Stan in Masterclass in Bayesian Statistics

- Stan and probabilistic programming
  - RStan
  - rstanarm and brms
- Dynamic HMC used in Stan
- MCMC convergence diagnostics used in Stan
- Posterior predictive model checking

- avehtari.github.io/masterclass

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 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=ZRpo41l02KQ&index=6&list=PLuwyh42iHquU4hUBQs20hkBsKSMrp6H0J
  - https://www.youtube.com/watch?v=6cc4N1vT8pk&index=7&list=PLuwyh42iHquU4hUBQs20hkBsKSMrp6H0J

Demos

- https://github.com/avehtari/BDA_R_demos/
  - git clone or download zip
  - demos_ch12/demo12_1.R - HMC
  - demos_rstan/rstan_demo.Rmd
  - demos_rstan/rstanarm_demo.Rmd
  - demos_rstan/ppc/poisson-ppc.Rmd

Demos

- https://github.com/avehtari/BDA_R_demos/
  - git clone or download zip
  - demos_ch12/demo12_1.R - HMC
  - demos_rstan/rstan_demo.Rmd
  - demos_rstan/rstanarm_demo.Rmd
  - demos_rstan/ppc/poisson-ppc.Rmd
- http://elevanth.org/blog/2017/11/28/build-a-better-markov-chain/
- http://sumsar.net/blog/2017/01/bayesian-computation-with-stan-and-farmer-jons/
- https://betanalpha.github.io/assets/case_studies/principled_bayesian_workflow.html
- http://mc-stan.org/documentation/case-studies.html
- https://cran.r-project.org/package=rstan
- https://cran.r-project.org/package=rstanarm
- https://cran.r-project.org/package=brms

Probabilistic programming

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

Probabilistic programming

- 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

- 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)
- Many frameworks Stan, PyMC3, Pyro (Uber), Edward (Google), Birch, 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 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, ...
  - autodiff to compute gradients of the log density
- 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

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

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

- Parameters may have constraints
- Stan makes transformation to unconstrained space and samples in unconstrained space
  - log transformation for `<lower=a>` and `<upper=b>`
  - 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
  - log transformation for `<lower=a>` and `<upper=b>`
  - 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 could be also written as

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

Binomial model - Stan code

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

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

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

- `target` is the log posterior density (usually)

Binomial model - Stan code

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

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

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

- `target` is the log posterior density (usually)
- `_lpdf` for continuous, `_lpmf` for discrete distributions

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())
source('stan_utility.R')

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())
source('stan_utility.R')

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

Stan

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

Stan hands on

- https://github.com/avehtari/BDA_R_demos/blob/master/demos_rstan/
  - git clone or download zip from https://github.com/avehtari/BDA_R_demos/
- Run and experiment with Bernoulli and Binomial model
  - Examine sample structure and do some plotting
- Probability of a girl birth given placenta previa:
  437 girls and 543 boys have been observed
  - is the ratio different from the population average 0.485?

Dynamic HMC in Stan

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

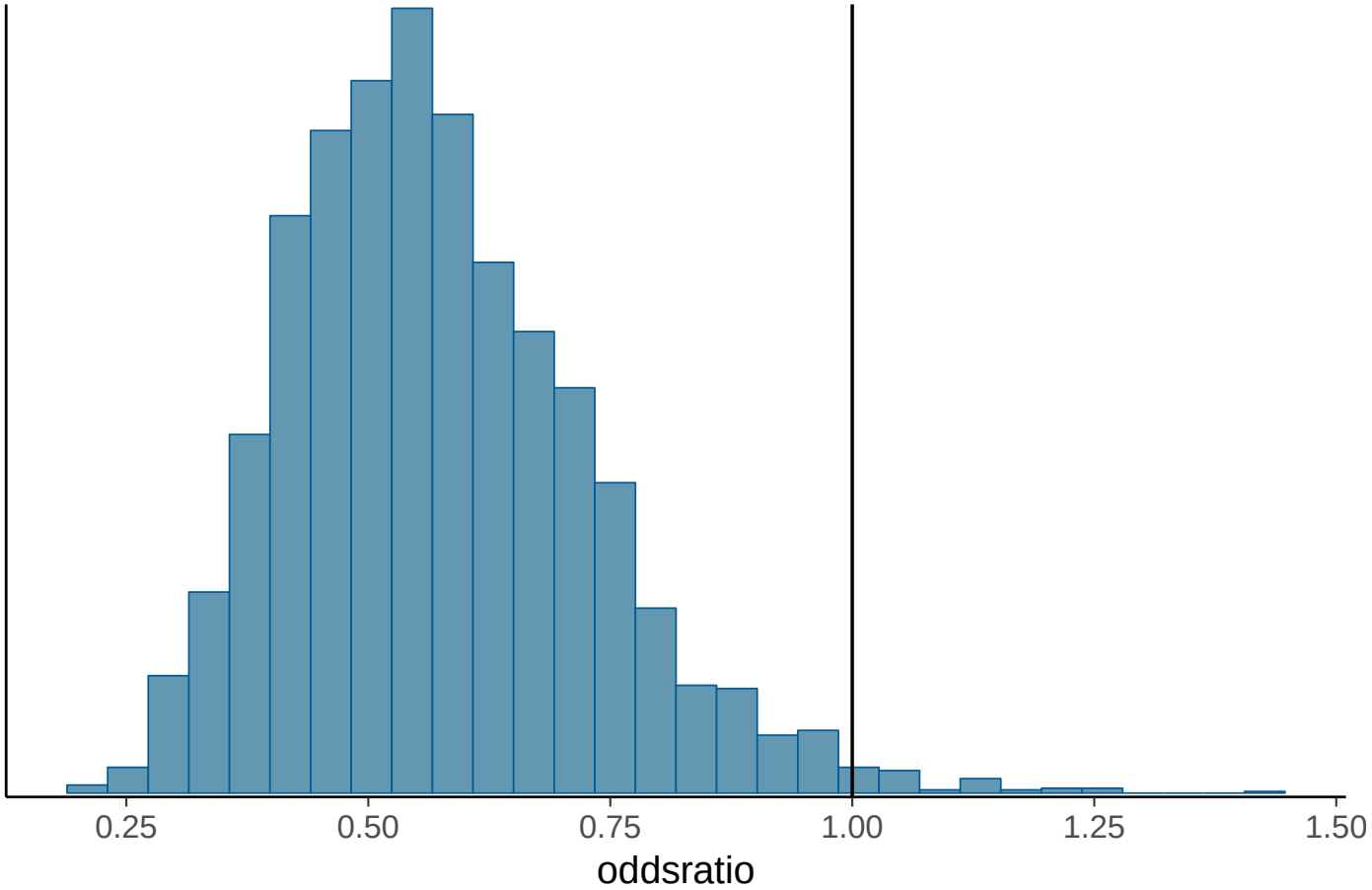
Difference between proportions

```
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

```
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)))
```



Hands on: Farmer Jöns (by Rasmus Bååth)

- Cows and disease:
  Farmer Jöns has a huge number of cows. Earlier this year he ran an experiment where he gave 10 cows medicine A and 10 medicine B and then measured whether they got sick or not during the summer season. The number of cows that got sick in groups A and B were 2 and 6 respectively.
  - How effective are the drugs?
  - What is the evidence that medicine A is better or worse than medicine B?

Inference diagnostics

```
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).
```

Inference diagnostics

```
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”

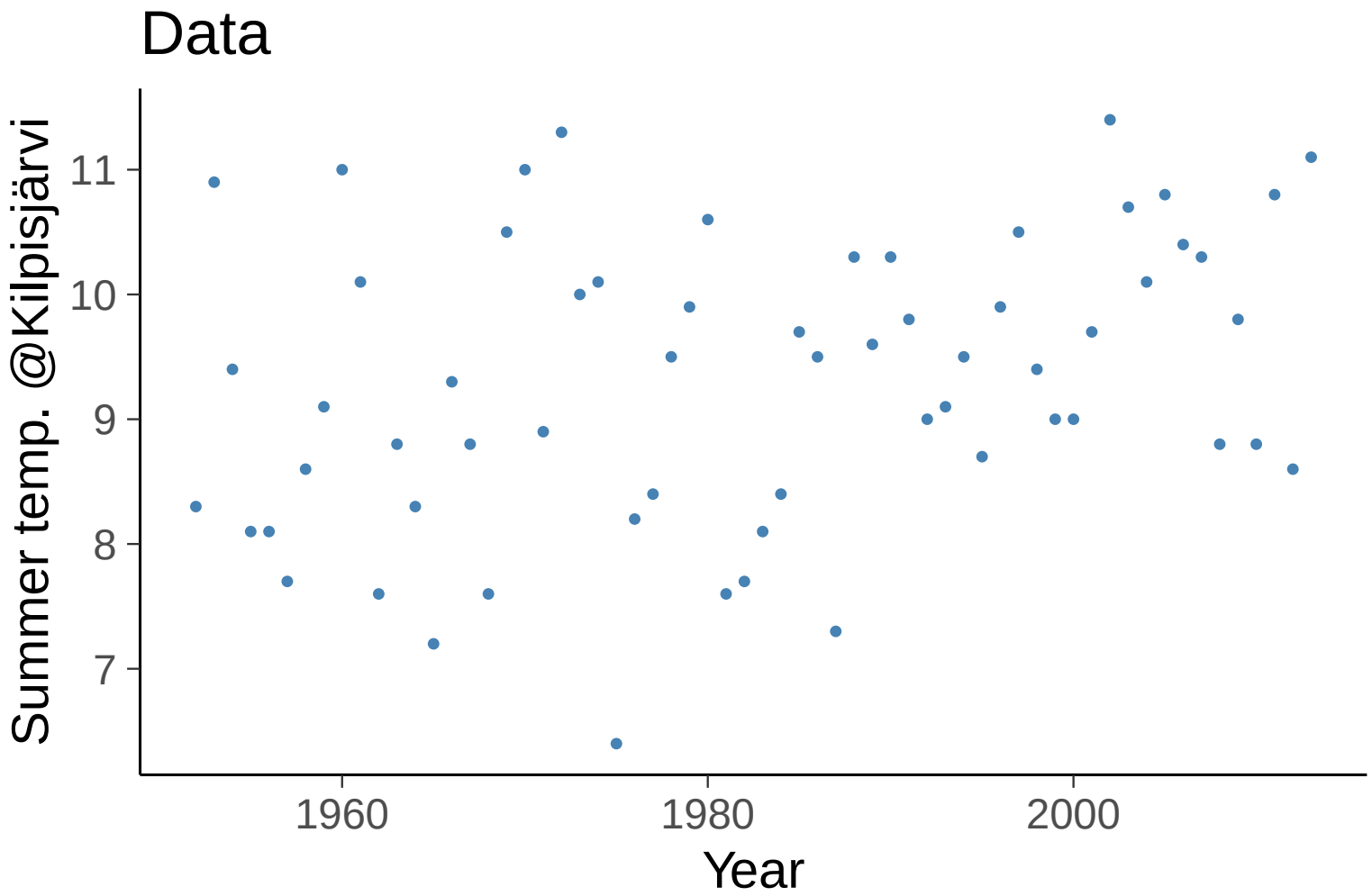
Rhat and n_eff

Hands on

- Check Rhat and n_eff for the previous model fits

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);
}
```

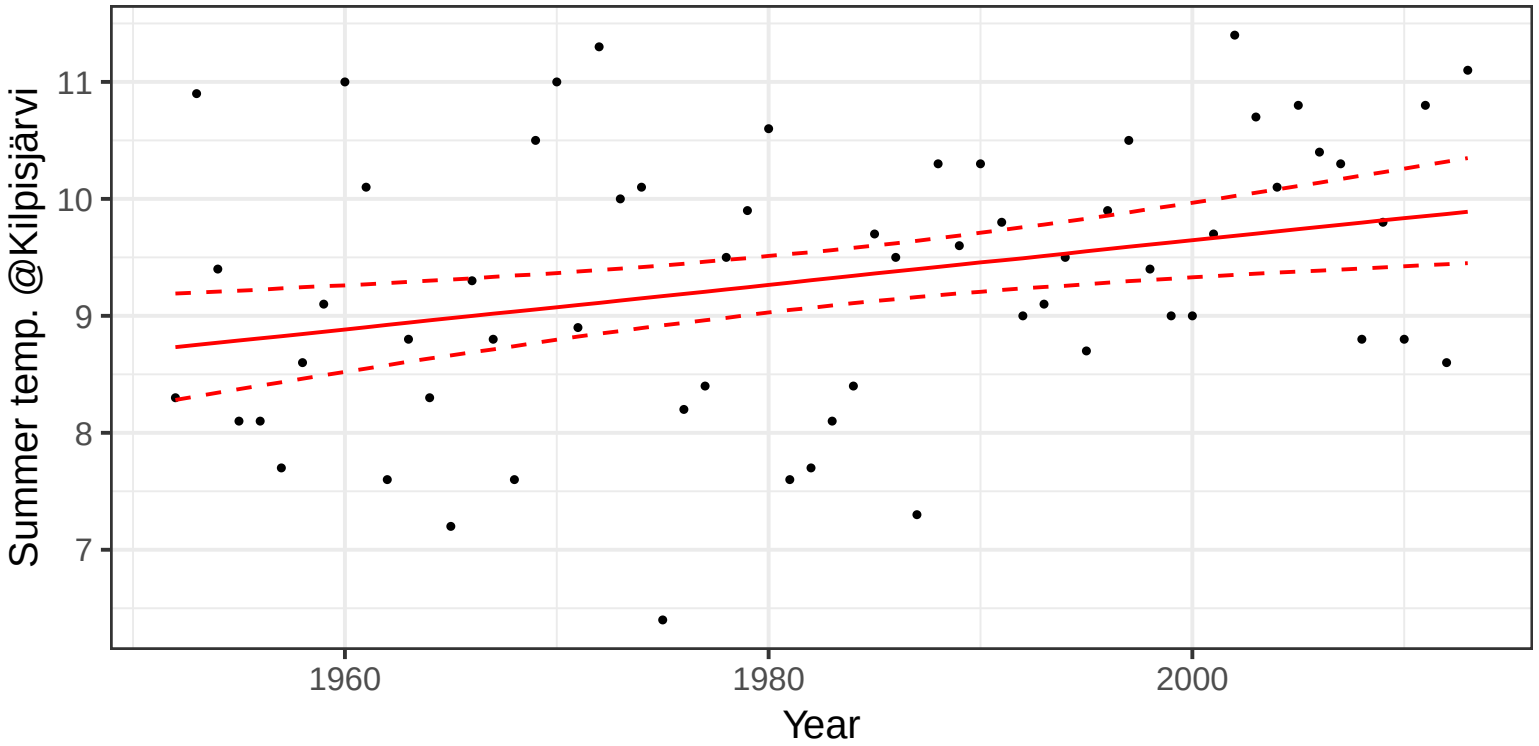
Student-t linear model

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

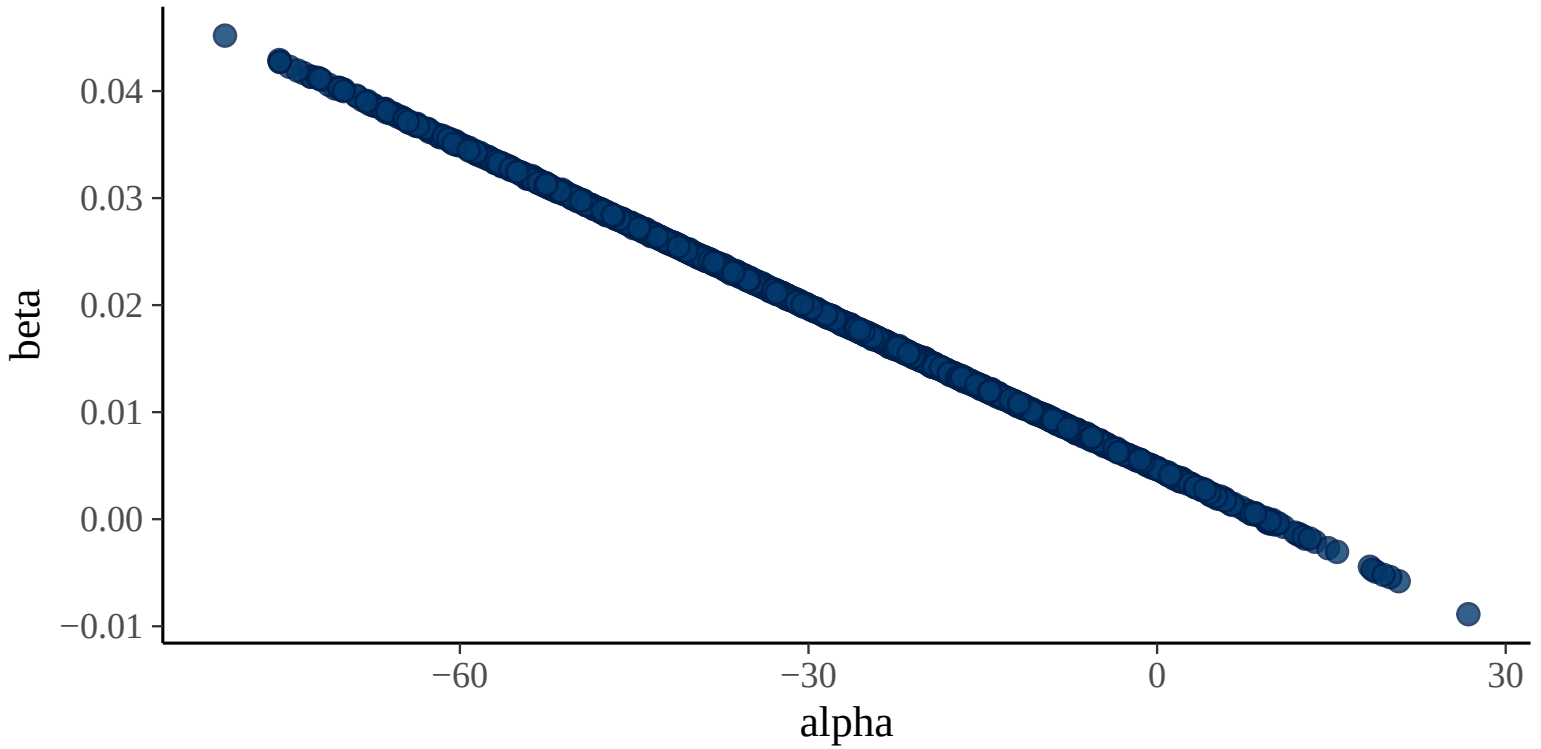
Priors

- Prior for temperature increase?

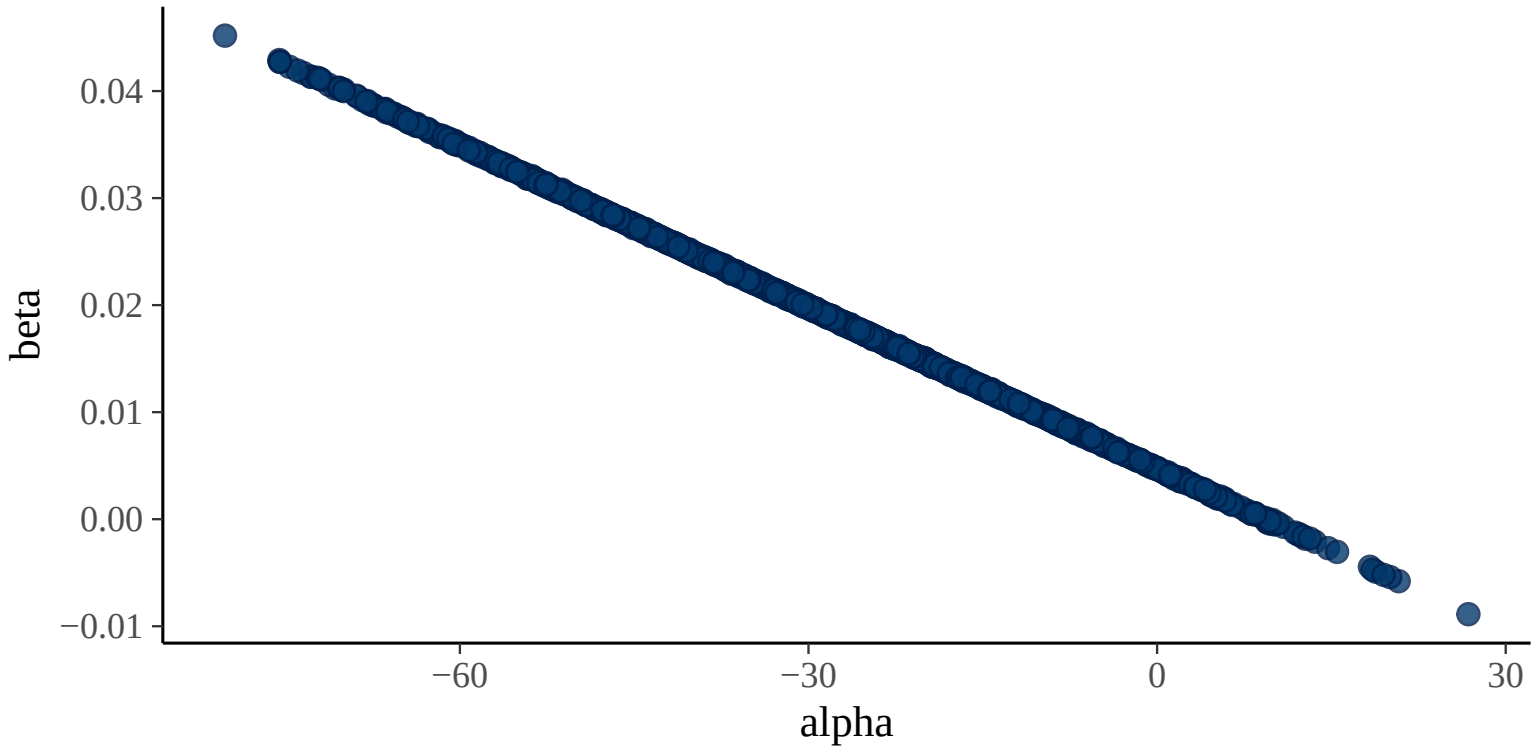
Kilpisjärvi summer temperature



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

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

Linear regression model with standardized data

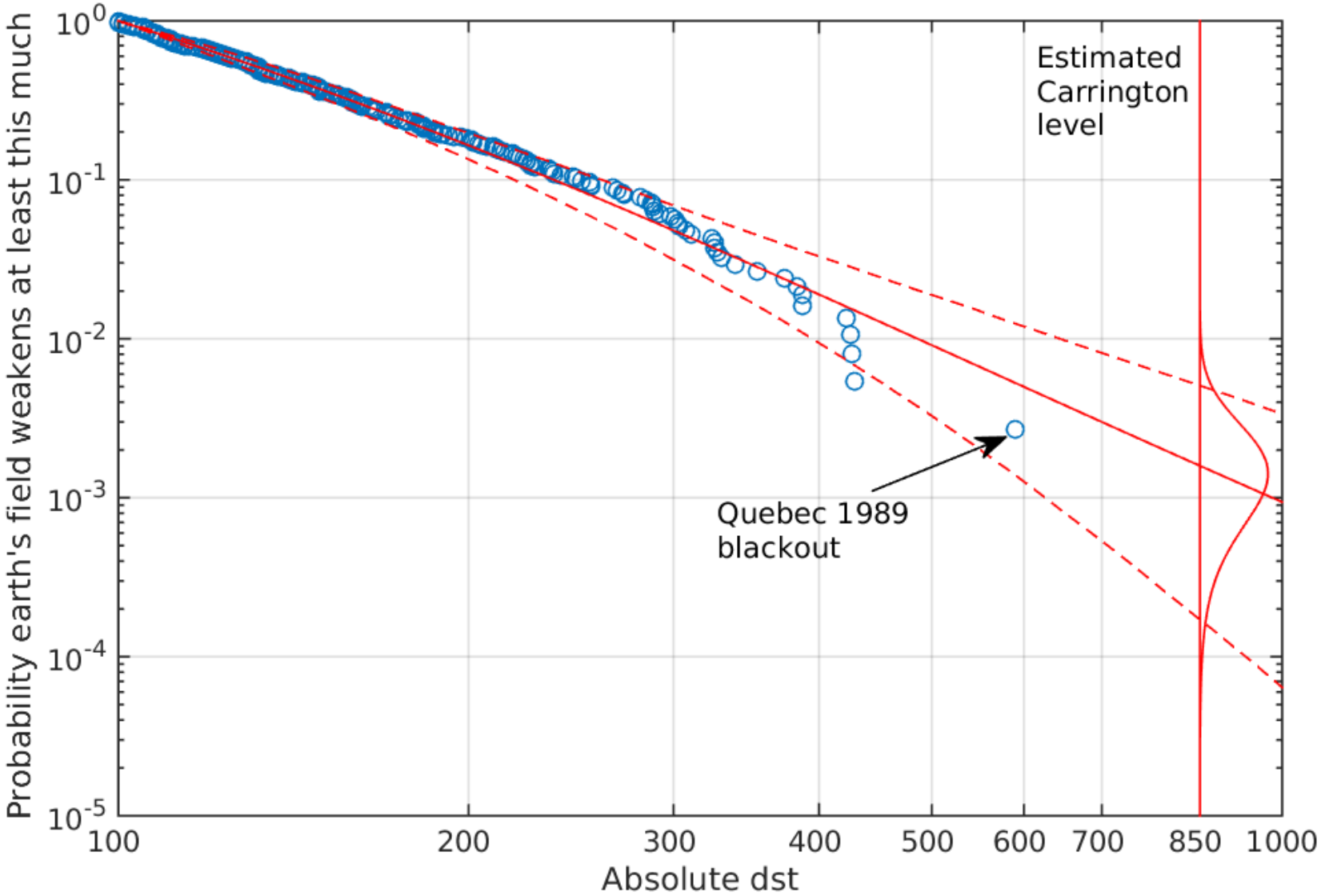
```
data {
  int<lower=0> N; // number of data points
  vector[N] x; //
  vector[N] y; //
  real xpred; // input location for prediction
}
transformed data {
  vector[N] x_std;
  vector[N] y_std;
  real xpred_std;
  x_std = (x - mean(x)) / sd(x);
  y_std = (y - mean(y)) / sd(y);
  xpred_std = (xpred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

Hands on: ShinyStan

- Use linear regression model with standardized data for Kilpisjärvi data and use ShinyStan to check the results

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
    int N;
    N <- dims(y)[1];
    if (fabs(k) > 1e-15)
      return -(1+1/k)*sum(log1pv(y*k/sigma)) -N*log(sigm
    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
    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
  - can be used to generate Stan code which can then be further modified

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

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

BRMS

```
data {
  int<lower=1> N;  // total number of observations
  vector[N] Y;  // response variable
  int<lower=1> K;  // number of population-level effects
  matrix[N, K] X;  // population-level design matrix
  int prior_only;  // should the likelihood be ignored?
}
transformed data {
  int Kc = K - 1;
  matrix[N, K - 1] Xc;  // centered version of X
  vector[K - 1] means_X;  // column means of X before centering
  for (i in 2:K) {
    means_X[i - 1] = mean(X[, i]);
    Xc[, i - 1] = X[, i] - means_X[i - 1];
  }
}
parameters {
  vector[Kc] b;  // population-level effects
  real temp_Intercept;  // temporary intercept
  real<lower=0> sigma;  // residual SD
  real<lower=1> nu;  // degrees of freedom or shape
}
transformed parameters {
}
```

BRMS

```
model {
  vector[N] mu = temp_Intercept + Xc * b;
  // priors including all constants
  target += student_t_lpdf(temp_Intercept | 3, 9, 10);
  target += student_t_lpdf(sigma | 3, 0, 10)
    - 1 * student_t_lccdf(0 | 3, 0, 10);
  target += gamma_lpdf(nu | 2, 0.1)
    - 1 * gamma_lccdf(1 | 2, 0.1);
  // likelihood including all constants
  if (!prior_only) {
    target += student_t_lpdf(Y | nu, mu, sigma);
  }
}
generated quantities {
  // actual population-level intercept
  real b_Intercept = temp_Intercept - dot_product(means_X, b);
}
```

Prior and posterior predictive checking

Hands on: Farmer Jöns (by Rasmus Bååth)

- Cows and milk:
  Farmer Jöns has a huge number of cows. Earlier this year he ran an experiment where he gave 10 cows a special diet that he had heard could make them produce more milk. He recorded the number of liters of milk from these "diet" cows and from 15 "normal" cows during one month. This is the data:
  diet_milk <- c(651, 679, 374, 601, 401, 609, 767, 709, 704, 679)
  normal_milk <- c(798, 1139, 529, 609, 553, 743, 151, 544, 488, 555, 257, 692, 678, 675, 538)
  - Was the diet any good, does it results in better milk production?
- You may check additional hints at
  http://sumsar.net/files/posts/2017-01-15-bayesian-computation-with-stan-and-farmer-jons/stan_exercise.html

Hands on: Farmer Jöns (by Rasmus Bååth)

- Cows and sunshine:
  Farmer Jöns has a huge number of cows. He is wondering whether the amount of time a cow spends outside in the sunshine affects how much milk she produces. To test this he makes a controlled experiment where he picks out 20 cows and assigns each a number of hours she should spend outside each day. The experiment runs for a month and Jöns records the number of liters of milk each cow produces:
  d <- data.frame(milk = c(685, 691, 476, 1151, 879, 725, 1190, 1107, 809, 539, 298, 805, 820, 498, 1026, 1217, 1177, 684, 1061, 834),
  hours = c(3, 7, 6, 10, 6, 5, 10, 11, 9, 3, 6, 6, 3, 5, 8, 11, 12, 9, 5, 5))
  - Does sunshine affect milk production positively or negatively?

Hands on

- More farmers exercises
  - Cows and Mutant Cows: “outliers” and Student’s t
  - Chickens and diet: Poisson model
  - Cows and more diets: more groups in comparison