

## 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 dynamic HMC

- Michael Betancourt (2018). A Conceptual Introduction to Hamiltonian Monte Carlo. <https://arxiv.org/abs/1701.02434>
- 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>
- Michael Betancourt (2018). Scalable Bayesian Inference with Hamiltonian Monte Carlo  
<https://www.youtube.com/watch?v=jUSZboSq1zg>

## 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\\_jebbs\\_2.pdf](http://www.stat.columbia.edu/~gelman/research/published/stan_jebbs_2.pdf)
- Carpenter et al (2017). Stan: A probabilistic programming language. Journal of Statistical Software 76(1). <https://doi.org/10.18637/jss.v076.i01>
- Stan development team (2018). Modeling Language User's Guide and Reference Manual <https://github.com/stan-dev/stan/releases/download/v2.17.0/stan-reference-2.17.0.pdf>
  - easiest to start from Part III Example Models
- Basics of Bayesian inference and Stan, part 1 Jonah Gabry & Lauren Kennedy
  - <https://www.youtube.com/watch?v=ZRpo41I02KQ&index=6&list=PLuwyh42iHquU4hUBQs20hkBsKSMrp6H0J>
  - <https://www.youtube.com/watch?v=6cc4N1vT8pk&index=7&list=PLuwyh42iHquU4hUBQs20hkBsKSMrp6H0J>

## Chapter 12 demos

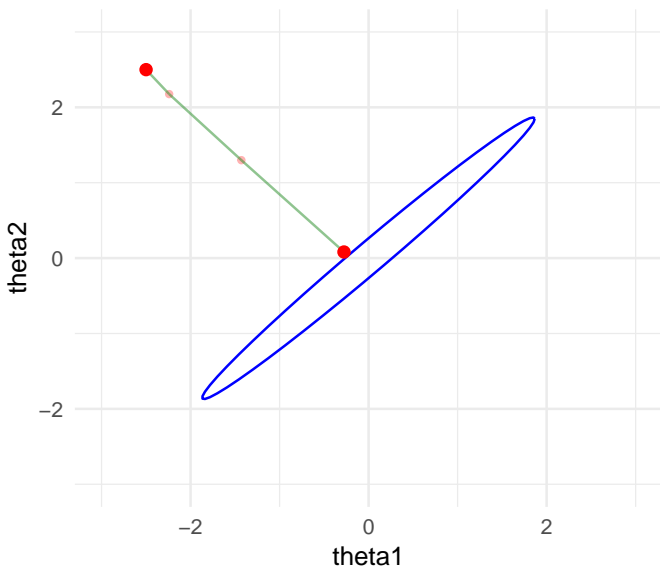
- demo12\_1: HMC
- <http://elevanth.org/blog/2017/11/28/build-a-better-markov-chain/>
- rstan\_demo
- rstanarm\_demo
- <http://sumsar.net/blog/2017/01/bayesian-computation-with-stan-and-farmer-jons/>
- <http://mc-stan.org/documentation/case-studies.html>
- <https://cran.r-project.org/package=rstan>
- <https://cran.r-project.org/package=rstanarm>

# Hamiltonian Monte Carlo

- Uses gradient information for more efficient sampling
- Augments parameter space with momentum variables

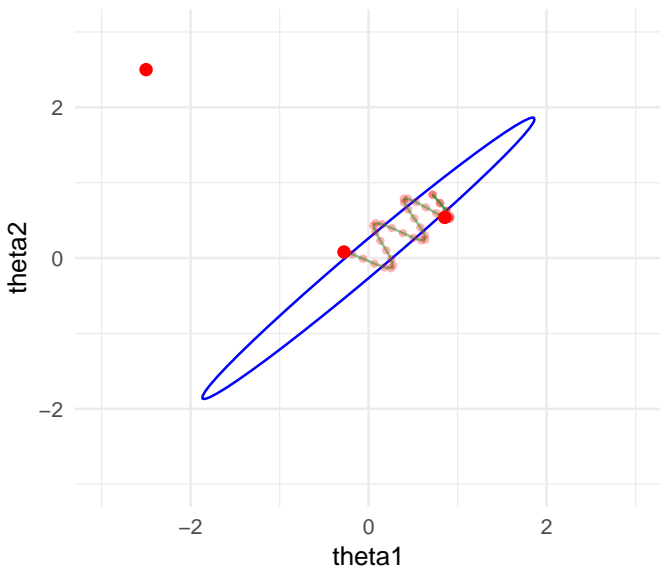
# Hamiltonian Monte Carlo

- Uses gradient information for more efficient sampling
- Augments parameter space with momentum variables



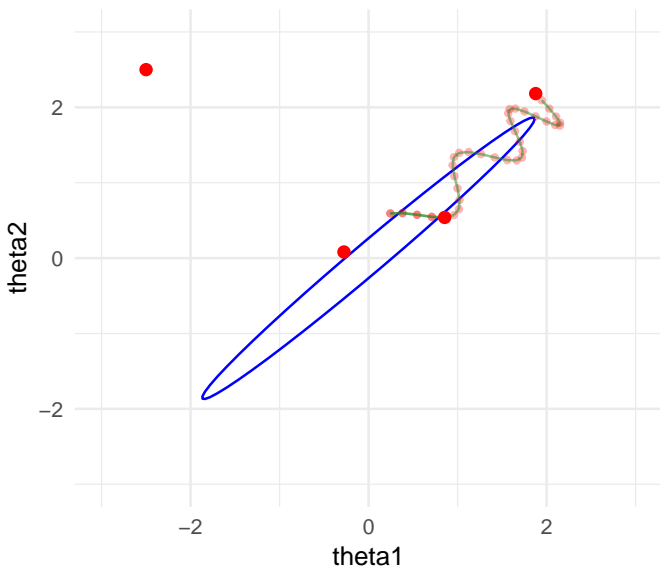
# Hamiltonian Monte Carlo

- Uses gradient information for more efficient sampling
- Augments parameter space with momentum variables



# Hamiltonian Monte Carlo

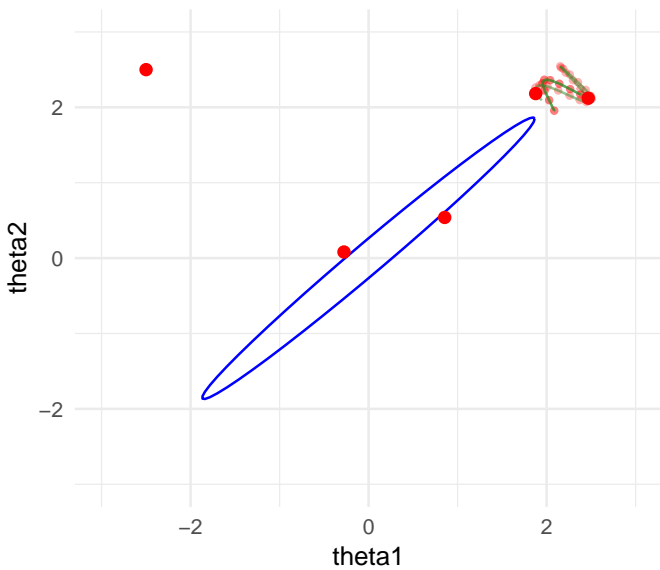
- Uses gradient information for more efficient sampling
- Augments parameter space with momentum variables





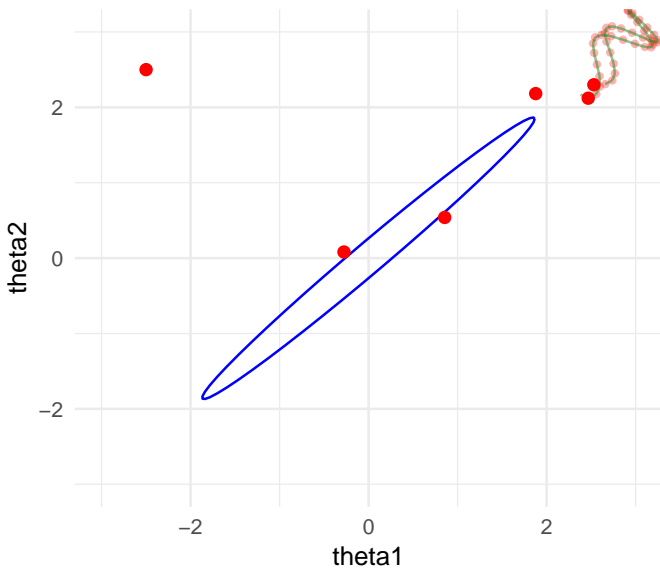
# Hamiltonian Monte Carlo

- Uses gradient information for more efficient sampling
- Augments parameter space with momentum variables



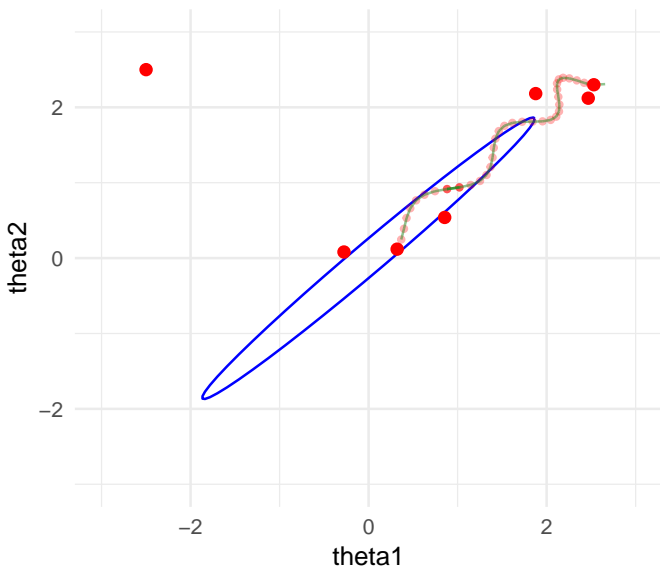
# Hamiltonian Monte Carlo

- Uses gradient information for more efficient sampling
- Augments parameter space with momentum variables



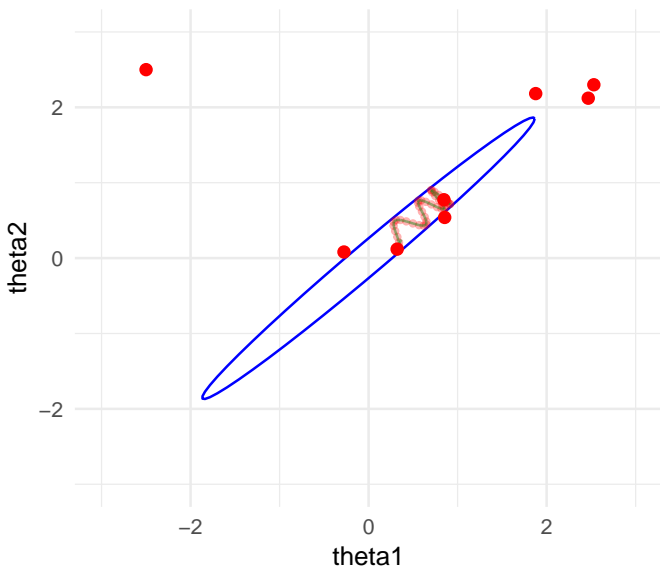
# Hamiltonian Monte Carlo

- Uses gradient information for more efficient sampling
- Augments parameter space with momentum variables



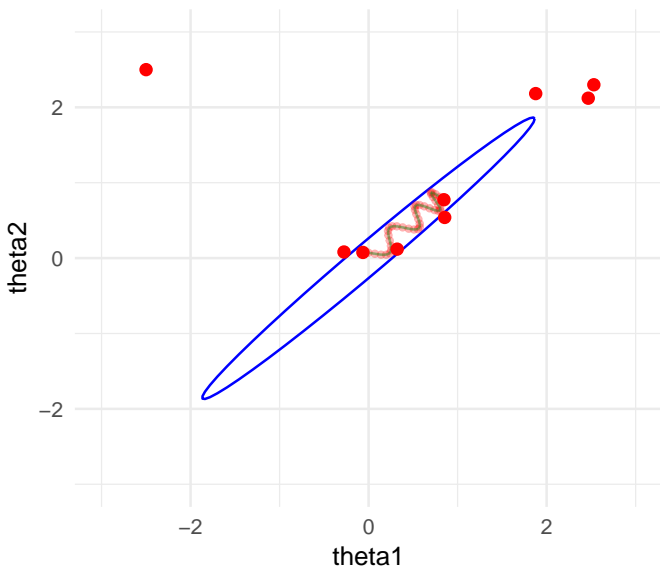
# Hamiltonian Monte Carlo

- Uses gradient information for more efficient sampling
- Augments parameter space with momentum variables



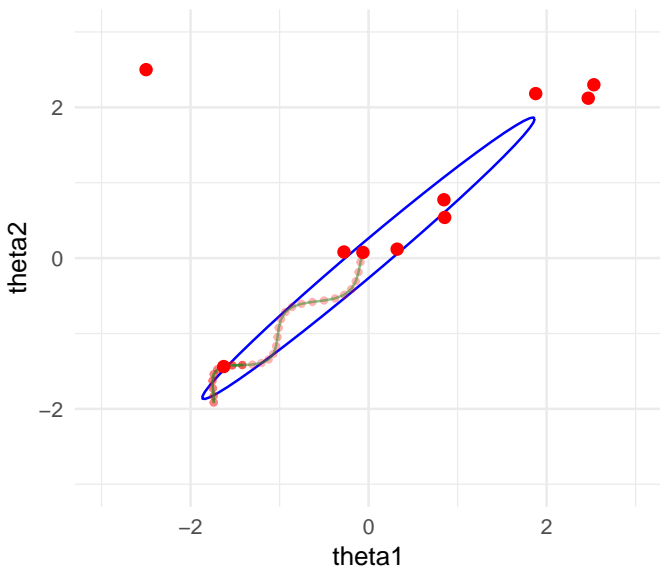
# Hamiltonian Monte Carlo

- Uses gradient information for more efficient sampling
- Augments parameter space with momentum variables



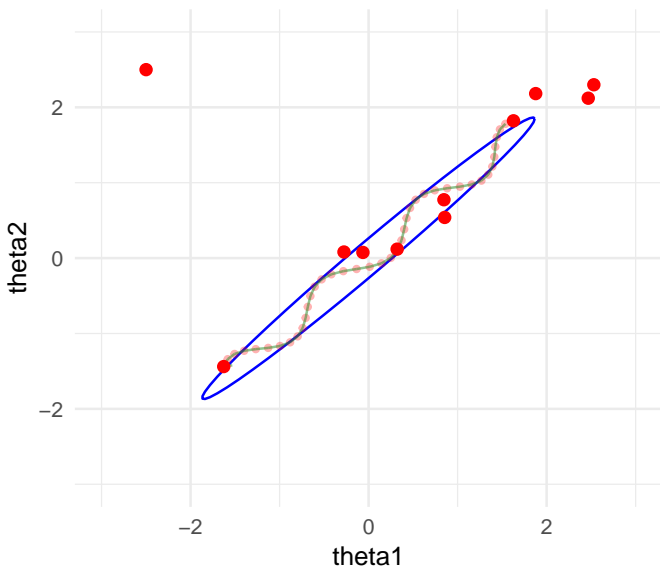
# Hamiltonian Monte Carlo

- Uses gradient information for more efficient sampling
- Augments parameter space with momentum variables



# Hamiltonian Monte Carlo

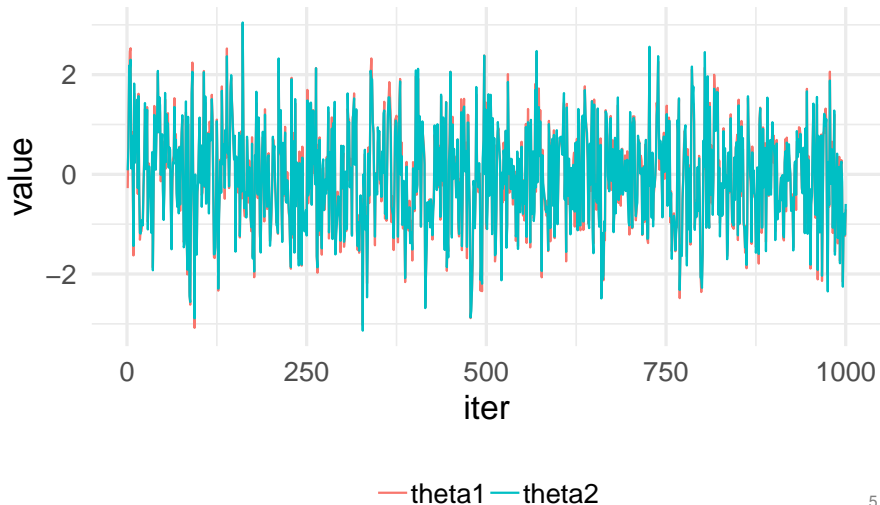
- Uses gradient information for more efficient sampling
- Augments parameter space with momentum variables



# Hamiltonian Monte Carlo

- Uses gradient information for more efficient sampling
- Augments parameter space with momentum variables

## Trends

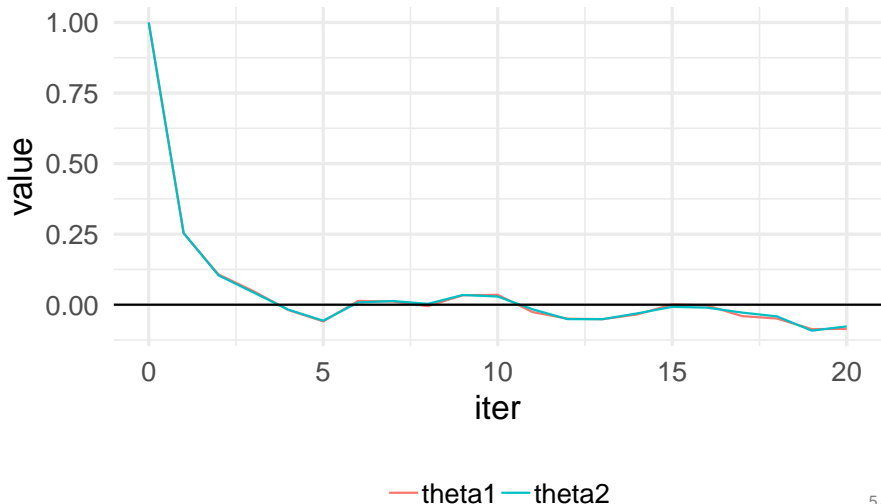




# Hamiltonian Monte Carlo

- Uses gradient information for more efficient sampling
- Augments parameter space with momentum variables

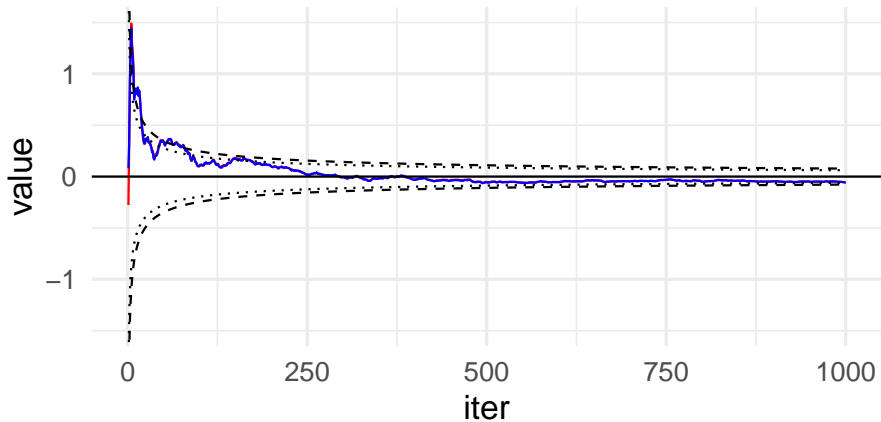
## Autocorrelation function



# Hamiltonian Monte Carlo

- Uses gradient information for more efficient sampling
- Augments parameter space with momentum variables

## Cumulative averages



-  $\theta_1$  —  $\theta_2$  - - 95% interval for MCMC error ··· 95% interval for independent error

# Hamiltonian Monte Carlo

- Uses gradient information for more efficient sampling
- Augments parameter space with momentum variables
- Simulation of Hamiltonian dynamics reduces random walk
  - Explanation of HMC with black board
  - <http://elevarth.org/blog/2017/11/28/build-a-better-markov-chain/>

# Hamiltonian Monte Carlo

- Uses gradient information for more efficient sampling
- Alternating dynamic simulation and sampling of the energy level

# Hamiltonian Monte Carlo

- Uses gradient information for more efficient sampling
- Alternating dynamic simulation and sampling of the energy level
- Parameters: step size, number of steps in each chain

# Hamiltonian Monte Carlo

- Uses gradient information for more efficient sampling
- Alternating dynamic simulation and sampling of the energy level
- Parameters: step size, number of steps in each chain
- No U-Turn Sampling (NUTS) and dynamic HMC
  - adaptively selects number of steps to improve robustness and efficiency
  - dynamic HMC refers to dynamic trajectory length
  - to keep reversibility of Markov chain, need to simulate on two direction
  - <http://elevanth.org/blog/2017/11/28/build-a-better-markov-chain/>

# Hamiltonian Monte Carlo

- Uses gradient information for more efficient sampling
- Alternating dynamic simulation and sampling of the energy level
- Parameters: step size, number of steps in each chain
- No U-Turn Sampling (NUTS) and dynamic HMC
  - adaptively selects number of steps to improve robustness and efficiency
  - dynamic HMC refers to dynamic trajectory length
  - to keep reversibility of Markov chain, need to simulate on two direction
  - <http://elevanth.org/blog/2017/11/28/build-a-better-markov-chain/>
- Dynamic simulation is discretized
  - small step size gives accurate simulation, but requires more log density evaluations
  - large step size reduces computation, but increases simulation error which needs to be taken into account in the Markov chain
  - black board explanation of the effect of step size

## Adaptive dynamic HMC in Stan

- Dynamic HMC using growing tree to increase simulation trajectory until no-U-turn criterion stopping
  - max treedepth to keep computation in control
  - pick a draw along the trajectory with probabilities adjusted to take into account the error in the discreteized dynamic simulation



# Adaptive dynamic HMC in Stan

- Dynamic HMC using growing tree to increase simulation trajectory until no-U-turn criterion stopping
  - max treedepth to keep computation in control
  - pick a draw along the trajectory with probabilities adjusted to take into account the error in the discreteized dynamic simulation
- Mass matrix and step size adaptation in Stan
  - mass matrix refers to having different scaling for different parameters and optionally also rotation to reduce correlations
  - mass matrix and step size adjustment and are estimated during initial adaptation phase
  - step size is adjusted to be as big as possible while keeping discretization error in control

# Adaptive dynamic HMC in Stan

- Dynamic HMC using growing tree to increase simulation trajectory until no-U-turn criterion stopping
  - max treedepth to keep computation in control
  - pick a draw along the trajectory with probabilities adjusted to take into account the error in the discreteized dynamic simulation
- Mass matrix and step size adaptation in Stan
  - mass matrix refers to having different scaling for different parameters and optionally also rotation to reduce correlations
  - mass matrix and step size adjustment and are estimated during initial adaptation phase
  - step size is adjusted to be as big as possible while keeping discretization error in control
- After adaptation the algorithm parameters are fixed and some further iterations included in the warmup

# Adaptive dynamic HMC in Stan

- Dynamic HMC using growing tree to increase simulation trajectory until no-U-turn criterion stopping
  - max treedepth to keep computation in control
  - pick a draw along the trajectory with probabilities adjusted to take into account the error in the discreteized dynamic simulation
- Mass matrix and step size adaptation in Stan
  - mass matrix refers to having different scaling for different parameters and optionally also rotation to reduce correlations
  - mass matrix and step size adjustment and are estimated during initial adaptation phase
  - step size is adjusted to be as big as possible while keeping discretization error in control
- After adaptation the algorithm parameters are fixed and some further iterations included in the warmup
- After warmup store iterations for inference

# Adaptive dynamic HMC in Stan

- Dynamic HMC using growing tree to increase simulation trajectory until no-U-turn criterion stopping
  - max treedepth to keep computation in control
  - pick a draw along the trajectory with probabilities adjusted to take into account the error in the discreteized dynamic simulation
- Mass matrix and step size adaptation in Stan
  - mass matrix refers to having different scaling for different parameters and optionally also rotation to reduce correlations
  - mass matrix and step size adjustment and are estimated during initial adaptation phase
  - step size is adjusted to be as big as possible while keeping discretization error in control
- After adaptation the algorithm parameters are fixed and some further iterations included in the warmup
- After warmup store iterations for inference
- See more details in Stan reference manual

## Max tree depth diagnostic

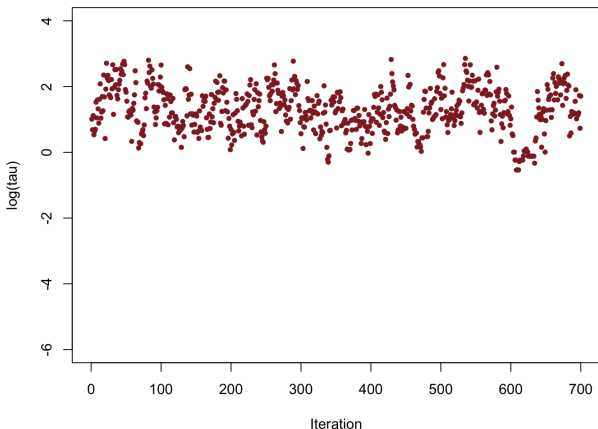
- Dynamic HMC specific diagnostic
- Indicates inefficiency in sampling leading to higher autocorrelations and lower  $n_{\text{eff}}$
- Different parameterizations matter

# Divergences

- HMC specific: Indicates that Hamiltonian dynamic simulation has problems going to narrow places
  - indicates possibility of biased estimates
- Different parameterizations matter
- [http://mc-stan.org/users/documentation/case-studies/divergences\\_and\\_bias.html](http://mc-stan.org/users/documentation/case-studies/divergences_and_bias.html)

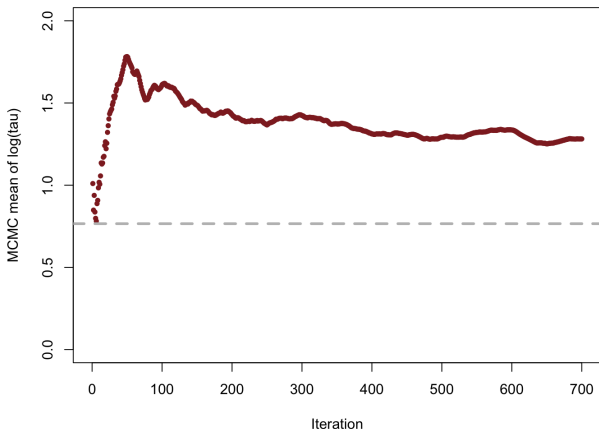
# Divergences

- HMC specific: Indicates that Hamiltonian dynamic simulation has problems going to narrow places
  - indicates possibility of biased estimates
- Different parameterizations matter
- [http://mc-stan.org/users/documentation/case-studies/divergences\\_and\\_bias.html](http://mc-stan.org/users/documentation/case-studies/divergences_and_bias.html)



# Divergences

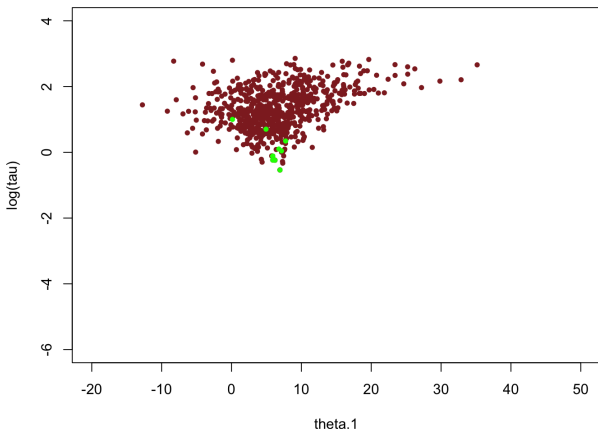
- HMC specific: Indicates that Hamiltonian dynamic simulation has problems going to narrow places
  - indicates possibility of biased estimates
- Different parameterizations matter
- [http://mc-stan.org/users/documentation/case-studies/divergences\\_and\\_bias.html](http://mc-stan.org/users/documentation/case-studies/divergences_and_bias.html)





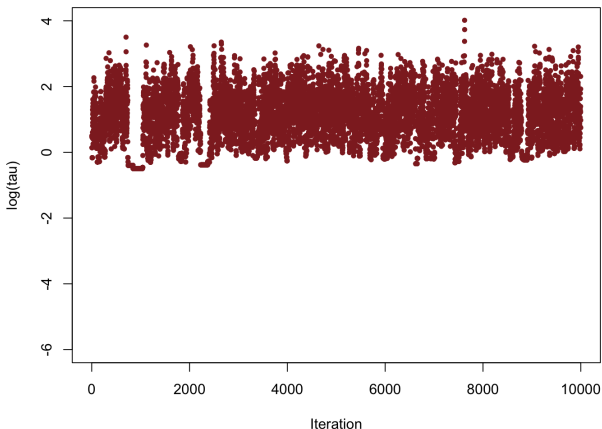
# Divergences

- HMC specific: Indicates that Hamiltonian dynamic simulation has problems going to narrow places
  - indicates possibility of biased estimates
- Different parameterizations matter
- [http://mc-stan.org/users/documentation/case-studies/divergences\\_and\\_bias.html](http://mc-stan.org/users/documentation/case-studies/divergences_and_bias.html)



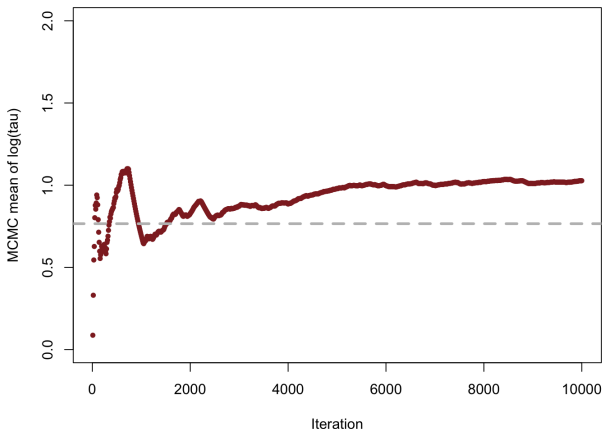
# Divergences

- HMC specific: Indicates that Hamiltonian dynamic simulation has problems going to narrow places
  - indicates possibility of biased estimates
- Different parameterizations matter
- [http://mc-stan.org/users/documentation/case-studies/divergences\\_and\\_bias.html](http://mc-stan.org/users/documentation/case-studies/divergences_and_bias.html)



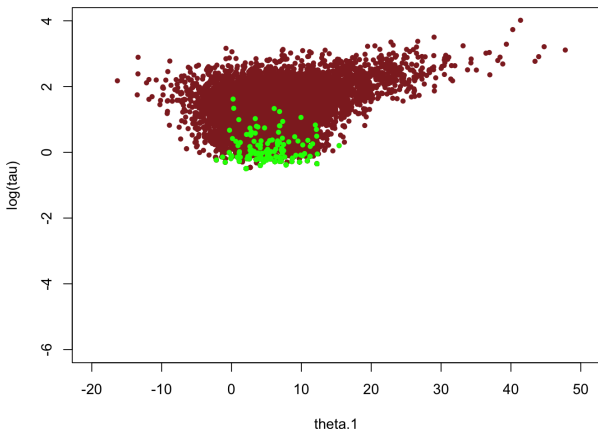
# Divergences

- HMC specific: Indicates that Hamiltonian dynamic simulation has problems going to narrow places
  - indicates possibility of biased estimates
- Different parameterizations matter
- [http://mc-stan.org/users/documentation/case-studies/divergences\\_and\\_bias.html](http://mc-stan.org/users/documentation/case-studies/divergences_and_bias.html)



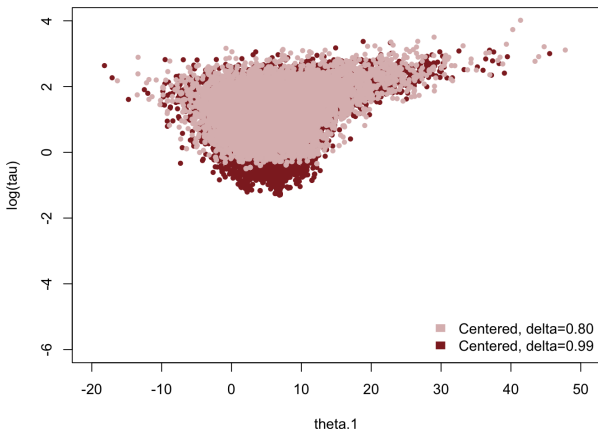
# Divergences

- HMC specific: Indicates that Hamiltonian dynamic simulation has problems going to narrow places
  - indicates possibility of biased estimates
- Different parameterizations matter
- [http://mc-stan.org/users/documentation/case-studies/divergences\\_and\\_bias.html](http://mc-stan.org/users/documentation/case-studies/divergences_and_bias.html)



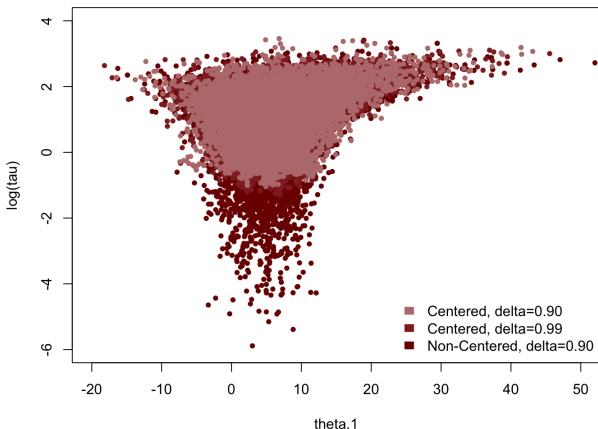
# Divergences

- HMC specific: Indicates that Hamiltonian dynamic simulation has problems going to narrow places
  - indicates possibility of biased estimates
- Different parameterizations matter
- [http://mc-stan.org/users/documentation/case-studies/divergences\\_and\\_bias.html](http://mc-stan.org/users/documentation/case-studies/divergences_and_bias.html)



# Divergences

- HMC specific: Indicates that Hamiltonian dynamic simulation has problems going to narrow places
  - indicates possibility of biased estimates
- Different parameterizations matter
- [http://mc-stan.org/users/documentation/case-studies/divergences\\_and\\_bias.html](http://mc-stan.org/users/documentation/case-studies/divergences_and_bias.html)



# 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, PyMC3, Pyro (Uber), Edward (Google), Birch, ELFI, ...

# Stan - probabilistic programming framework

- Language, inference engine, user interfaces, documentation, case studies, diagnostics, packages, ...



mc-stan.org

# Stan - probabilistic programming framework

- Language, inference engine, user interfaces, documentation, case studies, diagnostics, packages, ...
- More than ten thousand users in social, biological, and physical sciences, medicine, engineering, and business
- Several full time developers, 35 in dev team, more than 100 contributors



mc-stan.org

# Stan - probabilistic programming framework

- Language, inference engine, user interfaces, documentation, case studies, diagnostics, packages, ...
- More than ten thousand users in social, biological, and physical sciences, medicine, engineering, and business
- Several full time developers, 35 in dev team, 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 binomial  
}  
  
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 binomial  
}  
  
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 binomial  
}  
  
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

```
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  $\text{<lower=a>}$
  - e.g. logit transformation for  $\text{<lower=a,upper=b>}$
- 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);  // observation model  
}
```

## Binomial model - Stan code

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

- $\sim$  is syntactic sugar and this could be also written as

```
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);  // observation model  
}
```

- $\sim$  is syntactic sugar and this could be also written as

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

- Stan model code constructs log density!

## Binomial model - Stan code

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

- $\sim$  is syntactic sugar and this could be also written as

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

- Stan model code constructs log density!
- lpdf for continuous, lpmf for discrete distributions

# Stan

- Stan compiles (transpiles) 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 and PyStan

## RStan

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

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

---

## PyStan

```
import pystan
import stan_utility

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



# RStan and PyStan

## RStan

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

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

---

## PyStan

```
import pystan
import stan_utility

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

# RStan and PyStan

## RStan

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

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

---

## PyStan

```
import pystan
import stan_utility

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

# RStan and PyStan

## RStan

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

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

---

## 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

- Stan compiles 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

# Stan

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

## 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):

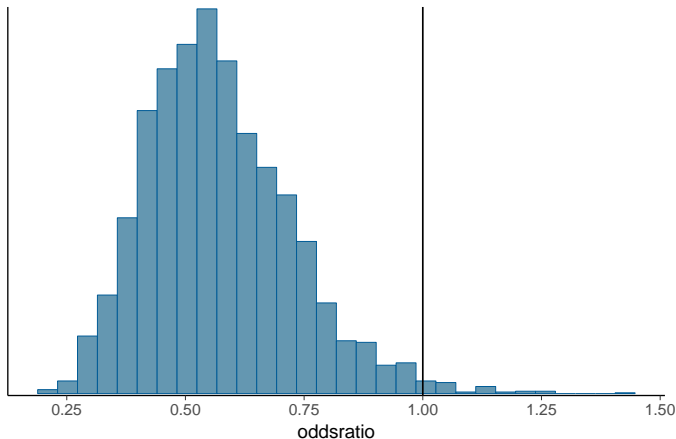
	mean	se_mean	sd	10%	50%	90%	n_eff	Rhat
theta1	0.1	0	0.0	0.0	0.1	0.1	3280	1
theta2	0.0	0	0.0	0.0	0.0	0.0	3171	1
oddsratio	0.6	0	0.2	0.4	0.6	0.8	3108	1
lp__	-253.5	0	1.0	-254.8	-253.2	-252.6	1922	1

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)
```

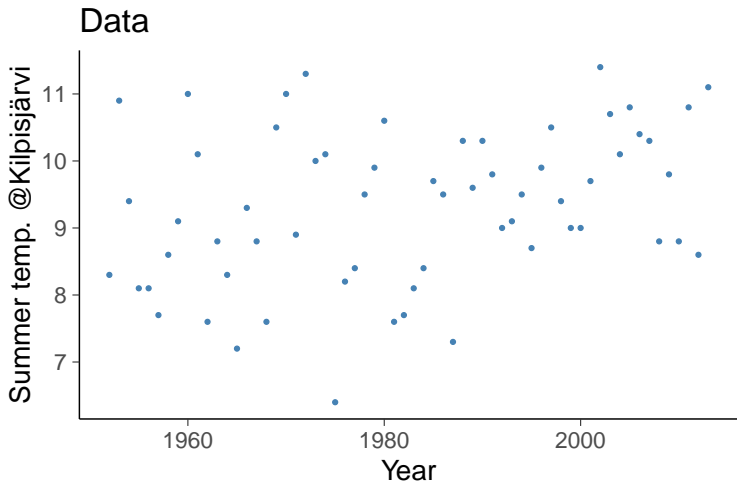
```
[1] "0 of 4000 iterations saturated the maximum tree depth of 10 (0%)"  
[1] "0 of 4000 iterations ended with a divergence (0%)"
```

# 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);  
}
```

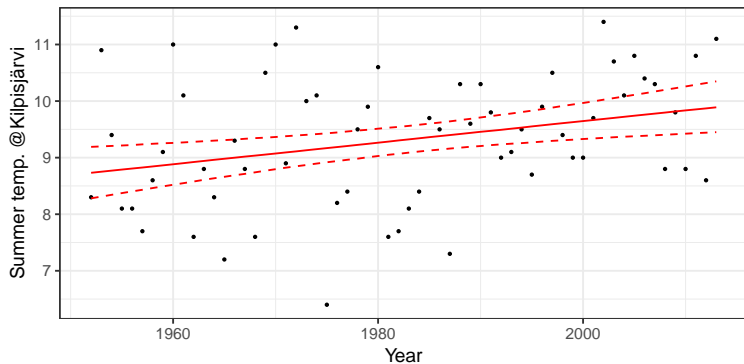
# Student-t linear model

```
...
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);
}
```

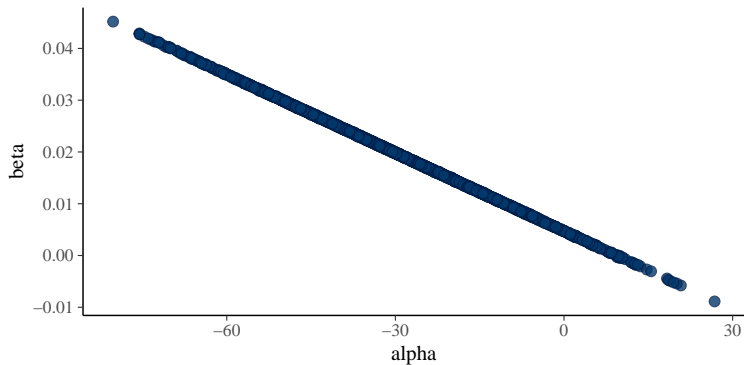
# Priors

- Prior for temperature increase?

# Kilpisjärvi summer temperature

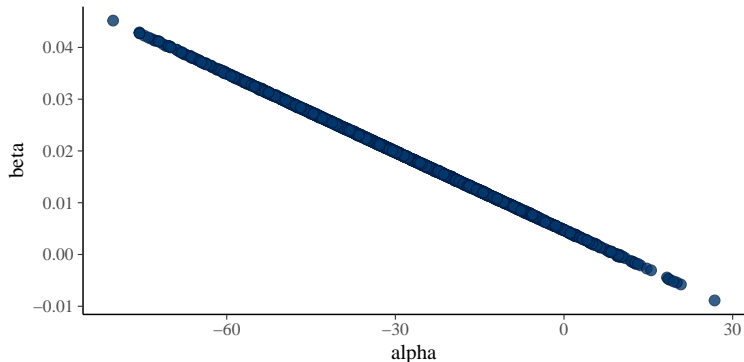


# Kilpisjärvi summer temperature





# Kilpisjärvi summer temperature



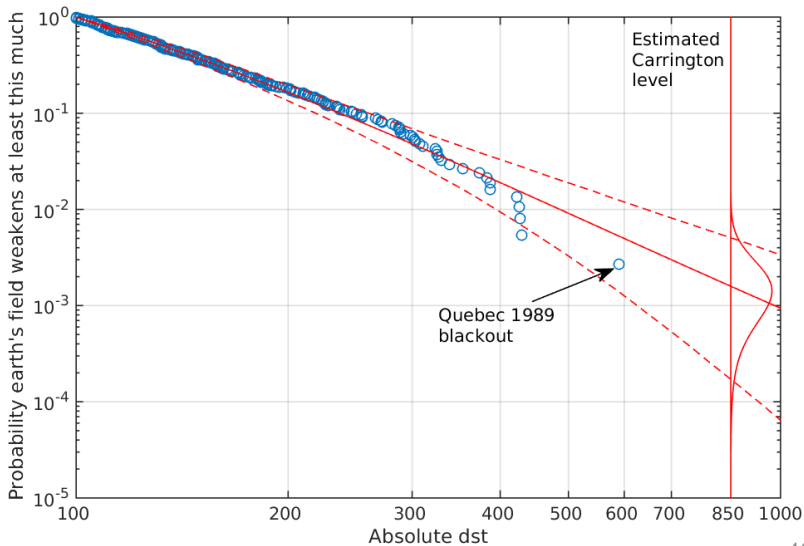
There were 14 transitions after warmup that exceeded the maximum treedepth. Increase `max_treedepth` above 10. See <http://mc-stan.org/misc/warnings.html#maximum-treedepth-exceeded> Examine the `pairs()` plot to diagnose sampling problems

# 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);  
}
```

# 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_log(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  
  }  
}
```

# RStanARM

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

```
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)
```

```
draws_bin2 <- as.data.frame(fit_bin2) %>%
  mutate(theta1 = plogis('(Intercept)'),
         theta2 = plogis('(Intercept)' + grp2),
         oddsratio = (theta2/(1-theta2))/(theta1/(1-theta1)))
```

```
mcmc_hist(draws_bin2, pars='oddsratio')
```

# 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())
```

## 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)