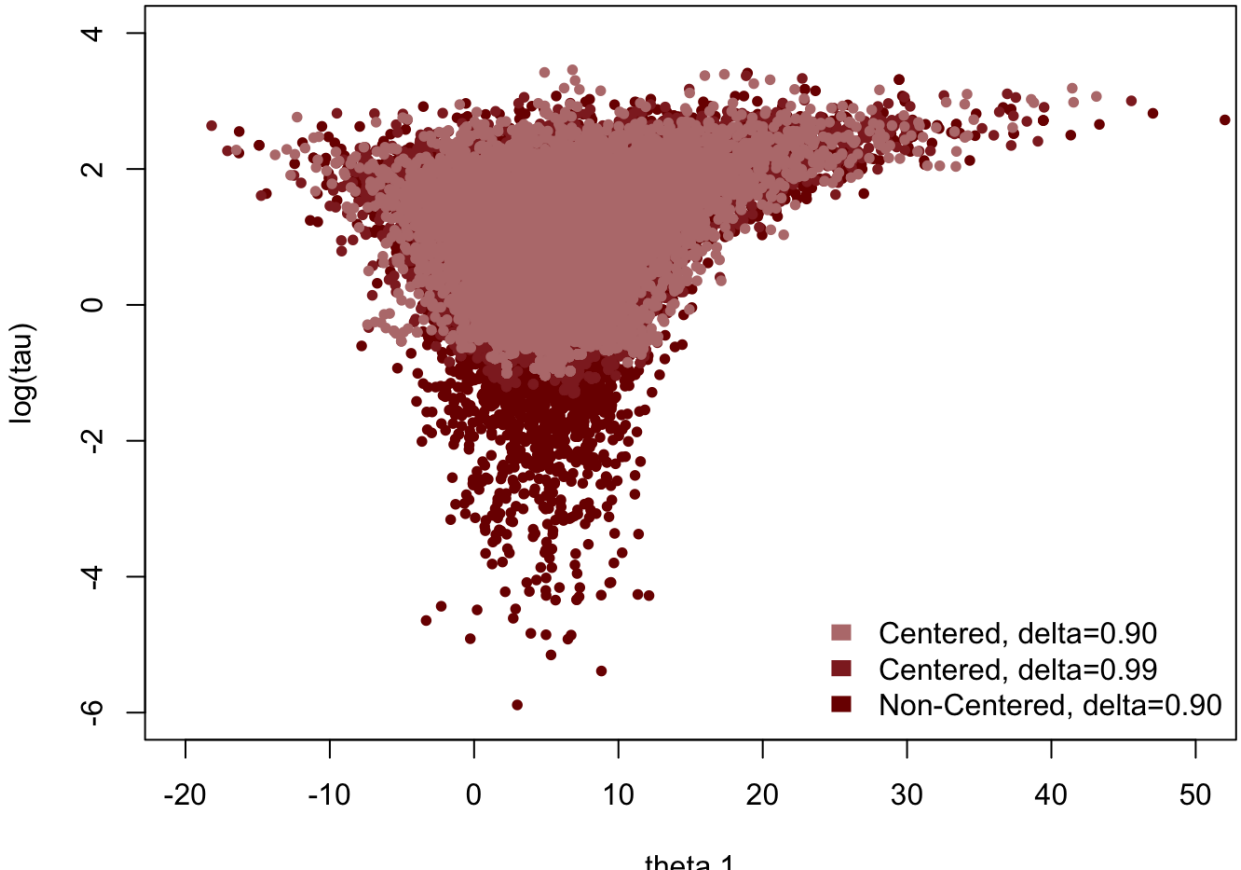
Divergences

- HMC specific: indicates that Hamiltonian dynamic simulation has problems with unexpected fast changes in log-density
  - indicates possibility of biased estimates
- http://mc-stan.org/users/documentation/case-studies/divergences_and_bias.html



Divergences

- HMC specific: indicates that Hamiltonian dynamic simulation has problems with unexpected fast changes in log-density
  - indicates possibility of biased estimates
- http://mc-stan.org/users/documentation/case-studies/divergences_and_bias.html



Problematic distributions

- Nonlinear dependencies
  - simple mass matrix scaling doesn't help

Problematic distributions

- Nonlinear dependencies
  - simple mass matrix scaling doesn't help
- Funnels
  - optimal step size depends on location

Problematic distributions

- Nonlinear dependencies
  - simple mass matrix scaling doesn't help
- Funnels
  - optimal step size depends on location
- Multimodal
  - difficult to move from one mode to another

Problematic distributions

- Nonlinear dependencies
  - simple mass matrix scaling doesn't help
- Funnels
  - optimal step size depends on location
- Multimodal
  - difficult to move from one mode to another
- Long-tailed with non-finite variance and mean
  - efficiency of exploration is reduced
  - central limit theorem doesn't hold for mean and variance

Probabilistic programming language

- Wikipedia “A probabilistic programming language (PPL) is a programming language designed to describe probabilistic models and then perform inference in those models”

Probabilistic programming language

- Wikipedia “A probabilistic programming language (PPL) is a programming language designed to describe probabilistic models and then perform inference in those models”
- To make probabilistic programming useful
  - inference has to be as automatic as possible
  - diagnostics for telling if the automatic inference doesn’t work
  - easy workflow (to reduce manual work)
  - fast enough (manual work replaced with automation)

Probabilistic programming

- Enables agile workflow for developing probabilistic models
  - language
  - automated inference
  - diagnostics
- Many frameworks Stan, PyMC, Pyro (Uber), TFP (Google), Turing.jl, JAGS, ELFI, ...

Stan - probabilistic programming framework

- Language, inference engine, user interfaces, documentation, case studies, diagnostics, packages, ...
  - autodiff to compute gradients of the log density

mc-stan.org

Stan - probabilistic programming framework

- Language, inference engine, user interfaces, documentation, case studies, diagnostics, packages, ...
  - autodiff to compute gradients of the log density
- More than 100K users in social, biological, and physical sciences, medicine, engineering, and business

mc-stan.org

Stan - probabilistic programming framework

- Language, inference engine, user interfaces, documentation, case studies, diagnostics, packages, ...
  - autodiff to compute gradients of the log density
- More than 100K users in social, biological, and physical sciences, medicine, engineering, and business
- Several full time developers, 40+ developers, more than 100 contributors

mc-stan.org

Stan - probabilistic programming framework

- Language, inference engine, user interfaces, documentation, case studies, diagnostics, packages, ...
  - autodiff to compute gradients of the log density
- More than 100K users in social, biological, and physical sciences, medicine, engineering, and business
- Several full time developers, 40+ developers, more than 100 contributors
- R, Python, Julia, Scala, Stata, Matlab, command line interfaces
- More than 100 R packages using Stan

mc-stan.org

Stan

- Stanislaw Ulam (1909-1984)
  - Monte Carlo method
  - H-Bomb

Binomial model - Stan code

```
data {
  int<lower=0> N;        // number of experiments
  int<lower=0,upper=N> y; // number of successes
}

parameters {
  real<lower=0,upper=1> theta; // parameter of the binom
}

model {
  theta ~ beta(1,1);      //prior
  y ~ binomial(N,theta); // observation model
}
```

Binomial model - Stan code

```
data {
  int<lower=0> N;      // number of experiments
  int<lower=0,upper=N> y; // number of successes
}

parameters {
  real<lower=0,upper=1> theta; // parameter of the binom
}

model {
  theta ~ beta(1,1);      //prior
  y ~ binomial(N,theta); // observation model
}
```

Binomial model - Stan code

```
data {
  int<lower=0> N;      // number of experiments
  int<lower=0,upper=N> y; // number of successes
}

parameters {
  real<lower=0,upper=1> theta; // parameter of the binom
}

model {
  theta ~ beta(1,1);      //prior
  y ~ binomial(N,theta); // observation model
}
```

Binomial model - Stan code

```
data {
  int<lower=0> N;        // number of experiments
  int<lower=0,upper=N> y; // number of successes
}
```

- Data type and size are declared
- Stan checks that given data matches type and constraints

Binomial model - Stan code

```
data {
  int<lower=0> N;       // number of experiments
  int<lower=0,upper=N> y; // number of successes
}
```

- Data type and size are declared
- Stan checks that given data matches type and constraints
  - If you are not used to strong typing, this may feel annoying, but it will reduce the probability of coding errors, which will reduce probability of data analysis errors

Binomial model - Stan code

```
parameters {
  real<lower=0,upper=1> theta;
}
```

- Parameters may have constraints
- Stan makes transformation to unconstrained space and samples in unconstrained space
  - e.g. log transformation for <lower=a>
  - e.g. logit transformation for <lower=a,upper=b>

Binomial model - Stan code

```
parameters {
  real<lower=0,upper=1> theta;
}
```

- Parameters may have constraints
- Stan makes transformation to unconstrained space and samples in unconstrained space
  - e.g. log transformation for <lower=a>
  - e.g. logit transformation for <lower=a,upper=b>
- For these declared transformation Stan automatically takes into account the Jacobian of the transformation (see BDA3 p. 21)

Binomial model - Stan code

```
model {
  theta ~ beta(1,1);       // prior
  y ~ binomial(N,theta);   // likelihood
}
```

Binomial model - Stan code

```
model {
  theta ~ beta(1,1);       // prior
  y ~ binomial(N,theta); // likelihood
}
```

- $\sim$ is syntactic sugar and this is equivalent to

```
model {
  target += beta_lpdf(theta | 1, 1);
  target += binomial_lpmf(y | N, theta);
}
```

Binomial model - Stan code

```
model {
  theta ~ beta(1,1);       // prior
  y ~ binomial(N,theta); // likelihood
}
```

- $\sim$ is syntactic sugar and this is equivalent to

```
model {
  target += beta_lpdf(theta | 1, 1);
  target += binomial_lpmf(y | N, theta);
}
```

- `target` is the log posterior density

Binomial model - Stan code

```
model {
  theta ~ beta(1,1);      // prior
  y ~ binomial(N,theta); // likelihood
}
```

- $\sim$ is syntactic sugar and this is equivalent to

```
model {
  target += beta_lpdf(theta | 1, 1);
  target += binomial_lpmf(y | N, theta);
}
```

- `target` is the log posterior density
- `_lpdf` for continuous, `_lpmf` for discrete distributions (discrete for the left hand side of `|`)

Binomial model - Stan code

```
model {
  theta ~ beta(1,1);      // prior
  y ~ binomial(N,theta); // likelihood
}
```

- $\sim$ is syntactic sugar and this is equivalent to

```
model {
  target += beta_lpdf(theta | 1, 1);
  target += binomial_lpmf(y | N, theta);
}
```

- `target` is the log posterior density
- `_lpdf` for continuous, `_lpmf` for discrete distributions (discrete for the left hand side of `|`)
- for Stan sampler there is no difference between prior and likelihood, all that matters is the final `target`

Binomial model - Stan code

```
model {
  theta ~ beta(1,1);      // prior
  y ~ binomial(N,theta); // likelihood
}
```

- $\sim$ is syntactic sugar and this is equivalent to

```
model {
  target += beta_lpdf(theta | 1, 1);
  target += binomial_lpmf(y | N, theta);
}
```

- `target` is the log posterior density
- `_lpdf` for continuous, `_lpmf` for discrete distributions (discrete for the left hand side of `|`)
- for Stan sampler there is no difference between prior and likelihood, all that matters is the final `target`
- you can write in Stan language any program to compute the log density (Stan language is Turing complete)

Stan

- Stan compiles (transplies) the model written in Stan language to C++
  - this makes the sampling for complex models and bigger data faster
  - also makes Stan models easily portable, you can use your own favorite interface

RStan

RStan

```
library(rstan)
rstan_options(auto_write = TRUE)
options(mc.cores = parallel::detectCores())

d_bin <- list(N = 10, y = 7)
fit_bin <- stan(file = 'binom.stan', data = d_bin)
```

RStan

RStan

```
library(rstan)
rstan_options(auto_write = TRUE)
options(mc.cores = parallel::detectCores())

d_bin <- list(N = 10, y = 7)
fit_bin <- stan(file = 'binom.stan', data = d_bin)
```

PyStan

PyStan

```
import pystan
import stan_utility

data = dict(N=10, y=8)
model = stan_utility.compile_model('binom.stan')
fit = model.sampling(data=data)
```

PyStan

PyStan

```
import pystan
import stan_utility

data = dict(N=10, y=8)
model = stan_utility.compile_model('binom.stan')
fit = model.sampling(data=data)
```

Stan

- Compilation (unless previously compiled model available)
- Warm-up including adaptation
- Sampling
- Generated quantities
- Save posterior draws
- Report divergences, n_{eff}, $\widehat{R}$

Difference between proportions

- An experiment was performed to estimate the effect of beta-blockers on mortality of cardiac patients
- A group of patients were randomly assigned to treatment and control groups:
  - out of 674 patients receiving the control, 39 died
  - out of 680 receiving the treatment, 22 died

Difference between proportions

```
data {
  int<lower=0> N1;
  int<lower=0> y1;
  int<lower=0> N2;
  int<lower=0> y2;
}
parameters {
  real<lower=0,upper=1> theta1;
  real<lower=0,upper=1> theta2;
}
model {
  theta1 ~ beta(1,1);
  theta2 ~ beta(1,1);
  y1 ~ binomial(N1,theta1);
  y2 ~ binomial(N2,theta2);
}

generated quantities {
  real oddsratio;
  oddsratio = (theta2/(1-theta2))/(theta1/(1-theta1));
}
```

Difference between proportions

```
data {
  int<lower=0> N1;
  int<lower=0> y1;
  int<lower=0> N2;
  int<lower=0> y2;
}
parameters {
  real<lower=0,upper=1> theta1;
  real<lower=0,upper=1> theta2;
}
model {
  theta1 ~ beta(1,1);
  theta2 ~ beta(1,1);
  y1 ~ binomial(N1,theta1);
  y2 ~ binomial(N2,theta2);
}

generated quantities {
  real oddsratio;
  oddsratio = (theta2/(1-theta2))/(theta1/(1-theta1));
}
```

Difference between proportions

```
generated quantities {
  real oddsratio;
  oddsratio = (theta2/(1-theta2))/(theta1/(1-theta1));
}
```

- generated quantities is run after the sampling

Difference between proportions

```
d_bin2 <- list(N1 = 674, y1 = 39, N2 = 680, y2 = 22)
fit_bin2 <- stan(file = 'binom2.stan', data = d_bin2)
```

```
starting worker pid=10151 on localhost:11783 at 10:03:27.872
starting worker pid=10164 on localhost:11783 at 10:03:28.087
starting worker pid=10176 on localhost:11783 at 10:03:28.295
starting worker pid=10185 on localhost:11783 at 10:03:28.461

SAMPLING FOR MODEL 'binom2' NOW (CHAIN 1).

Gradient evaluation took 6e-06 seconds
1000 transitions using 10 leapfrog steps per transition would take 0.06 seconds.
Adjust your expectations accordingly!


Iteration:    1 / 2000 [  0%]  (Warmup)
Iteration:  200 / 2000 [ 10%]  (Warmup)
...
Iteration: 1000 / 2000 [ 50%]  (Warmup)
Iteration: 1001 / 2000 [ 50%]  (Sampling)
...
Iteration: 2000 / 2000 [100%]  (Sampling)

 Elapsed Time: 0.012908 seconds (Warm-up)
               0.017027 seconds (Sampling)
               0.029935 seconds (Total)


SAMPLING FOR MODEL 'binom2' NOW (CHAIN 2).
...
```

Difference between proportions

```
monitor(fit_bin2, probs = c(0.1, 0.5, 0.9))
```

```
Inference for the input samples
(4 chains: each with iter=1000; warmup=0):
```

	mean	se_mean	sd	10%	50%	90%	n_eff	Rhat
theta1	0.1	0	0.0	0.0	0.1	0.1	3280	1
theta2	0.0	0	0.0	0.0	0.0	0.0	3171	1
oddsratio	0.6	0	0.2	0.4	0.6	0.8	3108	1
lp__	−253.5	0	1.0	−254.8	−253.2	−252.6	1922	1

```
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).
```

Difference between proportions

```
monitor(fit_bin2, probs = c(0.1, 0.5, 0.9))
```

```
Inference for the input samples
(4 chains: each with iter=1000; warmup=0):
```

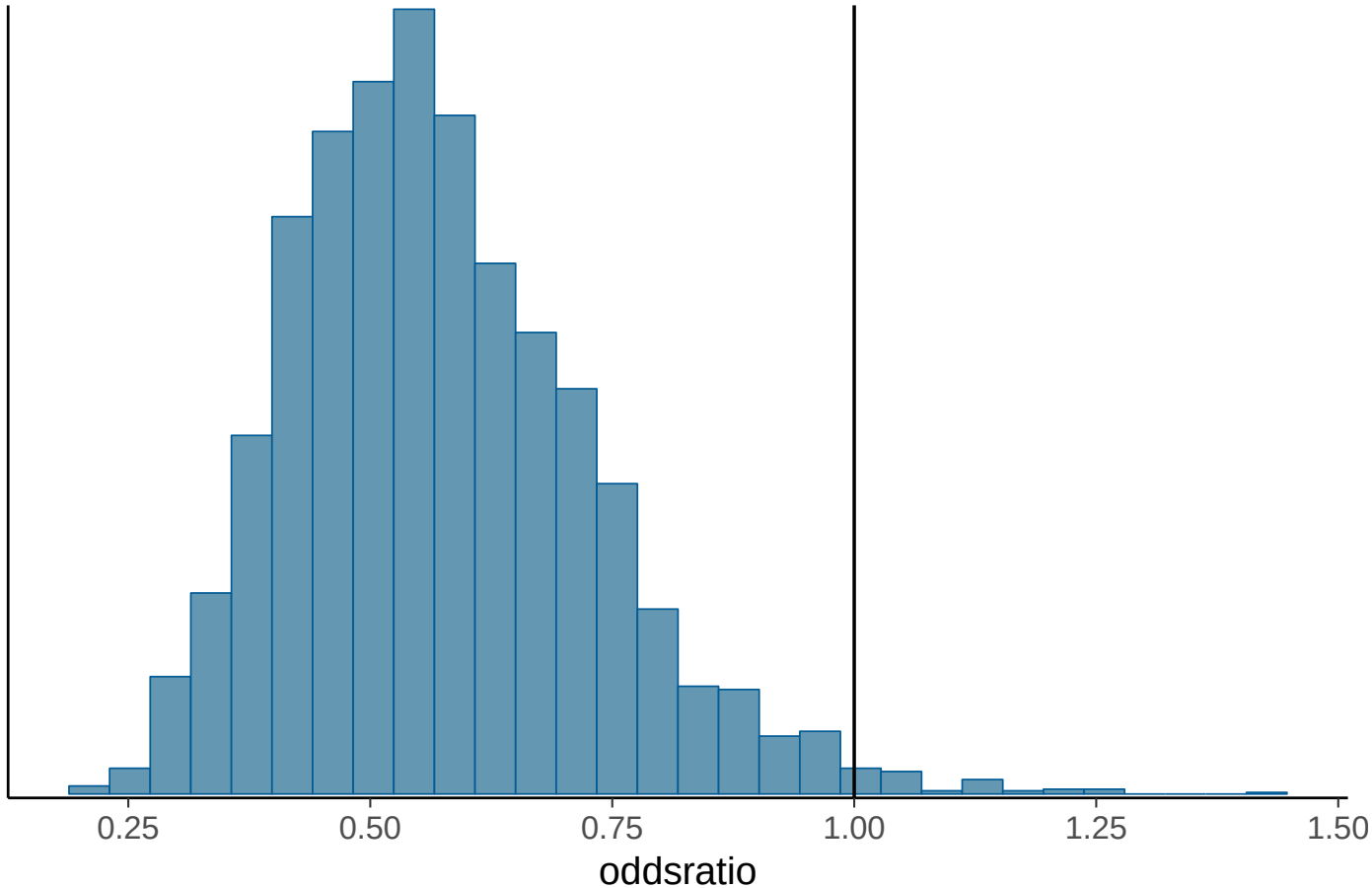
	mean	se_mean	sd	10%	50%	90%	n_eff	Rhat
theta1	0.1	0	0.0	0.0	0.1	0.1	3280	1
theta2	0.0	0	0.0	0.0	0.0	0.0	3171	1
oddsratio	0.6	0	0.2	0.4	0.6	0.8	3108	1
lp__	−253.5	0	1.0	−254.8	−253.2	−252.6	1922	1

```
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).
```

- `lp__` is the log density, ie, same as `target`

Difference between proportions

```
draws <- as.data.frame(fit_bin2)
mcmc_hist(draws, pars = 'oddsratio') +
  geom_vline(xintercept = 1) +
  scale_x_continuous(breaks = c(seq(0.25,1.5,by=0.25)))
```



HMC specific diagnostics

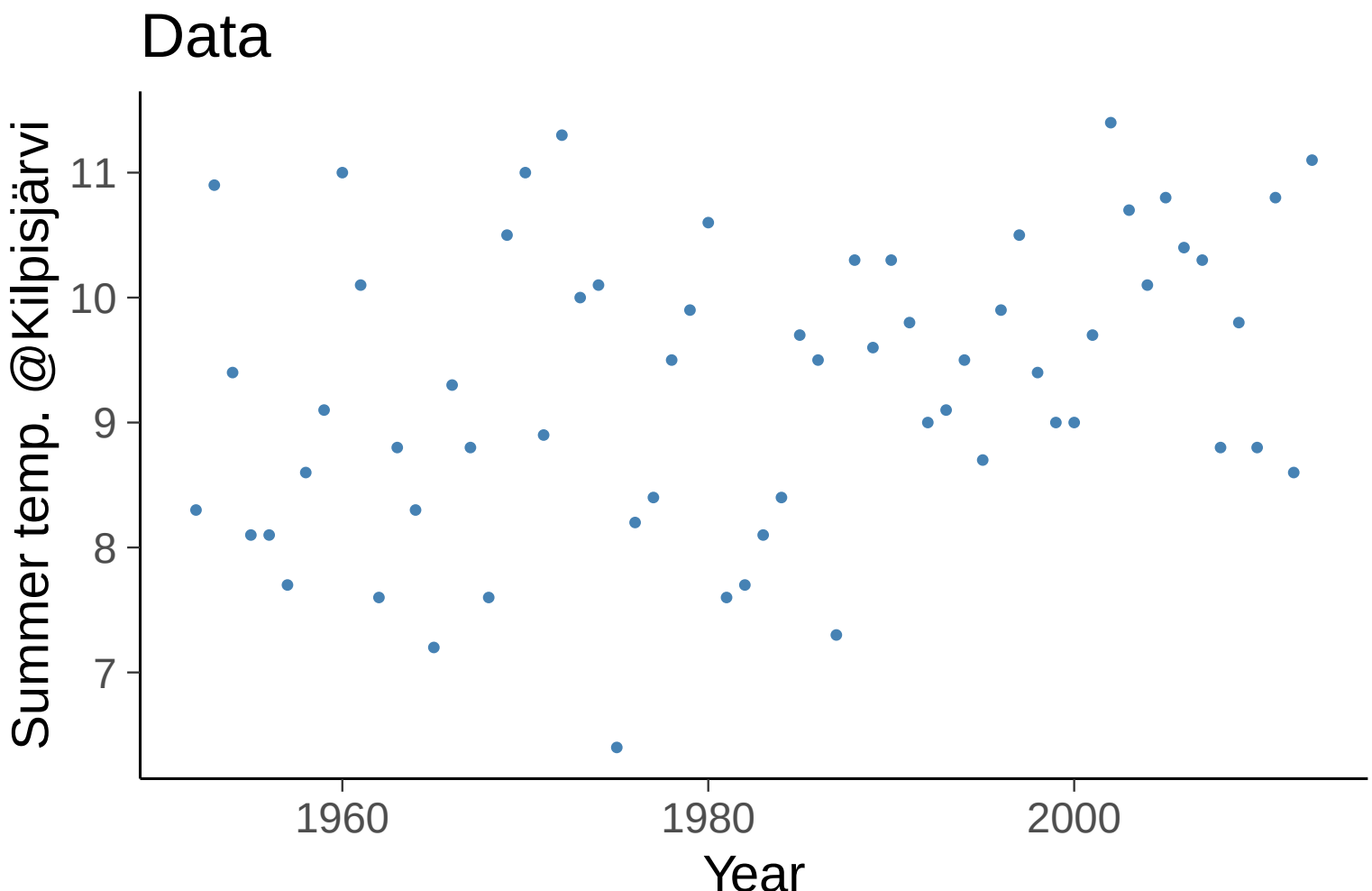
```
check_treedepth(fit_bin2)
check_energy(fit_bin2)
check_div(fit_bin2)

[1] "0 of 4000 iterations saturated the maximum tree depth of 10 (0%)"
[1] "0 of 4000 iterations ended with a divergence (0%)"
```

Shinystan

- Graphical user interface for analysing MCMC results

Kilpisjärvi summer temperature

- Temperature at Kilpisjärvi in June, July and August from 1952 to 2013
- Is there change in the temperature?



Gaussian linear model

```
data {
    int<lower=0> N; // number of data points
    vector[N] x; //
    vector[N] y; //
}
parameters {
    real alpha;
    real beta;
    real<lower=0> sigma;
}
transformed parameters {
    vector[N] mu;
    mu <- alpha + beta*x;
}
model {
    y ~ normal(mu, sigma);
}
```

Gaussian linear model

```
data {
    int<lower=0> N; // number of data points
    vector[N] x; //
    vector[N] y; //
}
```

- difference between `vector[N] x` and `real x[N]`

Gaussian linear model

```
parameters {
      real alpha;
      real beta;
      real<lower=0> sigma;
}
transformed parameters {
      vector[N] mu;
      mu <- alpha + beta*x;
}
```

- transformed parameters are deterministic transformations of parameters and data

Priors for Gaussian linear model

```
data {
    int<lower=0> N; // number of data points
    vector[N] x; //
    vector[N] y; //
    real pmualpha; // prior mean for alpha
    real psalpha;  // prior std for alpha
    real pmubeta;  // prior mean for beta
    real psbeta;   // prior std for beta
}
...
transformed parameters {
    vector[N] mu;
    mu <- alpha + beta*x;
}
model {
    alpha ~ normal(pmualpha,psalpha);
    beta ~ normal(pmubeta,psbeta);
    y ~ normal(mu, sigma);
}
```

Student-t linear model

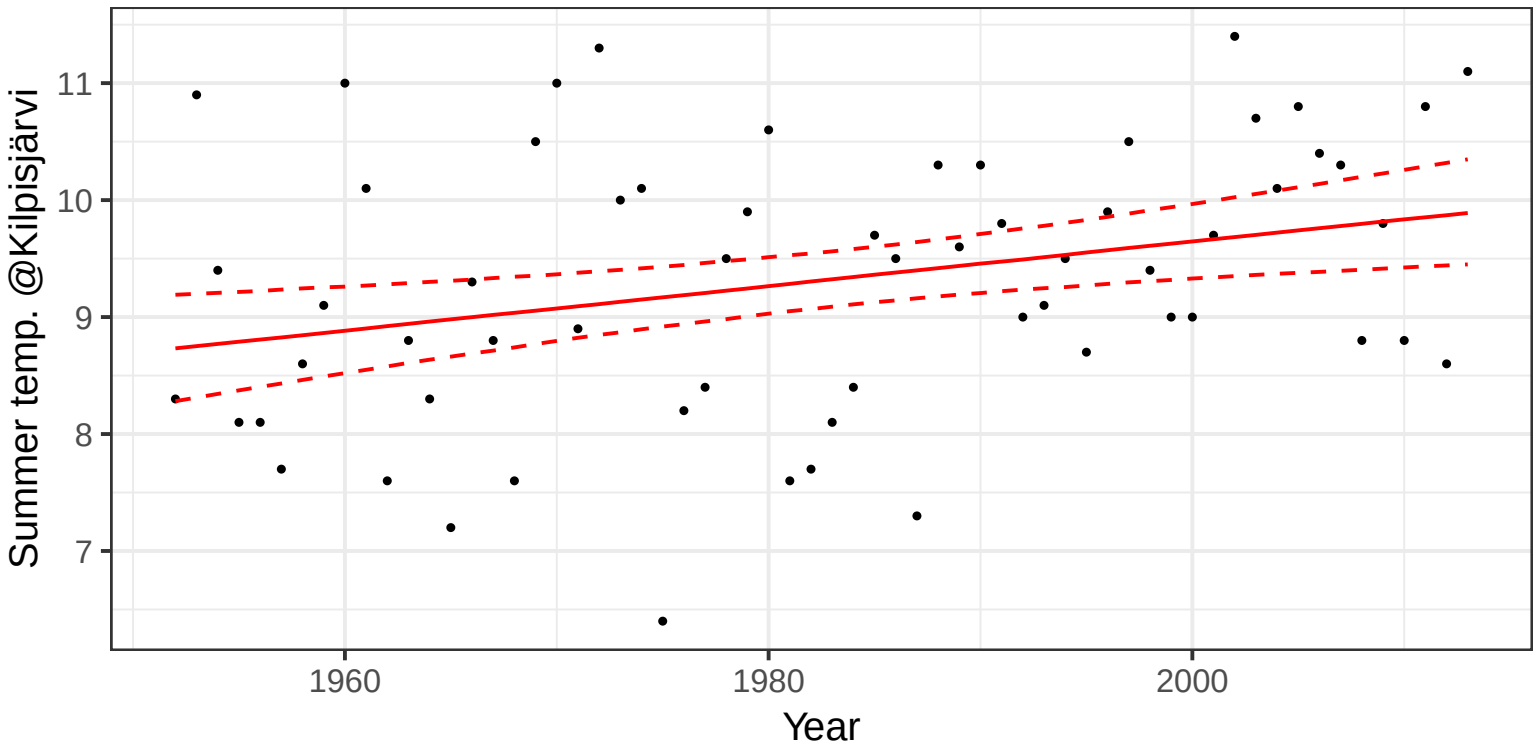
```
...
parameters {
  real alpha;
  real beta;
  real<lower=0> sigma;
  real<lower=1,upper=80> nu;
}
transformed parameters {
  vector[N] mu;
  mu <- alpha + beta*x;
}
model {
  nu ~ gamma(2,0.1);
  y ~ student_t(nu, mu, sigma);
}
```

Priors

- Prior for temperature increase?

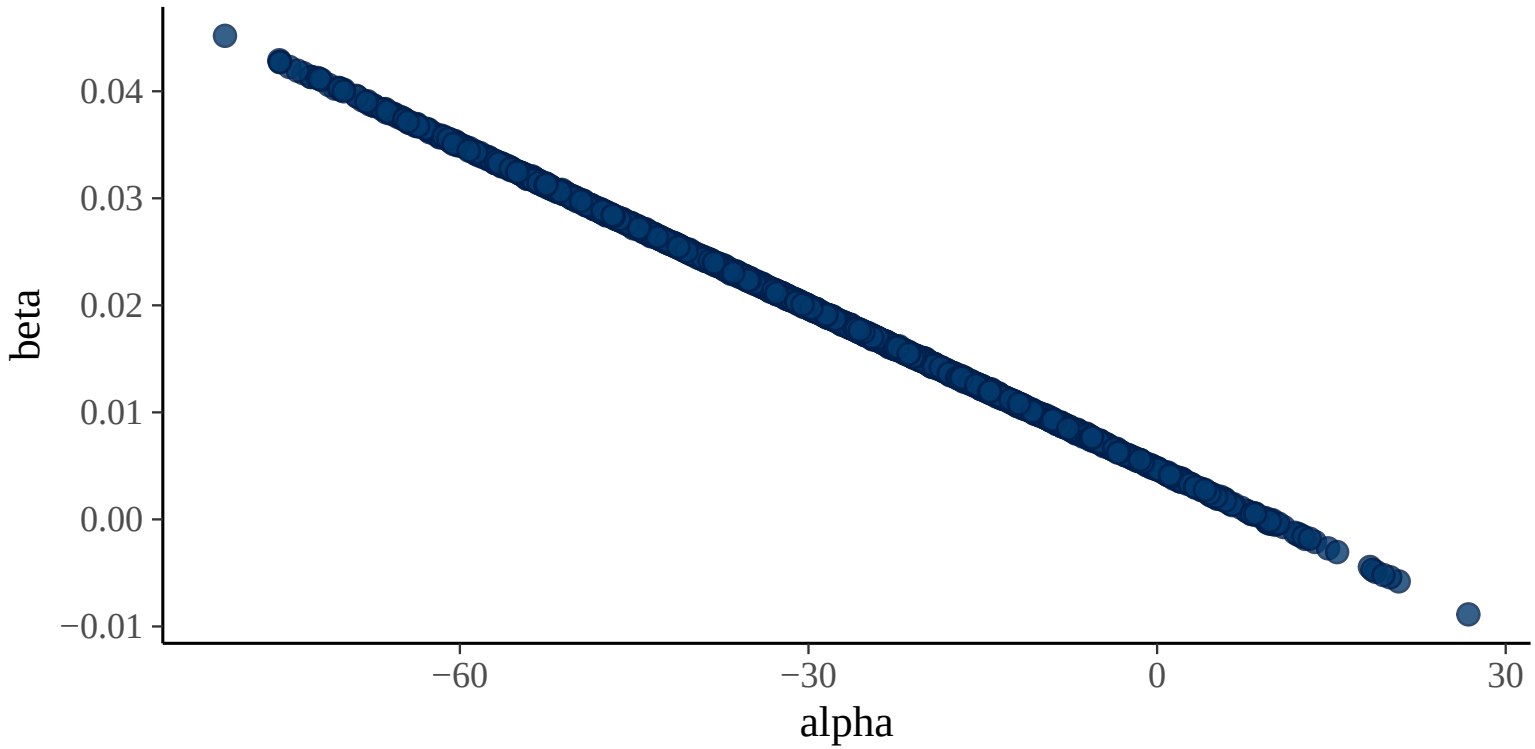
Kilpisjärvi summer temperature

Posterior fit



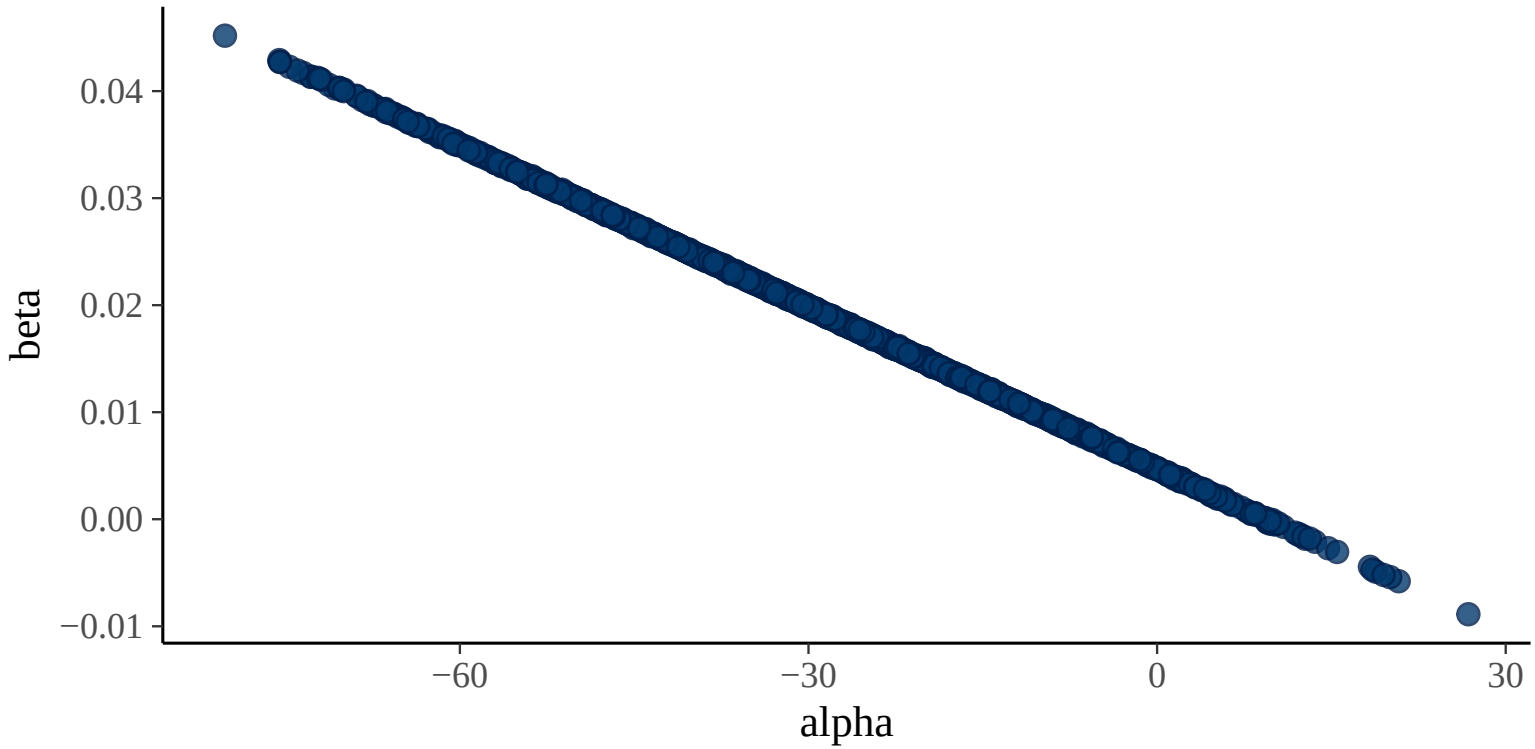
Kilpisjärvi summer temperature

Posterior draws of alpha and beta



Kilpisjärvi summer temperature

Posterior draws of alpha and beta



There were 14 transitions after warmup that exceeded the maximum treedepth. Increase max_treedepth above 10. See http://mc-stan.org/misc/warnings.html#maximum-treedepth-exceeded
Examine the pairs() plot to diagnose sampling problems

Linear regression model in Stan

```
data {
  int<lower=0> N; // number of data points
  vector[N] x; //
  vector[N] y; //
  real xpred; // input location for prediction
}
transformed data {
  vector[N] x_std;
  vector[N] y_std;
  real xpred_std;
  x_std = (x - mean(x)) / sd(x);
  y_std = (y - mean(y)) / sd(y);
  xpred_std = (xpred - mean(x)) / sd(x);
}
```

RStanARM

- RStanARM provides simplified model description with pre-compiled models
  - no need to wait for compilation
  - a restricted set of models

Two group Binomial model:

```
d_bin2 <- data.frame(N = c(674, 680), y = c(39,22), grp2 = c(0,1))
fit_bin2 <- stan_glm(y/N ~ grp2, family = binomial(), data = d_bin2,
                     weights = N)
```

RStanARM

- RStanARM provides simplified model description with pre-compiled models
  - no need to wait for compilation
  - a restricted set of models

Two group Binomial model:

```
d_bin2 <- data.frame(N = c(674, 680), y = c(39,22), grp2 = c(0,1))
fit_bin2 <- stan_glm(y/N ~ grp2, family = binomial(), data = d_bin2,
                     weights = N)
```

Gaussian linear model

```
fit_lin <- stan_glm(temp ~ year, data = d_lin)
```

BRMS

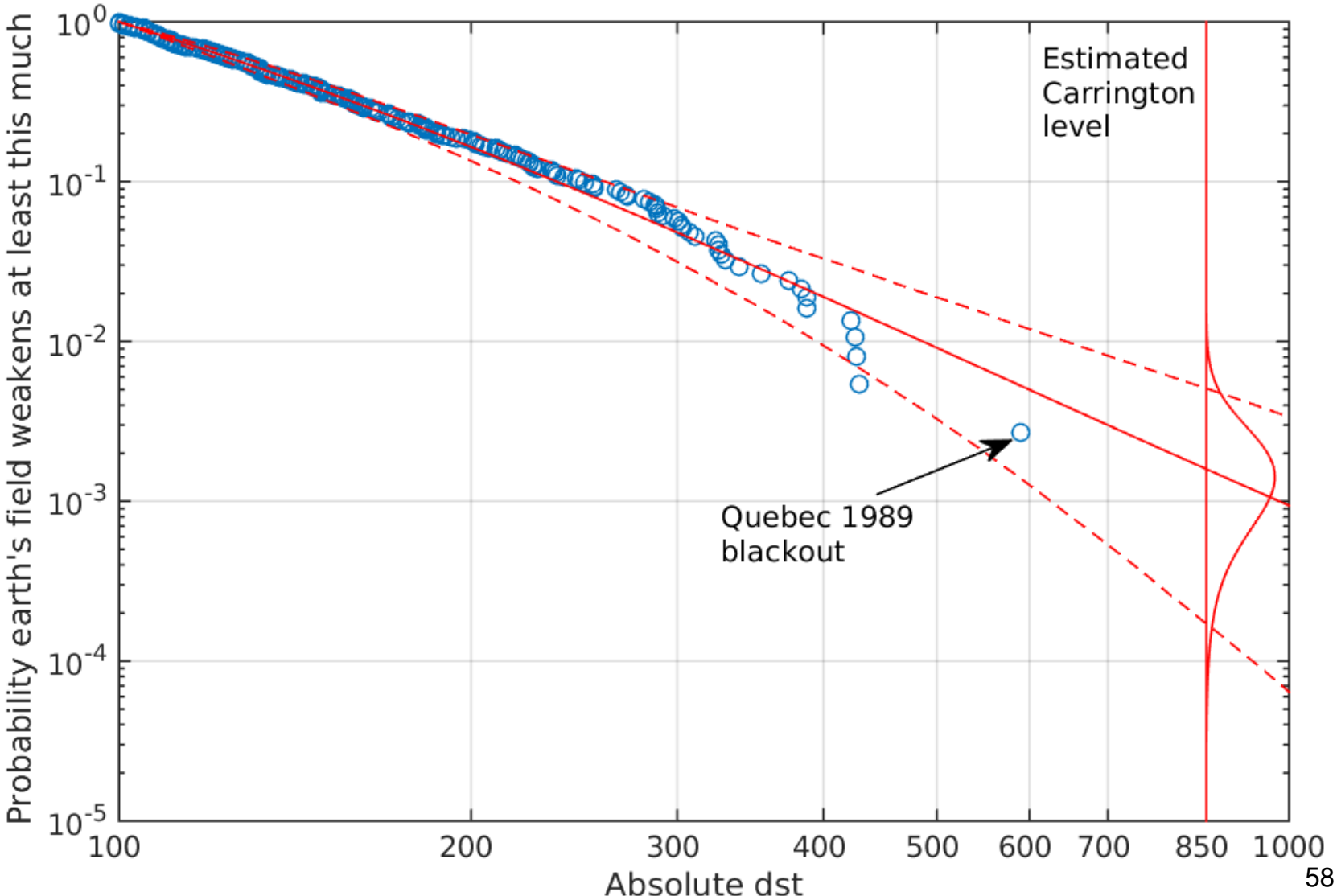
- BRMS provides simplified model description
  - a larger set of models than RStanARM, but still restricted
  - need to wait for the compilation

```
fit_bin2 <- brm(y/N ~ grp2, family = binomial(), data = d_bin2,
                weights = N)
```

```
fit_lin_t <- brm(temp ~ year, data = d_lin, family = student())
```

Extreme value analysis

Geomagnetic storms



Extreme value analysis

```
data {
  int<lower=0> N;
  vector<lower=0>[N] y;
  int<lower=0> Nt;
  vector<lower=0>[Nt] yt;
}
transformed data {
  real ymax;
  ymax <- max(y);
}
parameters {
  real<lower=0> sigma;
  real<lower=-sigma/ymax> k;
}
model {
  y ~ gpareto(k, sigma);
}
generated quantities {
  vector[Nt] predccdf;
  predccdf<-gpareto_ccdf(yt,k,sigma);
```

Functions

```
functions {
  real gpareto_lpdf(vector y, real k, real sigma) {
    // generalised Pareto log pdf with mu=0
    // should check and give error if k<0
    // and max(y)/sigma > -1/k
    int N;
    N <- dims(y)[1];
    if (fabs(k) > 1e-15)
      return -(1+1/k)*sum(log1pv(y*k/sigma)) -N*log(sigma);
    else
      return -sum(y/sigma) -N*log(sigma); // limit k->0
  }
  vector gpareto_ccdf(vector y, real k, real sigma) {
    // generalised Pareto log ccdf with mu=0
    // should check and give error if k<0
    // and max(y)/sigma < -1/k
    if (fabs(k) > 1e-15)
      return exp((-1/k)*log1pv(y/sigma*k));
    else
      return exp(-y/sigma); // limit k->0
  }
}
```

Other packages

- R
  - shinystan — interactive diagnostics
  - bayesplot — visualization and model checking (see model checking in Ch 6)
  - loo — cross-validation model assessment, comparison and averaging (see Ch 7)
  - projpred — projection predictive variable selection

- Python
  - ArviZ — visualization, and model checking and assessment (see Ch 6 and 7)

Different interfaces

- RStan / PyStan
  - C++ functions of Stan are called directly from R / Python
  - Higher integration between R/Python and Stan, but maybe more difficult to install due to more requirements of compatible C++ compilers and libraries
- CmdStanR / CmdStanPy
  - Lightweight interface on top of commandline program CmdStan
  - Lacks some features that are not needed in this course, but is usually easier to install
- More recent useful R packages
  - posterior: for handling posterior draws, convergence diagnostics, and summaries
  - tidybayes + ggdist: pretty plots