UiO : Matematisk institutt

Det matematisk-naturvitenskapelige fakultet

STK-4051/9051 Computational Statistics Spring 2021
Software for Bayesian Inference

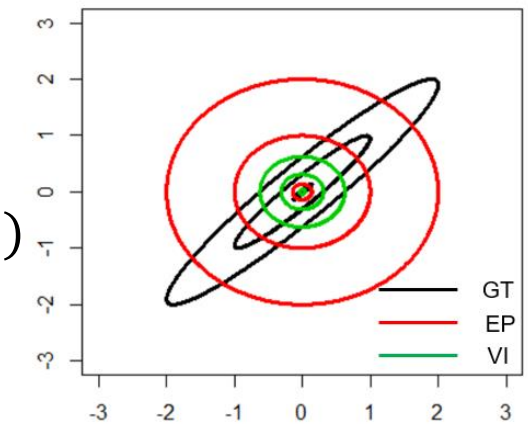
Instructor: Odd Kolbjørnsen, oddkol@math.uio.no

Variational inference

$$E[h(\boldsymbol{z})|\boldsymbol{x}] \approx \int_{\boldsymbol{z}} h(\boldsymbol{z})q^*(\boldsymbol{z})d\boldsymbol{z} \quad q^*(\boldsymbol{z}) = \arg\min_{q(\boldsymbol{z})\in\mathcal{Q}} \mathcal{D}(q(\boldsymbol{z}), p(\boldsymbol{z}|\boldsymbol{x}))$$

$$\mathrm{KL}(q(\boldsymbol{z})||\, p(\boldsymbol{z}|\boldsymbol{x})) = \int_{z} \log\left(\frac{q(\boldsymbol{z})}{p(\boldsymbol{z}|\boldsymbol{x})}\right) q(\boldsymbol{z})d\boldsymbol{z}$$

$$= E^q(\log q(\boldsymbol{Z})) - E^q(\log p(\boldsymbol{Z}|\boldsymbol{x}))$$

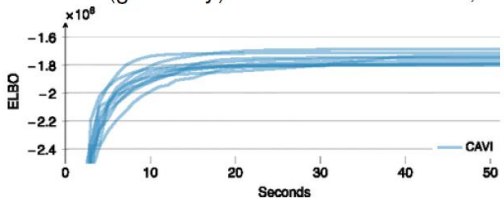


Minimizing KL is equivalent maximizing

$$\mathrm{ELBO}(q) = E^q[\log p(\boldsymbol{z}, \boldsymbol{x})] - E^q[\log q(\boldsymbol{z})]$$

CAVI: Coordinate ascent variational inference

- ELBO is (generally) a nonconvex function, local optimum



Software

- There is always an existing software that ”does your job”
- Even if the software does not do exactly what you want maybe it is good enough (cost - benefit)
- The challenge is to figure out how this works
  (and how you can make it do what you want)
- Reasons **not** to make your own computer code:
  - existing code is tested (less bugs)
  - existing code is optimized (speed)
  - often related to publications easier to get others to ”accept it”
- Reasons to make your own computer code:
  - understand the methodology better
  - improve/develop existing methodology
  - combining with other techniques
  - compete with existing computer code
  - because you have too much spare time

Software for Bayesian inference

- MCMC (https://en.wikipedia.org/wiki/Probabilistic_programming)
  - WinBUGS
  - OpenBUGS
  - JAGS (Just Another Gibbs Sampler)
  - TensorFlow Probability
  - R-STAN (Hamiltonian MC)
- Approximation
  - R-STAN (Variational Inference)
  - TensorFlow Probability (Variational Inference)
  - R-INLA (Integrated Nested Laplace Approximations)

BUGS = Bayesian inference Using Gibbs Sampler

What is Stan?

"A probabilistic programming language implementing full ***Bayesian statistical inference with MCMC sampling*** *(NUTS, HMC) and penalized maximum likelihood estimation with Optimization (L-BFGS)"*

"Stanislaw Ulam, namesake of Stan and co-inventor Monte Carlo methods shown here holding the Fermiac, Enrico Fermi's physical Monte Carlo simulator for neutron diffusion." (image from the Stan manual)

What does stan do?

- Samples from the posterior distribution (if your model is specified correctly)
- "Fits" bayesian models
- Empowers you to write your own Bayesian models, it's much easier than you think!

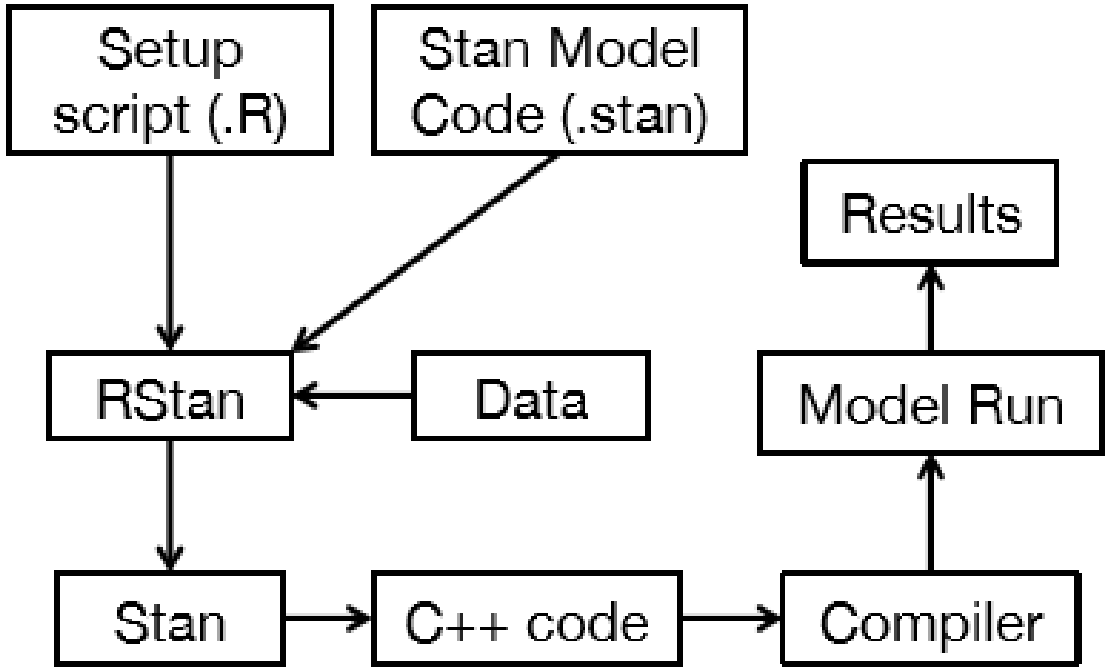
No U-Turn Sampler

Automatic Step Size and Number Adaptation

"Adaptive Hamiltonian MC"

How things are built together

Stan is a library with a number of interfaces, we will use the R interface called RStan.




```
library("rstan")
```

What does a stan call do in R?

- The stan function does all of the work of fitting a Stan model and returning the results as an instance of stanfit.

- The steps are roughly as follows:
  - Translate the Stan model to C++ code. (stanc)
  - Compile the C++ code into a binary shared object, which is loaded into the current R session (an object of S4 class stanmodel is created). (stan_model)
  - Draw samples and wrap them in an object of S4 class stanfit. (sampling)

- The returned object can be used with methods such as print, summary, and plot to inspect and retrieve the results of the fitted model.

- stan can also be used to sample again from a fitted model under different settings (e.g., different iter, data, etc.) by using the fit argument to specify an existing stanfit object. In this case, the compiled C++ code for the model is reused.

- Stan keeps track of what he has already compiled, if you change file you get:

```
hash mismatch so recompiling; make sure Stan code ends with a blank line
```

(not an error)

What does a stan call do in R?

```
fit = stan(file ='8schools.stan', data = schools_dat)
```

- The stan function does all of the work of fitting a Stan model and returning the results as an instance of stanfit.

- The steps are roughly as follows:
  - Translate the Stan model to C++ code. (stanc)
  - Compile the C++ code into a binary shared object, which is loaded into the current R session (an object of S4 class stanmodel is created). (stan_model)
  - Draw samples and wrap them in an object of S4 class stanfit. (sampling)

- The returned object can be used with methods such as print, summary, and plot to inspect and retrieve the results of the fitted model.

- stan can also be used to sample again from a fitted model under different settings (e.g., different iter, data, etc.) by using the fit argument to specify an existing stanfit object. In this case, the compiled C++ code for the model is reused.

- Stan keeps track of what he has already compiled, if you change file you get:

```
hash mismatch so recompiling; make sure Stan code ends with a blank line
```

(not an error)

Meta analysis of treatment effect in 8 schools

- Effect of coaching program for SAT-V (Scholastic Aptitude Test - Verbal)
- Data analyzed at each school separately to derive:
  - Estimated treatment effect (Treatment=Special preparation)
  - Standard error of the treatment effect
- Analysis of data at each school adjust for PSAT (initial level of students)
- By pooling data across experiments we can improve all estimates
- Is the pooling justified or is there an effect of “better” teaching at schools that does the best job
  - If so we should look to the best school to investigate what they did differently (~right)

Data from Rubin (1981)

Journal of Educational Statistics
Winter, 1981, Volume 6, Number 4, pp. 377-400

TABLE 1

Effects of Special Preparation on SAT-V Scores in Eight Randomized Experiments

School	Number of Students: Treatment	Number of Students: Control	Estimated Treatment Effect	Standard Error of Effect Estimate	Residual Variance
A	28	22	28.39	14.9	2415
B	39	40	7.94	10.2	1880
C	22	17	-2.75	16.3	2168
D	48	43	6.82	11.0	2612
E	25	74	-.64	9.4	1623[a]
F	37	35	.63	11.4	2046[a]
G	24	70	18.01	10.4	1841
H	16	19	12.16	17.6	2314

[a]Regression includes a quadratic term for PSAT-V

Code on webpage: `8schools.r`,
`8schools.stan`, `8schoolsDirect.stan`

Running Stan in RStudio

```r
#install.packages("rstan", repos = "https://cloud.r-project.org/", dependencies = TRUE)
library("rstan")
rstan_options(auto_write = TRUE)


schools_dat = list(J = 8,
                   y = c(28,  8, -3,  7, -1,  1, 18, 12),
                   sigma = c(15, 10, 16, 11,  9, 11, 10, 18))

fit = stan(file = 'C:/Users/files/Documents/R/Stan/8schools.stan', data = schools_dat)
```

rstan install

add to current session

Data (Rubin 1981)

Run

```
//
// Learn more about model development with Stan at:
//
//    http://mc-stan.org/users/interfaces/rstan.html
//    https://github.com/stan-dev/rstan/wiki/RStan-Getting-Started
//
// saved as 8schools.stan
data {
  int<lower=0> J;         // number of schools
  real y[J];              // estimated treatment effects
  real<lower=0> sigma[J]; // standard error of effect estimates
}
parameters {
  real mu;                // population treatment effect
  real<lower=0> tau;      // standard deviation in treatment effects
  vector[J] eta;          // unscaled deviation from mu by school
}
transformed parameters {
  vector[J] theta = mu + tau * eta;        // school treatment effects
}
model {
  target += normal_lpdf(eta | 0, 1);       // prior log-density
  target += normal_lpdf(y | theta, sigma); // log-likelihood
}

```

Stan program for the model

- RStudio version 1.2.x or later
  Has good support for R-STAN
- File -> New File -> Stan File

Be sure that your Stan programs ends in a blank line without any characters including spaces and comments.

Content of .stan file

- Data
  - Real numbers with constraints
  - $\boldsymbol{y},\ \boldsymbol{\sigma}$
- Transformed data: (not a good name)
  - Real numbers and equations executed once
  - Typically fixed hyper parameters
  - alpha $= 1, \text{beta} = 1$
  - Any variable that is defined wholly in terms of data or transformed data should be declared and defined in the transformed data block.
- Parameter
  - The random variables we will sample
  - $\boldsymbol{\eta} = (\eta_1, \dots, \eta_p),\ \mu, \tau$
- Transformed parameters
  - $x_i = \tau \cdot \eta_i + \mu$

- Model
  - Prior: $p(\boldsymbol{\eta}, \mu, \tau)$
  - Likelihood: $p(\boldsymbol{y}|\boldsymbol{x}, \mu, \tau)$
- Generated quantities
  - $h(\boldsymbol{x}, \mu, \tau)$

$$E[h(\boldsymbol{x}, \mu, \sigma)|y] = \int_{\boldsymbol{z}} h(\boldsymbol{x}, \mu, \tau) p(\boldsymbol{x}, \mu, \tau|y) d\boldsymbol{x} d\mu d\sigma$$

Prior of mu and tau are not defined in model
=> Improper prior: $f(\mu, \tau) \propto 1$

```
//
// Learn more about model development with Stan at:
//
//    http://mc-stan.org/users/interfaces/rstan.html
//    https://github.com/stan-dev/rstan/wiki/RStan-Getting-Started
//
// saved as 8schools.stan
data {
  int<lower=0> J;          // number of schools
  real y[J];               // estimated treatment effects
  real<lower=0> sigma[J]; // standard error of effect estimates
}
parameters {
  real mu;                 // population treatment effect
  real<lower=0> tau;       // standard deviation in treatment effects
  vector[J] eta;           // unscaled deviation from mu by school
}
transformed parameters {
  vector[J] theta = mu + tau * eta;        // school treatment effects
}
model {
  target += normal_lpdf(eta | 0, 1);       // prior log-density
  target += normal_lpdf(y | theta, sigma); // log-likelihood
}
```

No need to truncate priors, do that in the parameter bounds

- **BAD**: setting constraints on parameters but using a prior with other constraints

```
parameters{
    real alpha; //implies no constraints
}
model{
    alpha ~ uniform(0,1);
}
```

- **GOOD**::

```
parameters{
    real <lower=0,upper=1> alpha;
}
model{
    #alpha ~ uniform(0,1); // default uniform priors
}
```

Model

https://mc-stan.org/docs/2_26/functions-reference/

function	outcome	suffix
log probability mass function	discrete	`_lpmf`
log probability density function	continuous	`_lpdf`
log cumulative distribution function	any	`_lcdf`
log complementary cumulative distribution function	any	`_lccdf`
random number generator	any	`_rng`

For example, `normal_lpdf` is the log of the normal probability density function (pdf)

The notation

```
y ~ normal(mu, sigma);
```

provides the same (proportional) contribution to the model log density as the explicit target density increment,

```
target += normal_lpdf(y | mu, sigma);
```

In both cases, the effect is to add terms to the target log density. The only difference is that the example with the sampling (`~`) notation drops all additive constants in the log density; the constants are not necessary for any of Stan's sampling, approximation, or optimization algorithms.

Vectorization of model conditional independence assumed

```
ll = normal_lpdf(y | mu, sigma);
```

☺

is just a more efficient way to write

```
ll = 0;
for (n in 1:N)
  ll = ll + normal_lpdf(y[n] | mu[n], sigma);
```

😐

With the same arguments, the vectorized sampling statement

```
y ~ normal(mu, sigma);
```

☺

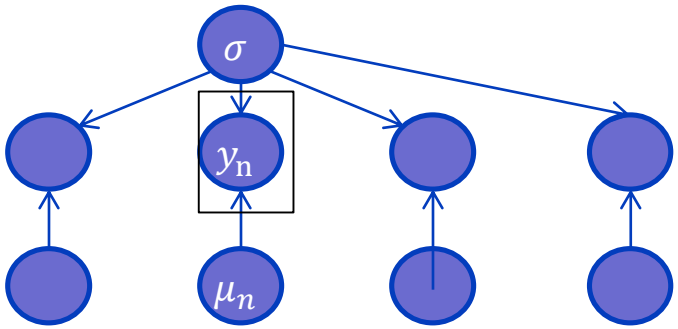
has the same effect on the total log probability as

```
for (n in 1:N)
  y[n] ~ normal(mu[n], sigma);
```

😐

Conditional independence

$$p(y_n|\boldsymbol{y}_{-n}, \boldsymbol{\mu}, \sigma) = p(y_n|\mu_n, \sigma)$$



- Dependency structure:
  - Built in:
    - MultiNormal
    - Hidden Markov models
    - Wishart/Inverse Wishart
  - Program it

Tip about priors

- No need to use conjugate priors
- Unlike BUGS (or other Gibbs based samplers), avoid super vauge priors if you can, i.e. `inv_gamma(0.1,0.1)`
- When in doubt, use a normal prior
- The Stan mailing list is very active

The Stan Forums (mc-stan.org)

Stan MCMC parameters

```
stan(file,
model_name = "anon_model",
model_code = "",
fit = NA,
data = list(),
pars = NA,
chains = 4,
iter = 2000,
warmup = floor(iter/2),
thin = 1,
init = "random",
seed = sample.int(.Machine$integer.max, 1),
algorithm = c("NUTS", "HMC", "Fixed_param"),
....)
```

… =
Specification of initial values
File name for diagnostics
Print out in R console
Save warm up
Number of cores to use
Call to other than default libraries
Parameters to control the sampler's behavior.

Alternative Model definitions

- file: The path to the Stan program. Use a .stan extension
- model_code: A character string (or variable) containing the model definition
- fit: An instance of S4 class stanfit derived from a previous fit; Time for recompiling the C++ code for the model can be saved.
- model_name: A string to use as the name of the model; (affects the name used in printed messages),
- data: A named list or environment providing the data for the model, or a character vector for all the names of objects to use as data.
- pars: A character vector specifying parameters of interest to be saved(or not). The default is to save all parameters from the model.
- include: include or exclude the parameters given by the pars argument
- Iter: total number of iterations (including warmup). The default is 2000.
- warmup: The number of samples in warmup (aka burnin) (also controls the number of iterations for which adaptation is run)
- chains: The number of Markov chains.
- thin: A positive integer specifying the period for saving samples.
- algorithm
  - “NUTS", which is the No-U-Turn sampler variant of Hamiltonian Monte Carlo (Hoffman and Gelman 2011, Betancourt 2017).
  - "HMC" (Hamiltonian Monte Carlo),
  - "Fixed_param” no sampling is performed (e.g., for simulating with in the generated quantities block).

Running Stan

N=10000

```
fit1 = stan(file = '8schools.stan',
data = schools_dat,
iter=1.1*N,
warmup=0.1*N,
thin=N/1000,
chains=4,
seed=231171,
refresh=10000)
```

```
SAMPLING FOR MODEL '8schools' NOW (CHAIN 4).
Chain 4:
Chain 4: Gradient evaluation took 0 seconds
Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
Chain 4: Adjust your expectations accordingly!
Chain 4:
Chain 4:
Chain 4: Iteration:     1 / 11000 [  0%]  (Warmup)
Chain 4: Iteration:  1001 / 11000 [  9%]  (Sampling)
Chain 4: Iteration: 11000 / 11000 [100%]  (Sampling)
Chain 4:
Chain 4:  Elapsed Time: 0.074 seconds (Warm-up)
Chain 4:                0.703 seconds (Sampling)
Chain 4:                0.777 seconds (Total)
Chain 4:
Warning messages:
1: There were 12 divergent transitions after warmup. Increasing adapt_delta above 0.8 may help. See
http://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
2: Examine the pairs() plot to diagnose sampling problems
```

😐

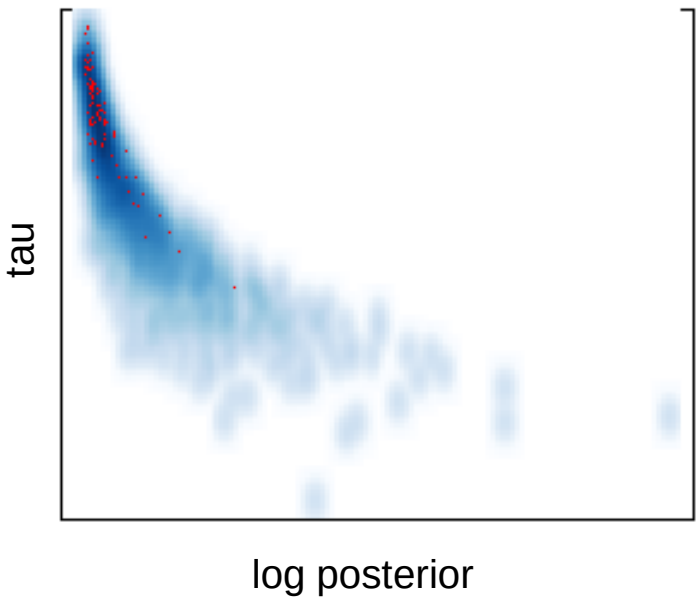
```
fit1 = stan(file = '8schools.stan',
data = schools_dat,
iter=1.1*N,
warmup=0.1*N,
thin=N/1000,
chains=4,
seed=231171,
refresh=10000,
control=list(adapt_delta=0.95))
```

```
SAMPLING FOR MODEL '8schools' NOW (CHAIN 4).
Chain 4:
Chain 4: Gradient evaluation took 0 seconds
Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
Chain 4: Adjust your expectations accordingly!
Chain 4:
Chain 4:
Chain 4: Iteration:     1 / 11000 [  0%]  (Warmup)
Chain 4: Iteration:  1001 / 11000 [  9%]  (Sampling)
Chain 4: Iteration: 11000 / 11000 [100%]  (Sampling)
Chain 4:
Chain 4:  Elapsed Time: 0.115 seconds (Warm-up)
Chain 4:                0.952 seconds (Sampling)
Chain 4:                1.067 seconds (Total)
Chain 4:
>
```

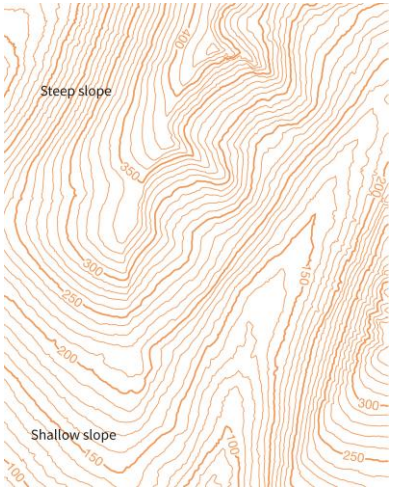
☺

Runtime Reported divergence

```
SAMPLING FOR MODEL '8schoolsDirect' NOW (CHAIN 4).
Chain 4:
Chain 4: Gradient evaluation took 0 seconds
Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
Chain 4: Adjust your expectations accordingly!
Chain 4:
Chain 4:
Chain 4: Iteration:     1 / 11000 [  0%]  (Warmup)
Chain 4: Iteration:  1001 / 11000 [  9%]  (Sampling)
Chain 4: Iteration: 11000 / 11000 [100%]  (Sampling)
Chain 4:
Chain 4:  Elapsed Time: 0.132 seconds (Warm-up)
Chain 4:                0.712 seconds (Sampling)
Chain 4:                0.844 seconds (Total)
Chain 4:
warning messages:
1: There were 111 divergent transitions after warmup. Increasing adapt_delta above 0.7 may help. See
http://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
2: Examine the pairs() plot to diagnose sampling problems
```



- Divergence: the simulated Hamiltonian trajectory does not conserve energy. (i.e. depart from the true trajectory)
- Limits the ability to explore the posterior distribution.
  (can cause irreducible issues)
- When this divergence is too high, the simulation has gone off the rails and cannot be trusted



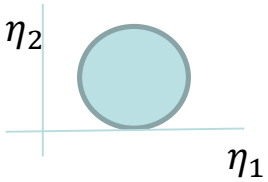
The effect of clever transformations

```stan
// saved as 8schools.stan
data {
  int<lower=0> J;          // number of schools
  real y[J];               // estimated treatment effects
  real<lower=0> sigma[J]; // standard error of effect estimates
}
parameters {
  real mu;                 // population treatment effect
  real<lower=0> tau;       // standard deviation in treatment effects
  vector[J] eta;           // unscaled deviation from mu by school
}
transformed parameters {
  vector[J] theta = mu + tau * eta;        // school treatment effects
}
model {
  target += normal_lpdf(eta | 0, 1);       // prior log-density
  target += normal_lpdf(y | theta, sigma); // log-likelihood
}
```

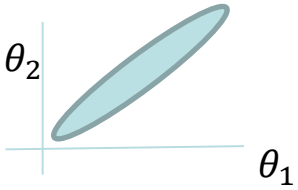
```stan
// saved as 8schoolsDirect.stan
data {
  int<lower=0> J;          // number of schools
  real y[J];               // estimated treatment effects
  real<lower=0> sigma[J]; // standard error of effect estimates
}
parameters {
  real mu;                 // population treatment effect
  real<lower=0> tau;       // standard deviation in treatment effects
  vector[J] theta;           // unscaled deviation from mu by school
}
transformed parameters {
  vector[J] eta=(theta - mu) /tau;        // school treatment effects normalized
}
model {
  target += normal_lpdf(theta | mu, tau);       // prior log-density
  target += normal_lpdf(y | theta, sigma); // log-likelihood
}
```

Sample $\eta_i \sim N(0,1)$
Compute: θ= $\mu + \tau \cdot \eta$

Sample $\theta_i \sim N(\mu, \tau^2)$
Compute: $\eta = (\theta - \mu)/\tau$



Identical models. The difference is correlations vs not
recap Exercise 42 vs 39



Running two models

```
SAMPLING FOR MODEL '8schools' NOW (CHAIN 4).
Chain 4:
Chain 4: Gradient evaluation took 0 seconds
Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
Chain 4: Adjust your expectations accordingly!
Chain 4:
Chain 4:
Chain 4: Iteration:     1 / 11000 [  0%]  (Warmup)
Chain 4: Iteration:  1001 / 11000 [  9%]  (Sampling)
Chain 4: Iteration: 11000 / 11000 [100%]  (Sampling)
Chain 4:
Chain 4:  Elapsed Time: 0.088 seconds (Warm-up)
Chain 4:                0.789 seconds (Sampling)
Chain 4:                0.877 seconds (Total)
Chain 4:
>
```

No reported errors !

```
SAMPLING FOR MODEL '8schoolsDirect' NOW (CHAIN 4).
Chain 4:
Chain 4: Gradient evaluation took 0 seconds
Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
Chain 4: Adjust your expectations accordingly!
Chain 4:
Chain 4:
Chain 4: Iteration:     1 / 11000 [  0%]  (Warmup)
Chain 4: Iteration:  1001 / 11000 [  9%]  (Sampling)
Chain 4: Iteration: 11000 / 11000 [100%]  (Sampling)
Chain 4:
Chain 4:  Elapsed Time: 0.142 seconds (Warm-up)
Chain 4:                2.068 seconds (Sampling)
Chain 4:                2.21 seconds (Total)
Chain 4:
warning messages:
1: There were 96 divergent transitions after warmup. Increasing adapt_delta above 0.9 may help. See
http://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
2: There were 1 chains where the estimated Bayesian Fraction of Missing Information was low. See
http://mc-stan.org/misc/warnings.html#bfmi-low
3: Examine the pairs() plot to diagnose sampling problems

4: Tail Effective Samples Size (ESS) is too low, indicating posterior variances and tail quantiles may be unreliable.
Running the chains for more iterations may help. See
http://mc-stan.org/misc/warnings.html#tail-ess
```

Problem with:
divergence,
Effective sample size
Bayesian Fraction of Missing Information (BFMI)

$$\mathrm{BFMI} \approx \widehat{\mathrm{BFMI}} \equiv \frac{\sum_{n=1}^{N} \left(E_n - E_{n-1}\right)^2}{\sum_{n=0}^{N} \left(E_n - \bar{E}\right)^2}.$$

Related to
lag 1 correlation
of log posterior

Result of chain:

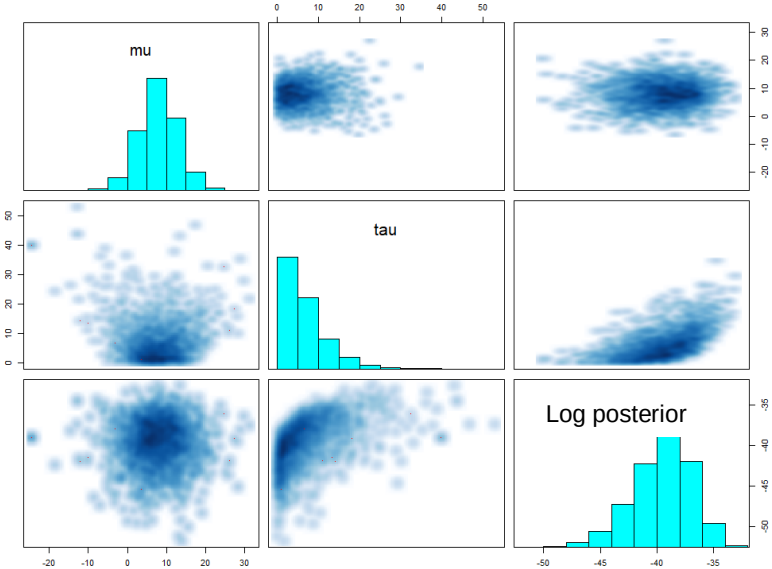
```
Inference for Stan model: 8schools.
4 chains, each with iter=11000; warmup=1000; thin=10;
post-warmup draws per chain=1000, total post-warmup draws=4000.

           mean se_mean   sd   2.5%    25%    50%    75%  97.5% n_eff Rhat
mu         7.86    0.10 5.27  -2.47   4.71   7.85  11.18  18.19  2977    1
tau        6.58    0.10 5.76   0.24   2.36   5.20   9.12  20.78  3247    1
eta[1]     0.37    0.02 0.94  -1.52  -0.25   0.39   1.04   2.16  3870    1
eta[2]     0.02    0.01 0.88  -1.76  -0.56   0.02   0.61   1.72  3988    1
eta[3]    -0.16    0.02 0.93  -1.97  -0.78  -0.16   0.45   1.70  3699    1
eta[4]    -0.02    0.01 0.89  -1.78  -0.60  -0.04   0.55   1.77  3990    1
eta[5]    -0.36    0.01 0.87  -2.07  -0.92  -0.37   0.18   1.41  3746    1
eta[6]    -0.24    0.01 0.88  -1.95  -0.83  -0.25   0.36   1.52  3931    1
eta[7]     0.34    0.01 0.91  -1.47  -0.25   0.34   0.95   2.08  3999    1
eta[8]     0.05    0.01 0.93  -1.83  -0.57   0.06   0.66   1.87  3875    1
theta[1]  11.31    0.14 8.35  -2.47   5.90  10.18  15.62  31.51  3704    1
theta[2]   8.00    0.10 6.17  -4.40   4.12   7.91  11.85  20.76  3810    1
theta[3]   6.32    0.13 7.85 -11.93   2.37   6.83  10.93  20.94  3905    1
theta[4]   7.71    0.10 6.43  -5.16   3.83   7.64  11.67  20.97  3920    1
theta[5]   5.02    0.11 6.42  -9.50   1.54   5.45   9.28  16.41  3651    1
theta[6]   6.06    0.11 6.70  -8.84   2.30   6.35  10.26  18.71  4001    1
theta[7]  10.65    0.11 6.85  -1.14   6.00   9.98  14.36  26.34  3798    1
theta[8]   8.40    0.13 7.97  -6.71   3.88   8.15  12.53  26.74  3869    1
lp__     -39.56    0.04 2.66 -45.35 -41.18 -39.36 -37.64 -35.12  3897    1

Samples were drawn using NUTS(diag_e) at Wed Apr 22 18:42:52 2020.
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).
```

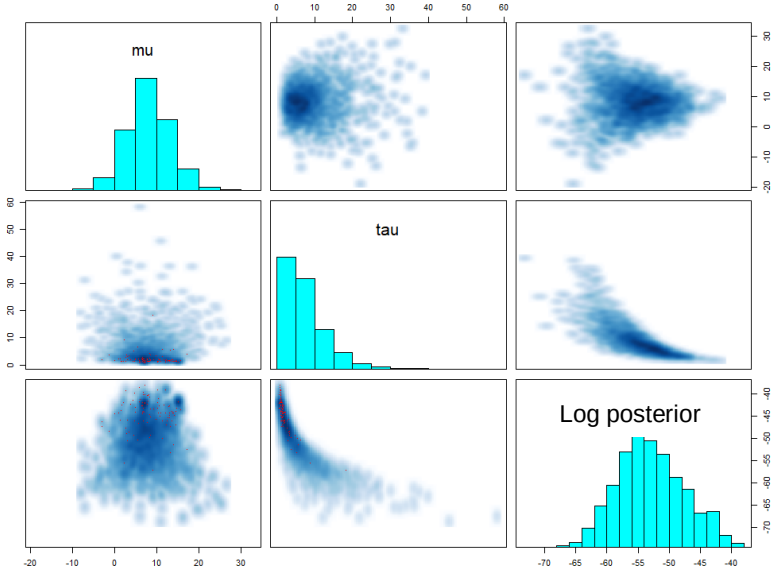
```
Inference for Stan model: 8schoolsDirect.
4 chains, each with iter=11000; warmup=1000; thin=10;
post-warmup draws per chain=1000, total post-warmup draws=4000.

           mean se_mean   sd   2.5%    25%    50%    75%  97.5% n_eff Rhat
mu         8.07    0.13 5.24  -2.29   4.72   7.89  11.41  18.56  1742 1.00
tau        7.31    0.18 5.63   0.99   3.21   5.92   9.81  21.43   942 1.01
theta[1]  11.99    0.15 8.65  -2.39   6.55  10.74  16.13  32.66  3462 1.00
theta[2]   7.89    0.11 6.52  -5.34   3.91   7.87  12.09  20.98  3227 1.00
theta[3]   5.97    0.17 8.25 -13.15   1.49   6.57  11.23  21.38  2489 1.00
theta[4]   7.68    0.14 6.89  -6.51   3.56   7.68  12.00  21.15  2306 1.00
theta[5]   4.92    0.17 6.57  -8.87   0.72   5.34   9.34  16.50  1534 1.00
theta[6]   5.81    0.15 7.08 -10.06   1.57   6.18  10.59  18.40  2153 1.00
theta[7]  11.25    0.13 6.85  -1.04   6.58  10.64  15.29  26.21  2722 1.00
theta[8]   8.74    0.14 8.29  -7.44   3.92   8.41  13.44  26.92  3292 1.00
eta[1]     0.45    0.02 0.93  -1.43  -0.14   0.47   1.05   2.26  3602 1.00
eta[2]    -0.06    0.03 0.90  -1.95  -0.62  -0.03   0.50   1.68   823 1.01
eta[3]    -0.22    0.02 0.91  -1.95  -0.82  -0.24   0.38   1.60  3151 1.00
eta[4]    -0.04    0.02 0.89  -1.78  -0.65  -0.05   0.53   1.74  1924 1.00
eta[5]    -0.40    0.02 0.84  -2.07  -0.94  -0.40   0.16   1.28  3041 1.00
eta[6]    -0.29    0.02 0.90  -2.11  -0.88  -0.29   0.30   1.47  1834 1.00
eta[7]     0.39    0.02 0.87  -1.40  -0.15   0.41   0.97   2.09  2375 1.00
eta[8]     0.07    0.02 0.94  -1.78  -0.58   0.07   0.72   1.90  2770 1.00
lp__     -52.93    0.27 5.40 -62.68 -56.74 -53.27 -49.23 -42.03   397 1.01

Samples were drawn using NUTS(diag_e) at Wed Apr 22 18:46:06 2020.
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).
```

Effective number of samples is consistently high

Effective number of samples is variable
some low some high
Have not investigated full space

Pairs() is specialized in STAN



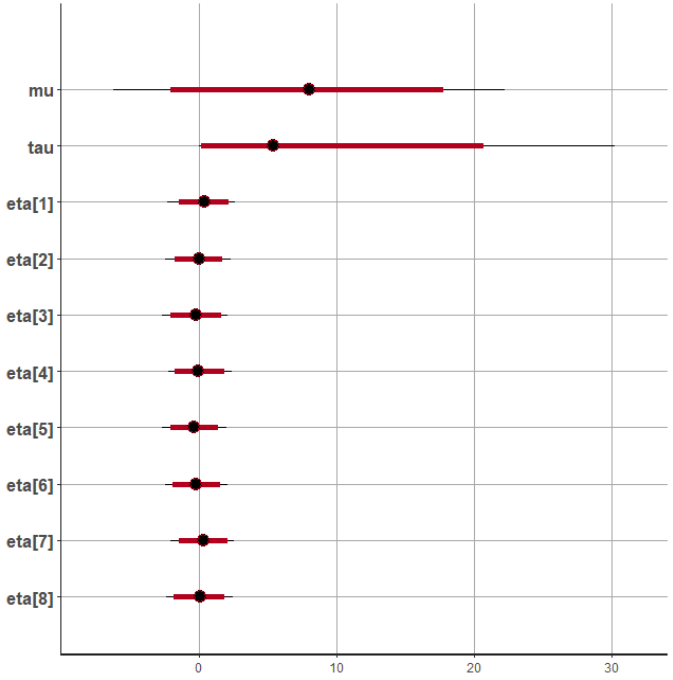


The models are the same but log posterior is different why?

Credibility intervals

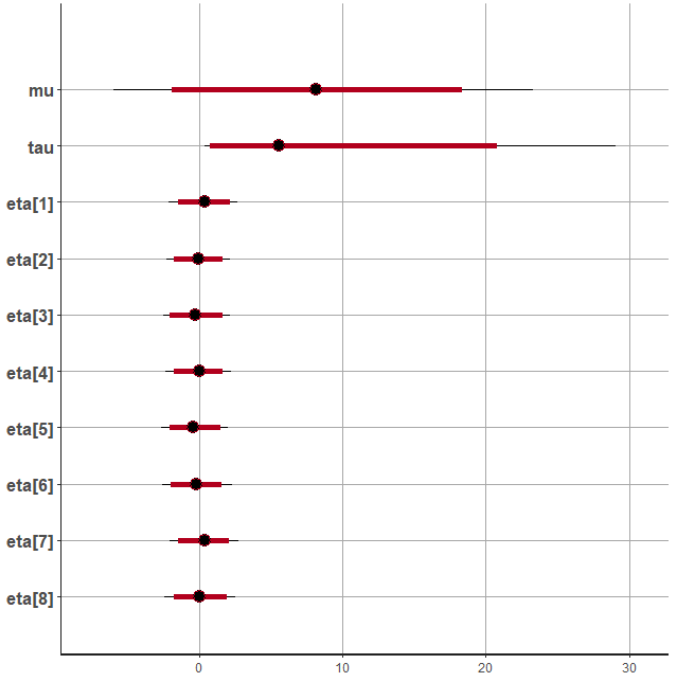
Transformed

plot(fit1,pars = c("mu", "tau", "eta"),ci_level = 0.95,outer_level = 0.99)



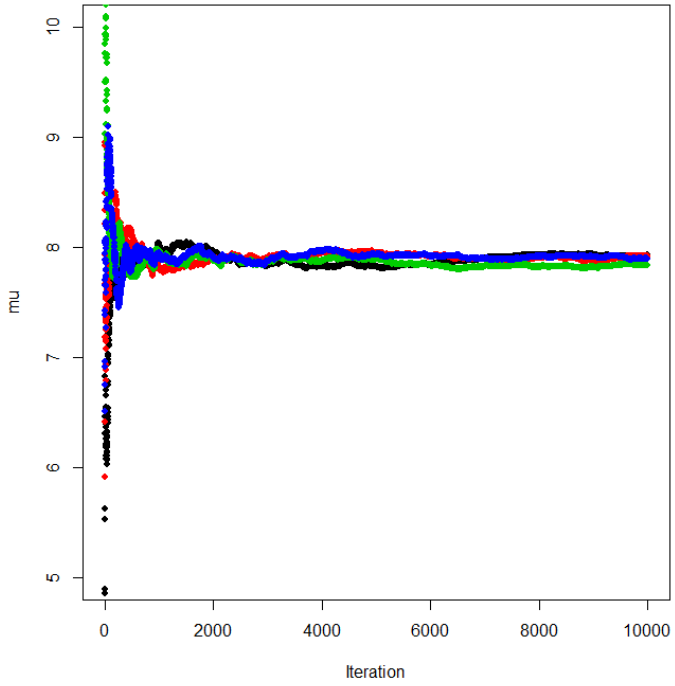
Direct

plot(fit2,pars = c("mu", "tau", "eta"),ci_level = 0.95,outer_level = 0.99)

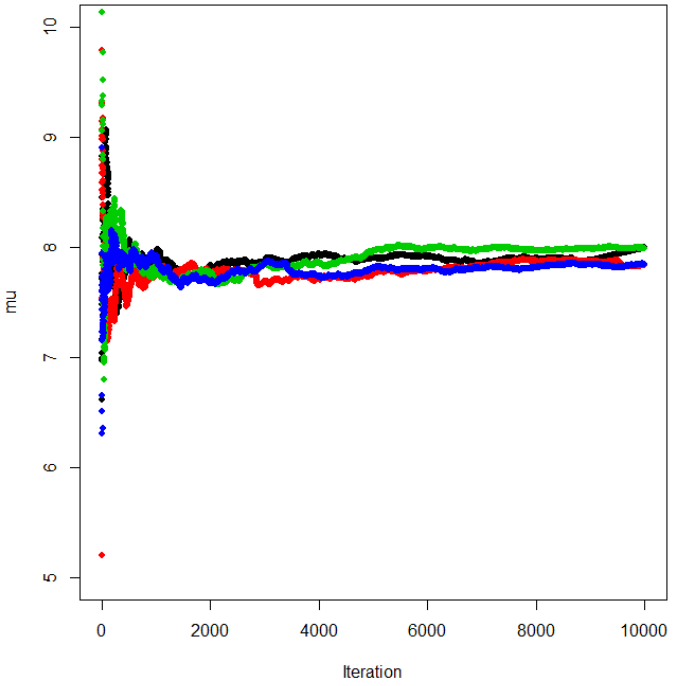


Convergence for MCMC, cumulative mean mu

Transformed

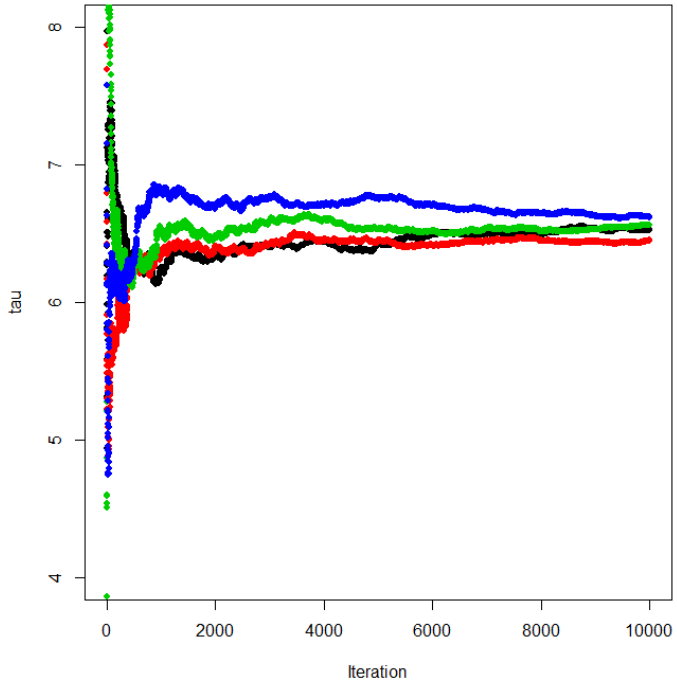


Direct

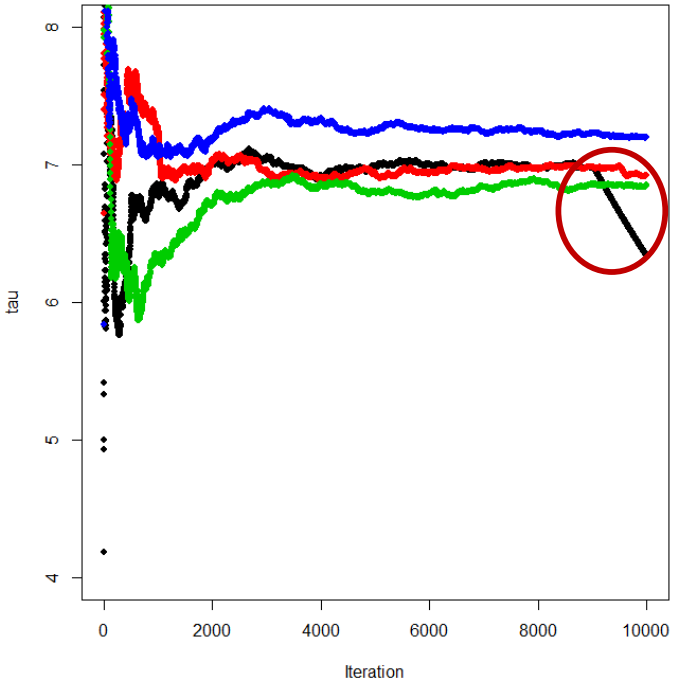


Convergence for MCMC, cumulative mean tau

Transformed

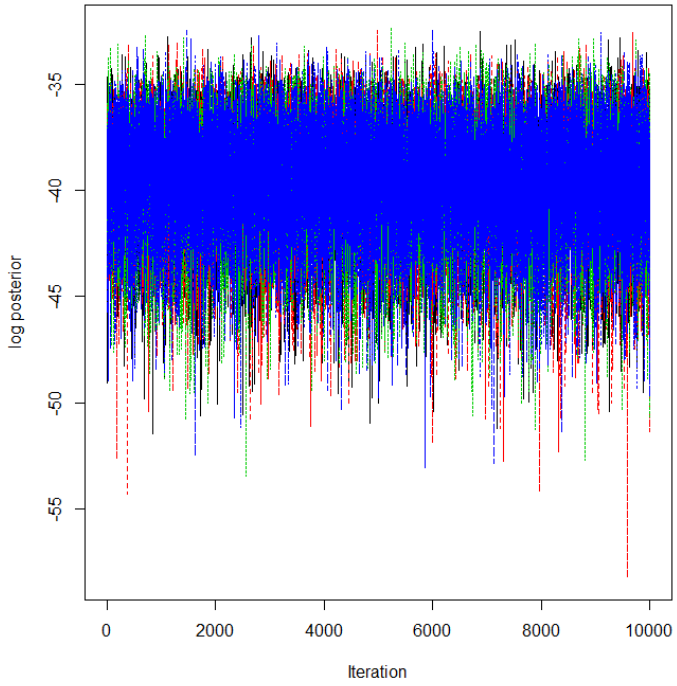


Direct

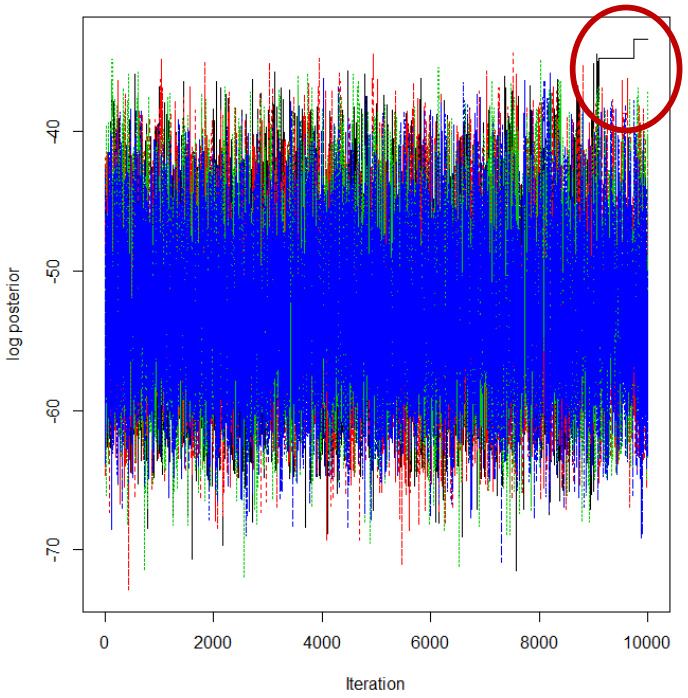


Likelihood

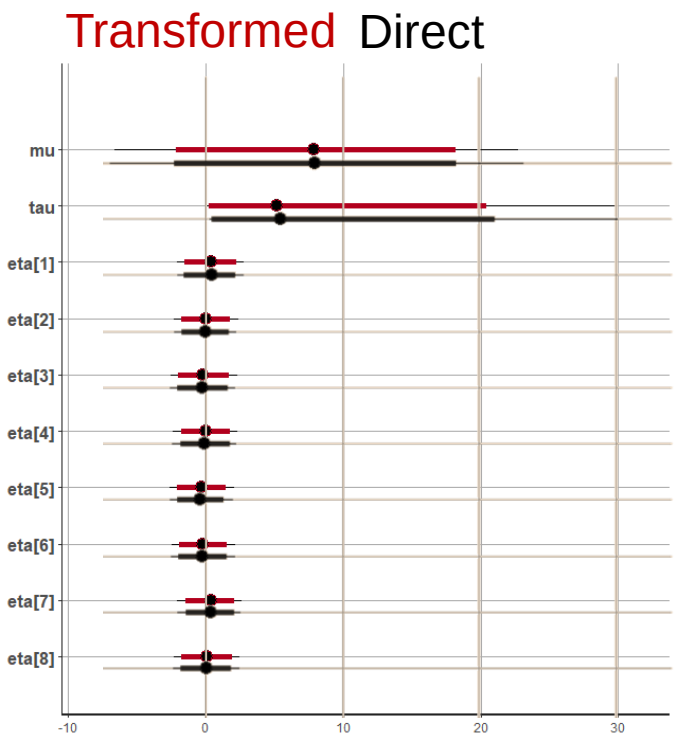
Transformed



Direct



Result MCMC long



- The Transformed approach has better convergence properties
- In a long run 100000 samples
- The difference between the two chains is not large
- Results
  - The effect of coaching is not significant 5%-level, but has a strong indication
  - The «good» school appears as a «lucky shot»

Speeding up Stan models

- Avoid repeated operations

```
// 1/alpha is repeated
for(n in 1:N)
    y[n] ~ exponential(1/alpha * x[n]);
```

- Vectorization is always faster

```
// not vectorized
for(n in 1:N)
    y[n] ~ normal(beta0 + beta1 * x[n], sigma);
//vectorized
y ~ normal(beta0 + beta1 * x, sigma);
```

- Priors: More informative the better (think better initial conditions), use MLE to get initial estimates
- Parallization: can run multiple chains if you have multiple cores, but each chain is still serial
- More advanced: Access `increment_log_prob` directly

Stan tips and tricks

#1 tip: Read the Manual! It is excellent

Other things we didn't really talk about:

- Local variables in the model block, can be used to store intermediate results
- Matrices vs arrays, Column vector vs row vector
- Constrained data types
- Functions
- Logical operations/Other types of looping
- Elementwise operators
- Built-in functions
- Print statements
- Missing data
- Prediction
- Discrete variables

Variational Bayes in STAN

- The same STAN program can be used to make inference by variational Bayes
- vb() replace stan()

$$E[h(\boldsymbol{z})|\boldsymbol{x}] \approx \int_{\boldsymbol{z}} h(\boldsymbol{z}) q^*(\boldsymbol{z}) d\boldsymbol{z}$$

$$q^*(\boldsymbol{z}) = \underset{q(\boldsymbol{z}) \in \mathcal{Q}}{\arg\min}\, \mathcal{D}(q(\boldsymbol{z}), p(\boldsymbol{z}|\boldsymbol{x}))$$

Variational Bayes in STAN

- vb(object,
  data = list(),
  pars = NA,
  include = TRUE,
  init = 'random',
  check_data = TRUE,
  sample_file = tempfile(fileext = '.csv'),
  algorithm = c("meanfield", "fullrank"),
  importance_resampling = FALSE,
  keep_every = 1,
  ...)

Making a stanmodel

- stan_model(file,
  model_name = "anon_model",
  model_code = "",
  …)

- object An object of class stanmodel.
- data A named list or environment providing the data for the model
- check_data If TRUE the data will be preprocessed

```
data {
  int<lower=0> J;
  real y[J];
  real<lower=0> sigma[J];
}
```

- algorithm
  - "meanfield" uses a fully factorized Gaussian for the approximation
  - "fullrank", uses a Gaussian with a full-rank covariance matrix for the approximation.
- importance_resampling: If TRUE we do importance resampling to adjust the draws at the optimum to be more like draws from the posterior distribution

StanModel

- file: The path to the Stan program. Use a .stan extension on file
- model_code: A character string either containing the model definition
- model_name: A string to use as the name of the model; (affects the name used in printed messages)

Running Variational Bayes

```
> fitvb1b= vb(stanModel8s, data = schools_dat,algorithm = "fullrank")
Chain 1: ------------------------------------------------------------
Chain 1: EXPERIMENTAL ALGORITHM:
Chain 1:   This procedure has not been thoroughly tested and may be unstable
Chain 1:   or buggy. The interface is subject to change.
Chain 1: ------------------------------------------------------------
Chain 1:
Chain 1:
Chain 1:
Chain 1: Gradient evaluation took 0 seconds
Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
Chain 1: Adjust your expectations accordingly!
Chain 1:
Chain 1:
Chain 1: Begin eta adaptation.
Chain 1: Iteration:   1 / 250 [  0%]  (Adaptation)
Chain 1: Iteration:  50 / 250 [ 20%]  (Adaptation)
Chain 1: Iteration: 100 / 250 [ 40%]  (Adaptation)
Chain 1: Iteration: 150 / 250 [ 60%]  (Adaptation)
Chain 1: Iteration: 200 / 250 [ 80%]  (Adaptation)
Chain 1: Success! Found best value [eta = 1] earlier than expected.
Chain 1:
Chain 1: Begin stochastic gradient ascent.
Chain 1:   iter             ELBO   delta_ELBO_mean   delta_ELBO_med   notes
Chain 1:    100          -29.037             1.000            1.000
Chain 1:    200          -26.790             0.542            1.000
Chain 1:    300          -26.172             0.369            0.084
Chain 1:    400          -25.928             0.279            0.084
```

...

```
Chain 1:   5200          -26.115             0.020            0.021
Chain 1:   5300          -25.679             0.019            0.018
Chain 1:   5400          -26.180             0.021            0.019
Chain 1:   5500          -25.561             0.022            0.021
Chain 1:   5600          -25.602             0.022            0.021
Chain 1:   5700          -27.570             0.028            0.024
Chain 1:   5800          -25.933             0.031            0.024
Chain 1:   5900          -25.951             0.029            0.024
Chain 1:   6000          -25.604             0.026            0.021
Chain 1:   6100          -25.645             0.023            0.019
Chain 1:   6200          -25.851             0.022            0.017
Chain 1:   6300          -25.938             0.021            0.014
Chain 1:   6400          -25.794             0.019            0.008   MEDIAN ELBO CONVERGED
Chain 1:
Chain 1: Drawing a sample of size 1000 from the approximate posterior...
Chain 1: COMPLETED.
```

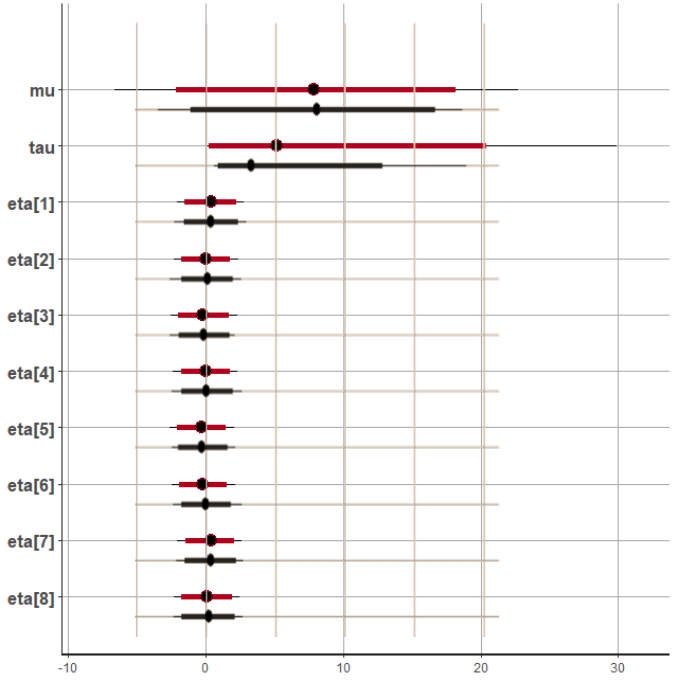
```
>  print(fitvb1a)
Inference for Stan model: 8schools.
1 chains, each with iter=1000; warmup=0; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=1000.

          mean se_mean   sd  2.5%   25%   50%   75% 97.5% n_eff khat
mu        7.41     NaN 3.93 -0.56  4.76  7.43  9.87 15.40   NaN 0.92
tau       4.19     NaN 3.91  0.64  1.71  3.00  5.03 15.75   NaN 0.89
eta[1]    0.32     NaN 0.94 -1.68 -0.28  0.35  0.95  2.10   NaN 0.66
eta[2]   -0.07     NaN 0.88 -1.75 -0.65 -0.06  0.52  1.64   NaN 0.80
eta[3]   -0.16     NaN 0.93 -1.90 -0.75 -0.16  0.48  1.66   NaN 0.72
eta[4]    0.05     NaN 0.99 -1.84 -0.57  0.08  0.69  2.07   NaN 0.84
eta[5]   -0.27     NaN 0.79 -1.81 -0.79 -0.28  0.23  1.33   NaN 0.71
eta[6]   -0.18     NaN 1.03 -2.15 -0.88 -0.19  0.55  1.83   NaN 0.74
eta[7]    0.37     NaN 0.90 -1.35 -0.22  0.38  0.96  2.19   NaN 0.70
eta[8]    0.22     NaN 0.91 -1.51 -0.40  0.21  0.82  2.04   NaN 0.65
theta[1]  8.79     NaN 6.58 -3.05  4.94  8.43 12.25 22.22   NaN 0.77
theta[2]  7.08     NaN 6.11 -5.59  3.78  7.25 10.74 18.82   NaN 0.81
theta[3]  6.83     NaN 6.76 -7.22  3.13  7.25 10.54 18.58   NaN 0.72
theta[4]  7.65     NaN 7.49 -7.21  3.95  7.55 10.90 23.30   NaN 0.74
theta[5]  6.13     NaN 5.81 -6.67  3.09  6.42  9.61 16.27   NaN 0.79
theta[6]  6.75     NaN 7.13 -7.86  3.27  6.92 10.37 20.69   NaN 0.66
theta[7]  8.84     NaN 6.41 -2.49  5.18  8.50 11.88 21.51   NaN 0.63
theta[8]  8.34     NaN 6.35 -3.80  4.72  8.25 11.59 22.64   NaN 0.63
lp__      0.00     NaN 0.00  0.00  0.00  0.00  0.00  0.00   NaN 0.74

Approximate samples were drawn using VB(meanfield) at Thu Apr 23 01:46:05 2020.
We recommend genuine 'sampling' from the posterior distribution for final inferences!
```

We recommend genuine 'sampling' from the posterior distribution for final inferences!

Results are not trusted by developer ☹ (they are statisticians)

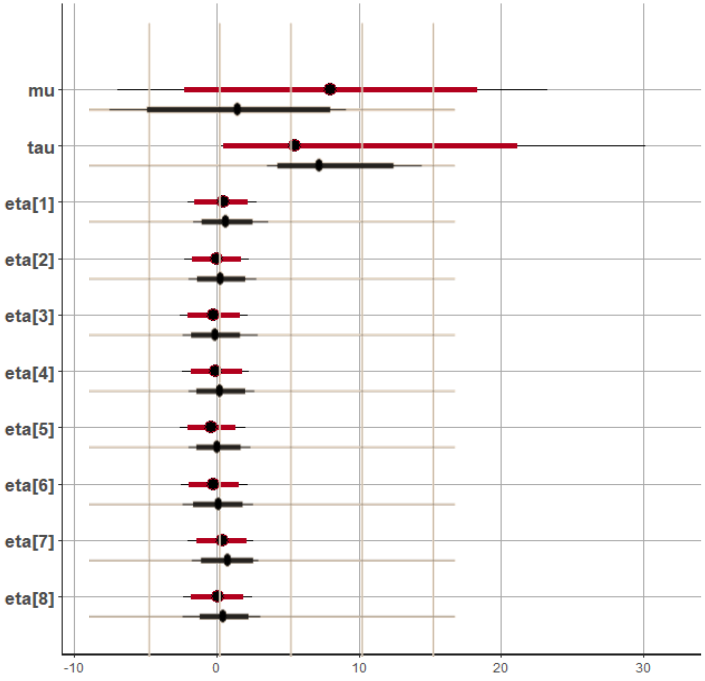
Full rank VB - compared to MCMC

Transformed



MCMC / VB full rank

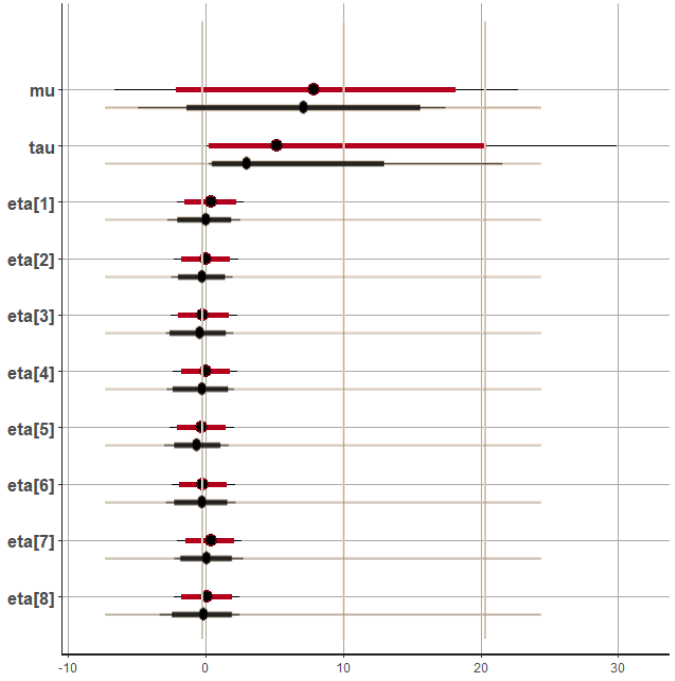
Direct



MCMC / VB full rank

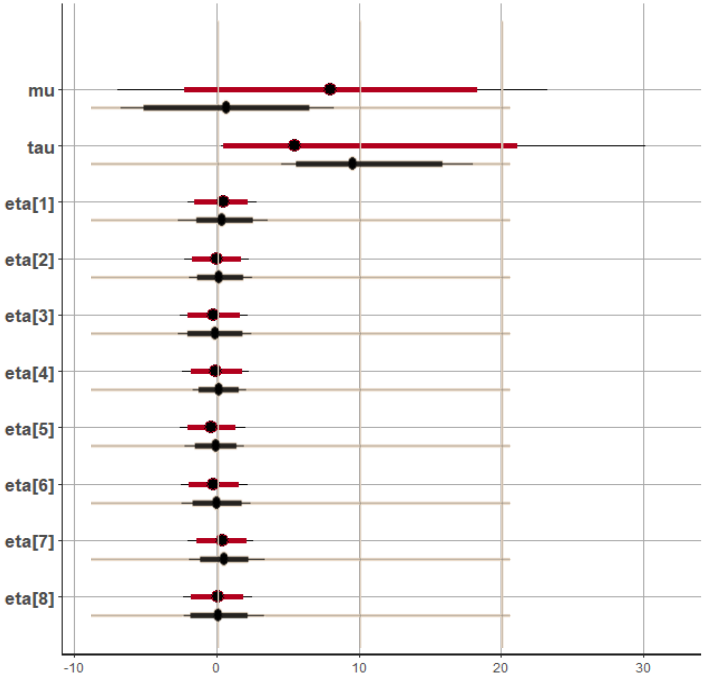
Mean field VB - compared to MCMC

Transformed



MCMC / VB mean field

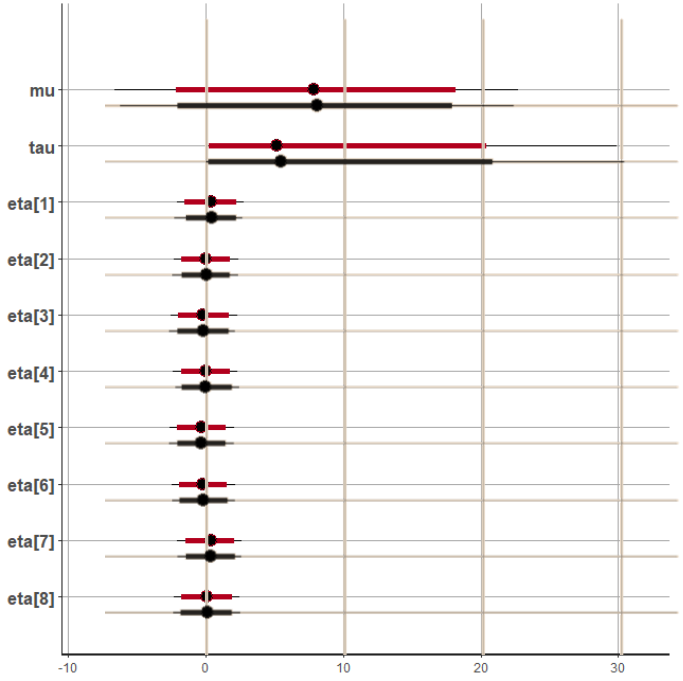
Direct



MCMC / VB mean field

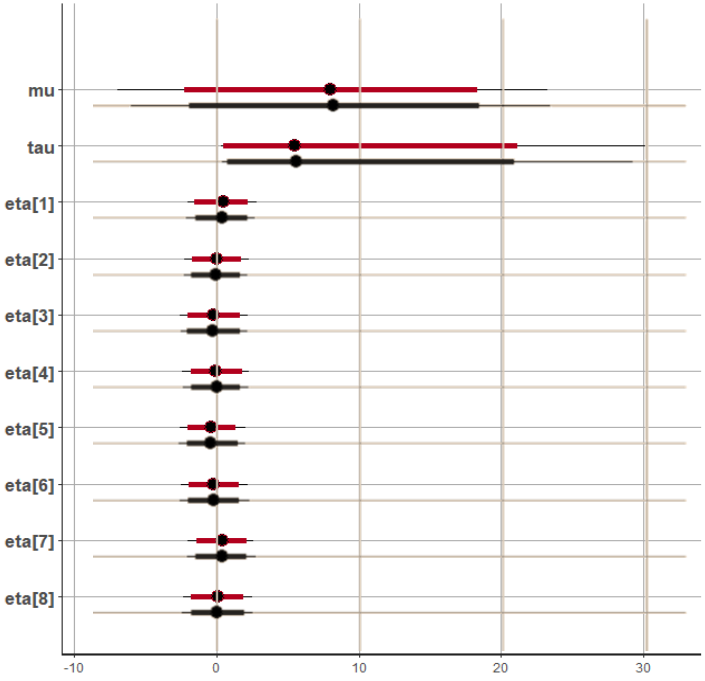
«Short» MCMC compared to MCMC

Transformed



MCMC long / MCMC short

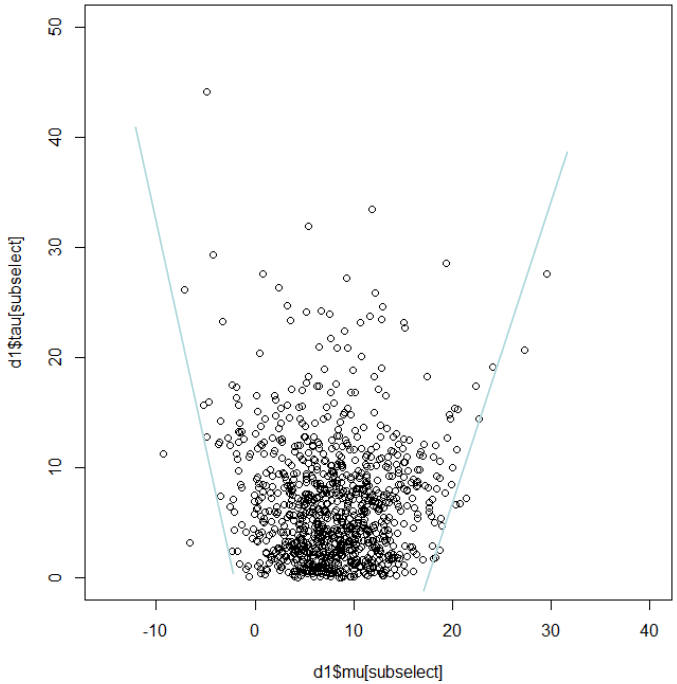
Direct



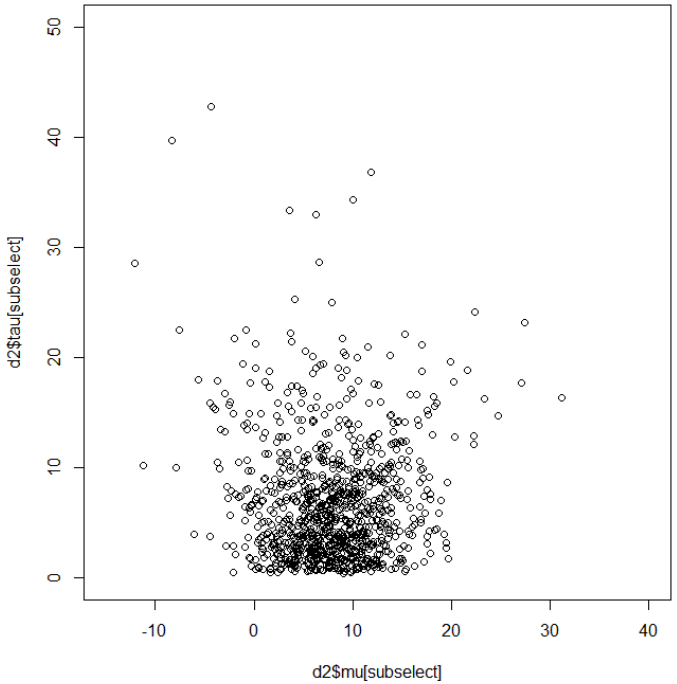
MCMC long / MCMC short

Joint distribution MCMC

Transformed

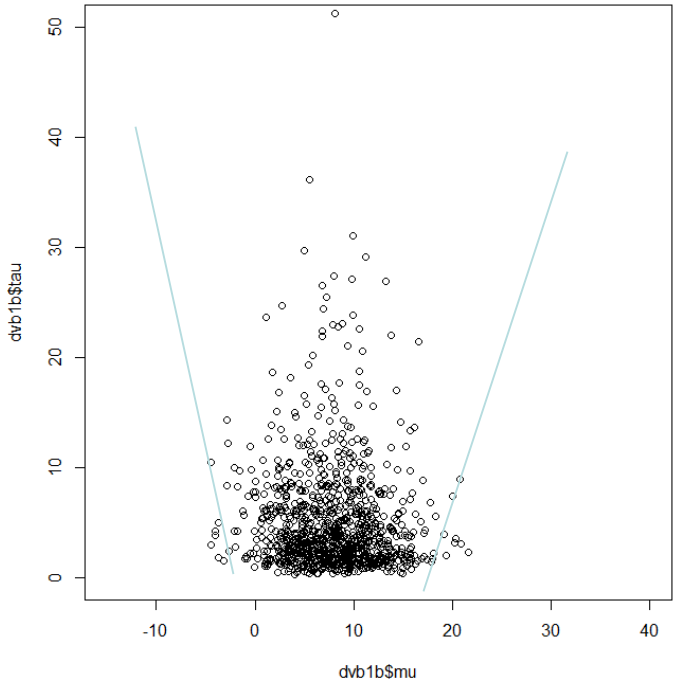


Direct

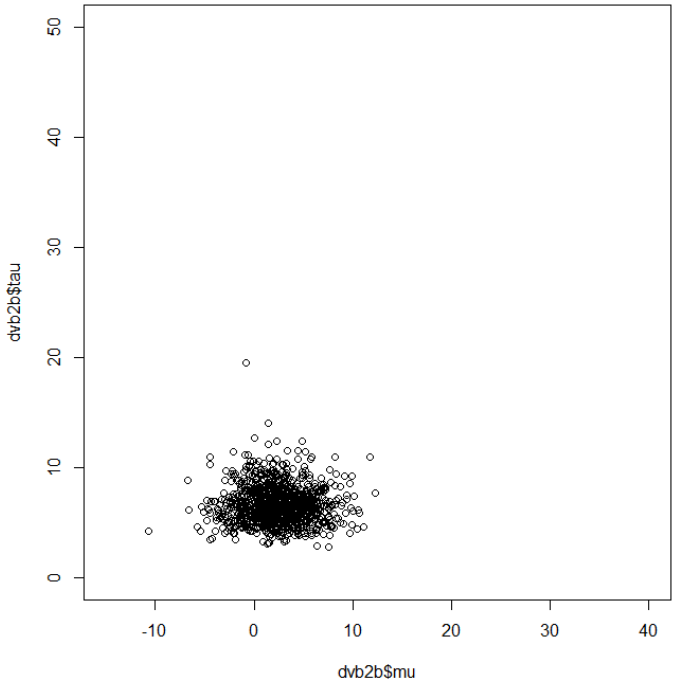


Joint distribution VB full rank

Transformed

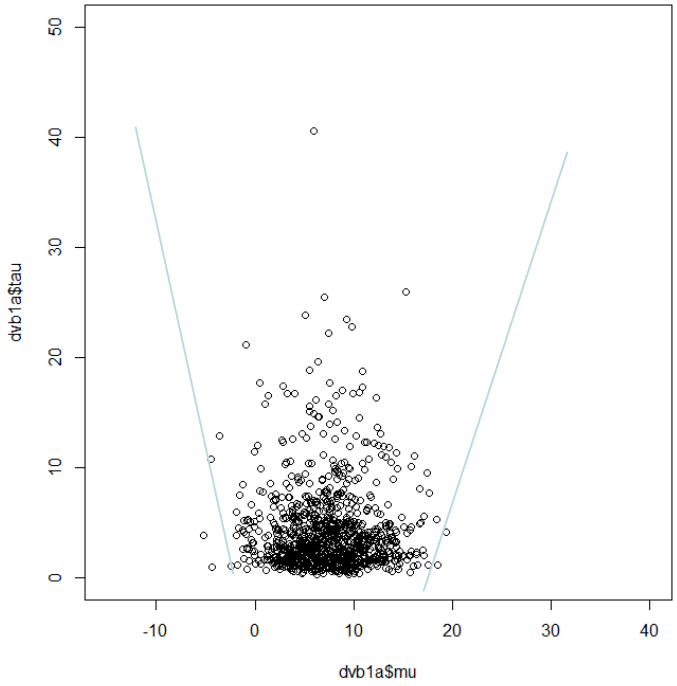


Direct

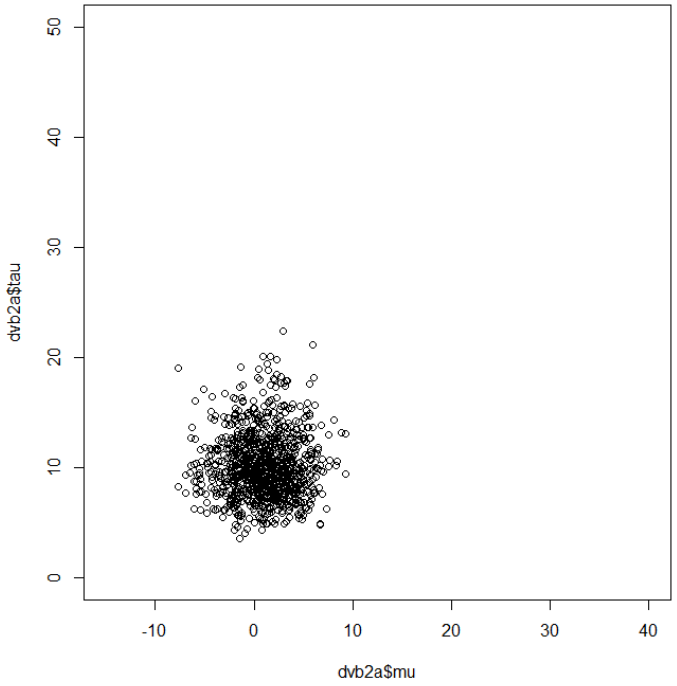


Joint distribution VB mean filed

Transformed



Direct



STAN

- Possible to do inference with
  - MCMC
  - VB

  Same STAN program
- Quite robust/adaptive MCMC sampler
- Still work to be done for VB
  - Methodologically weaknesses
  - Implementational weaknesses
- Test show
  - VB underestimate uncertainty
  - VB sensitive to parameterization
  - VB can be “far off”
  - NUTS is also sensitive to parameterization, but can compensate by longer chain

References

- Slides: Introduction to Stan by Cameron Bracken at University of Colorado Boulder February 2015
- https://github.com/stan-dev/rstan/wiki/RStan-Getting-Started
- https://mc-stan.org/docs/2_26/reference-manual
- https://mc-stan.org/docs/2_26/functions-reference
- https://mc-stan.org/rstan/reference/
- https://faculty.ucr.edu/~jflegal/203/STAN_tutorial.pdf
- https://mc-stan.org/users/documentation/case-studies/divergences_and_bias.html
- Betancourt, Michael 2016a. “Diagnosing Suboptimal Cotangent Disintegrations in Hamiltonian Monte Carlo. ” *arXiv* 1604.00695. https://arxiv.org/abs/1604.00695.
- Rubin DB (1981). “Estimation in Parallel Randomized Experiments.” Journal of Educational Statistics, 6, 377–400