R Stan examples

true

June 09, 2025 at 23:42

Contents

Simple examples

```
library(rstan)
```

```
## Loading required package: StanHeaders
```

```
##
## rstan version 2.32.7 (Stan version 2.32.2)
```

```
## For execution on a local, multicore CPU with excess RAM we recommend calling
## options(mc.cores = parallel::detectCores()).
## To avoid recompilation of unchanged Stan programs, we recommend calling
## rstan_options(auto_write = TRUE)
## For within-chain threading using `reduce_sum()` or `map_rect()` Stan functions,
## change `threads_per_chain` option:
## rstan_options(threads_per_chain = 1)
```

```
library(spida2)
```

```
##
## Attaching package: 'spida2'
```

```
## The following object is masked from 'package:base':
##
##     grepv
```

```
library(p3d)
```

```
## Loading required package: rgl
```

```
##
## Attaching package: 'p3d'
```

```
## The following objects are masked from 'package:spida2':
##
##     cell, center, ConjComp, dell, disp, ell, ell.conj, ellbox, ellplus,
##     ellpt, ellptc, ellpts, ellptsc, elltan, elltanc, na.include, uv
```

```
rstan_options(auto_write = TRUE)
options(mc.cores = parallel::detectCores())
# windowsFonts(Arial=windowsFont("TT Arial")) # not necessary for unix and Mac
```

```
data(hw)
data(hwoutliers)
if(interactive) {
  Init3d(cex = 2)
  Plot3d(Health ~ Weight + Height | Outlier, hwoutliers)
  fg()
  Id3d(labels = hwoutliers$Outlier, pad = 3)
}

fit <- lm(Health ~ Weight + Height, hwoutliers,
          subset = Outlier == 'none')
summary(fit)
```

```
##
## Call:
## lm(formula = Health ~ Weight + Height, data = hwoutliers, subset = Outlier ==
##     "none")
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.171701 -0.053776  0.008552  0.058974  0.166962
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.02009    0.08529  11.960 5.02e-08 ***
## Weight      -0.65487    0.12380  -5.290 0.000192 ***
## Height       0.72500    0.13639   5.315 0.000184 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1008 on 12 degrees of freedom
## Multiple R-squared:  0.7182, Adjusted R-squared:  0.6712
## F-statistic: 15.29 on 2 and 12 DF,  p-value: 0.000501
```

```
if(interactive) {
  Fit3d(fit, alpha = .5)
  fg()
}
fit3 <- lm(Health ~ Weight + Height, hwoutliers,
           subset = Outlier %in% c('Type 3', 'none'))

if(interactive) Fit3d(fit3, col = 'magenta')
```

Generic Stan model for regression with improper uniform prior on betas and uniform on sigma

```
library(rstan)
rstan_options(auto_write = TRUE)
options(mc.cores = parallel::detectCores())
# windowsFonts(Arial=windowsFont("TT Arial"))


reg_model <-
"
data {
  int N;   // number of observations
```

```
  int P;   // number of columns of X matrix (including intercept)
  matrix[N,P] X;   // X matrix including intercept
  vector[N] y;  // response
}
parameters {
  vector[P] beta;   // default uniform prior if nothing specied in model
  real <lower=0> sigma;
}
model {
  y ~ normal( X * beta, sigma ); // note that * is matrix mult.
                                 // For elementwise multiplication use .*
}
"
# Create reg_model 'dynamic shared object module' which is compiled C++ code
# that generates HMC samples from the posterior distribution

system.time(
   reg_model_dso <- stan_model( model_code = reg_model)
)
```

```
## Trying to compile a simple C file
```

```
## Running /usr/lib/R/bin/R CMD SHLIB foo.c
## using C compiler: ‘gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0’
## gcc -I"/usr/share/R/include" -DNDEBUG   -I"/home/georges/R/x86_64-pc-linux-gnu-library/4.5/Rcpp/inclu
## In file included from /home/georges/R/x86_64-pc-linux-gnu-library/4.5/RcppEigen/include/Eigen/Core:19
##                  from /home/georges/R/x86_64-pc-linux-gnu-library/4.5/RcppEigen/include/Eigen/Dense:1
##                  from /home/georges/R/x86_64-pc-linux-gnu-library/4.5/StanHeaders/include/stan/math/p
##                  from <command-line>:
## /home/georges/R/x86_64-pc-linux-gnu-library/4.5/RcppEigen/include/Eigen/src/Core/util/Macros.h:679:10
##   679 | #include <cmath>
##       |          ^~~~~~~
## compilation terminated.
## make: *** [/usr/lib/R/etc/Makeconf:202: foo.o] Error 1
```

```
##    user  system elapsed
##  27.846   1.112  28.959
```

```
#
# Prepare the data list
# striplevels
#
head( dat <- subset(hwoutliers, Outlier == 'none') )
```

```
##   Height Weight Health Type Outlier
## 1 0.6008 0.3355  1.280    0    none
## 2 0.9440 0.6890  1.208    0    none
## 3 0.6150 0.6980  1.036    0    none
## 4 1.2340 0.7617  1.395    0    none
## 5 0.7870 0.8910  0.912    0    none
## 6 0.9150 0.9330  1.175    0    none
```

```
head( Xmat <- model.matrix(Health ~ Weight*Height, dat) )
```

```
##   (Intercept) Weight Height Weight:Height
## 1           1 0.3355 0.6008     0.2015684
```

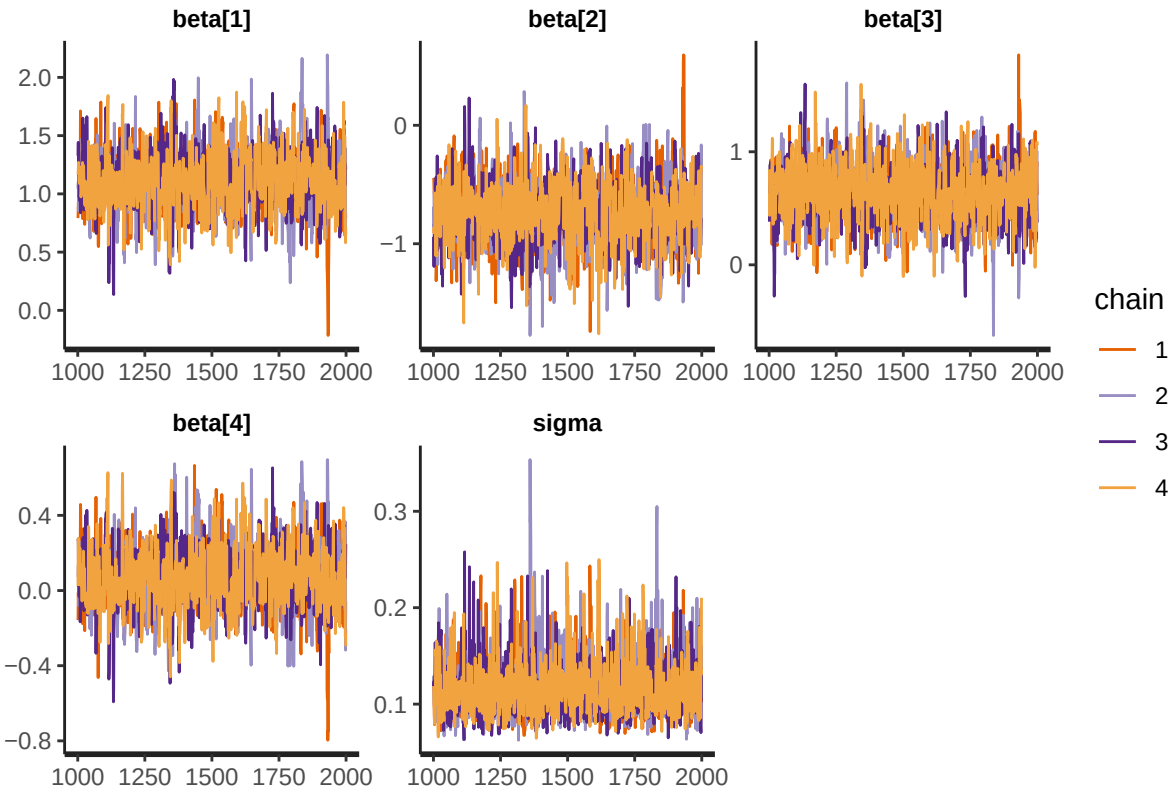
```
## 2           1 0.6890 0.9440     0.6504160
## 3           1 0.6980 0.6150     0.4292700
## 4           1 0.7617 1.2340     0.9399378
## 5           1 0.8910 0.7870     0.7012170
## 6           1 0.9330 0.9150     0.8536950
```

```
dat_list <- list(N = nrow(Xmat), P = ncol(Xmat),
                 X = Xmat, y = dat$Health)
#
# Sample from the posterior distribution
#
system.time(
  fit_stan <- sampling(reg_model_dso, dat_list)
)
```

```
##    user  system elapsed
##   1.536   0.589   1.137
## ain 4:
## Chain 4: Gradient evaluation took 1.2e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.12 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
```

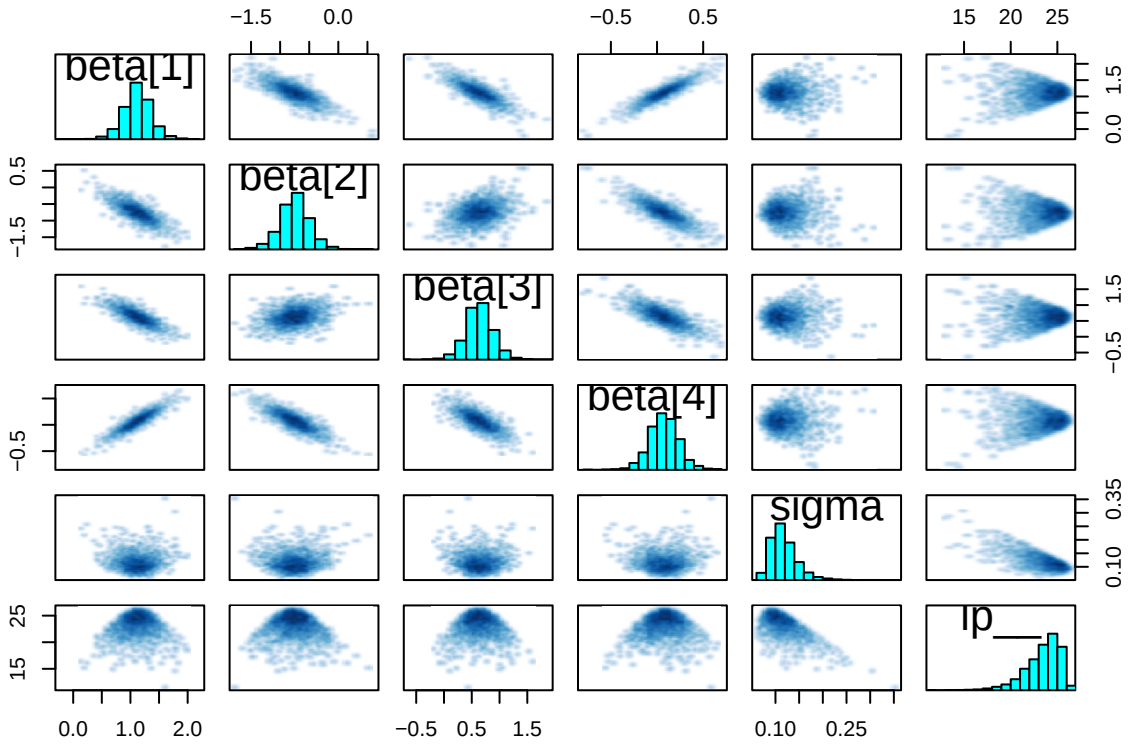
```
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.042 seconds (Warm-up)
## Chain 1:                0.04 seconds (Sampling)
## Chain 1:                0.082 seconds (Total)
## Chain 1:
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.041 seconds (Warm-up)
## Chain 3:                0.042 seconds (Sampling)
## Chain 3:                0.083 seconds (Total)
## Chain 3:
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.042 seconds (Warm-up)
## Chain 2:                0.048 seconds (Sampling)
## Chain 2:                0.09 seconds (Total)
## Chain 2:
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.045 seconds (Warm-up)
## Chain 4:                0.054 seconds (Sampling)
## Chain 4:                0.099 seconds (Total)
## Chain 4:
```

```
names(fit_stan)
traceplot(fit_stan)
```




```
pairs(fit_stan,pars=c('beta','sigma','lp__')) # note log-lik profiles
```

```
## Warning in par(usr): argument 1 does not name a graphical parameter
## Warning in par(usr): argument 1 does not name a graphical parameter
## Warning in par(usr): argument 1 does not name a graphical parameter
## Warning in par(usr): argument 1 does not name a graphical parameter
## Warning in par(usr): argument 1 does not name a graphical parameter
## Warning in par(usr): argument 1 does not name a graphical parameter
```



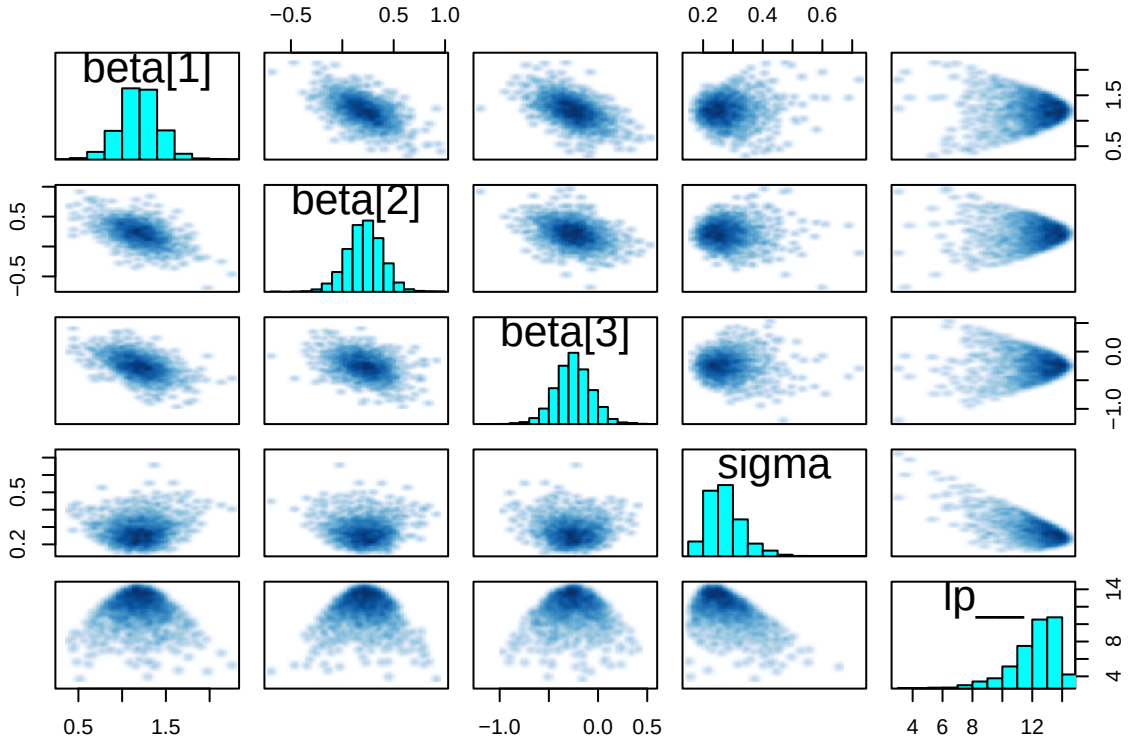

```
get_posterior_mean(fit_stan)[,5]

# Define a function that returns the posterior mean prediction

fun <- function(stanfit) {
  post <- get_posterior_mean(stanfit)
  beta <- post[,ncol(post)]  # use last column (all chains)
  function(Weight, Height) beta[1] + beta[2] * Weight + beta[3] * Height
}
fun(fit_stan)  # this is a closure
fun(fit_stan)(2,3)
if(interactive) Fit3d(fun(fit_stan), col = 'black')

#
# Including Type 3 outlier
#
dat3 <- subset(hwoutliers, Outlier %in% c('none','Type 3'))
dat3

head( Xmat3 <- model.matrix(Health ~ Weight + Height, dat3) )

dat3_list <- list(N = nrow(Xmat3), P = ncol(Xmat3),
                  X = Xmat3, y = dat3$Health)

system.time(
  fit3_stan <- sampling(reg_model_dso, dat3_list)  # same dso
)
```

```
##  1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
```

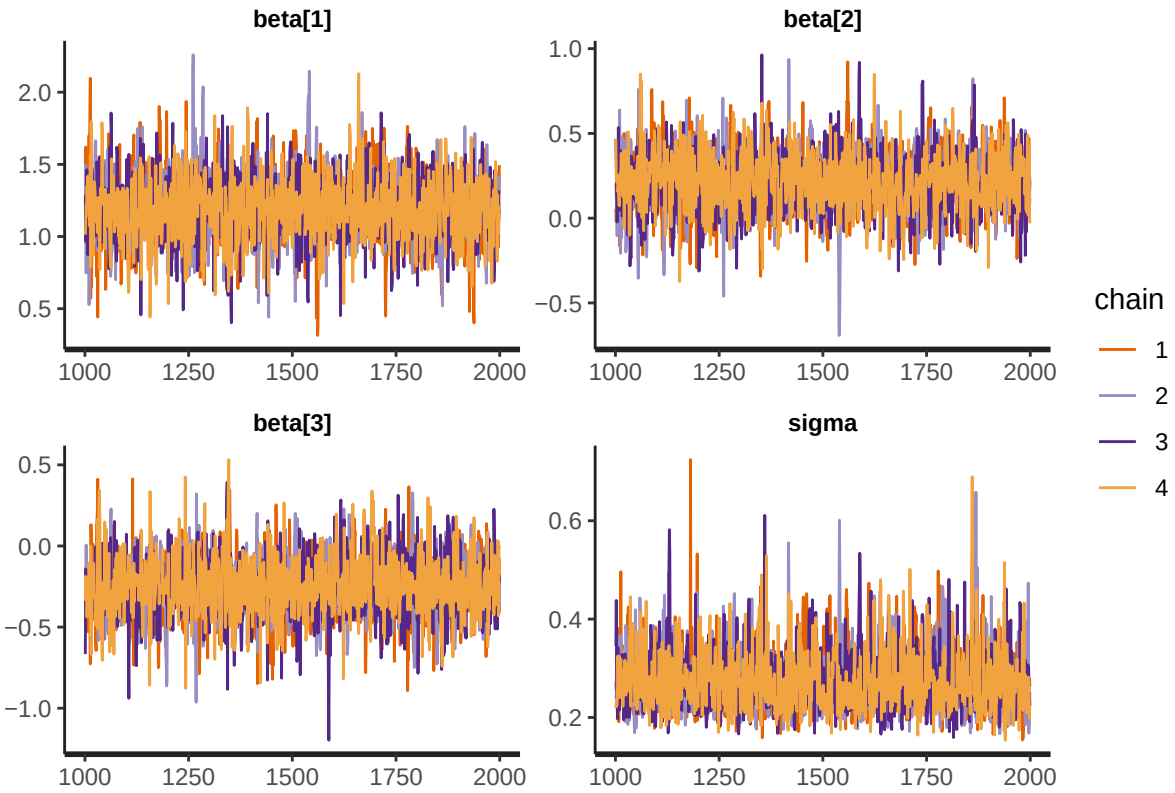
```
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.016 seconds (Warm-up)
## Chain 3:                0.017 seconds (Sampling)
## Chain 3:                0.033 seconds (Total)
## Chain 3:
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.016 seconds (Warm-up)
## Chain 4:                0.017 seconds (Sampling)
## Chain 4:                0.033 seconds (Total)
## Chain 4:
```

```
print(fit3_stan)
pairs(fit3_stan,pars=c('beta','sigma','lp__'))
```

```
## Warning in par(usr): argument 1 does not name a graphical parameter
## Warning in par(usr): argument 1 does not name a graphical parameter
## Warning in par(usr): argument 1 does not name a graphical parameter
## Warning in par(usr): argument 1 does not name a graphical parameter
## Warning in par(usr): argument 1 does not name a graphical parameter
```




```
traceplot(fit3_stan)
```




```
get_posterior_mean(fit3_stan)[,5]

if(interactive) Fit3d(fun(fit3_stan), col = 'black')

#
#  So far very boring
#  -- nothing new, MCMC with normal error and uniform prior
#     like OLS
#
#  It's as easy as pi to use a different family of distributions
#  for error.
#
#  Exactly the same except for the error distribution and
#  add nu for degrees for freedom for t distribution

robust_model <-
"
data {
  int N;   // number of observations
  int P;   // number of columns of X matrix (including intercept)
  matrix[N,P] X;   // X matrix including intercept
  vector[N] y;  // response
  int nu;   // degrees for freedom for student_t
}
parameters {
  vector[P] beta;   // default uniform prior if nothing specied in model
  real <lower=0> sigma;
}
model {
```

```
  y ~ student_t(nu, X * beta, sigma );
}
"

system.time(
  robust_model_dso <- stan_model(model_code = robust_model)
)
```

```
## Trying to compile a simple C file
```

```
## en/include/Eigen/src/Core/util/Macros.h:679:10: fatal error: cmath: No such file or directory
##   679 | #include <cmath>
##       |          ^~~~~~~
## compilation terminated.
## make: *** [/usr/lib/R/etc/Makeconf:202: foo.o] Error 1
```

```
##    user  system elapsed
##  29.242   1.055  30.319
```

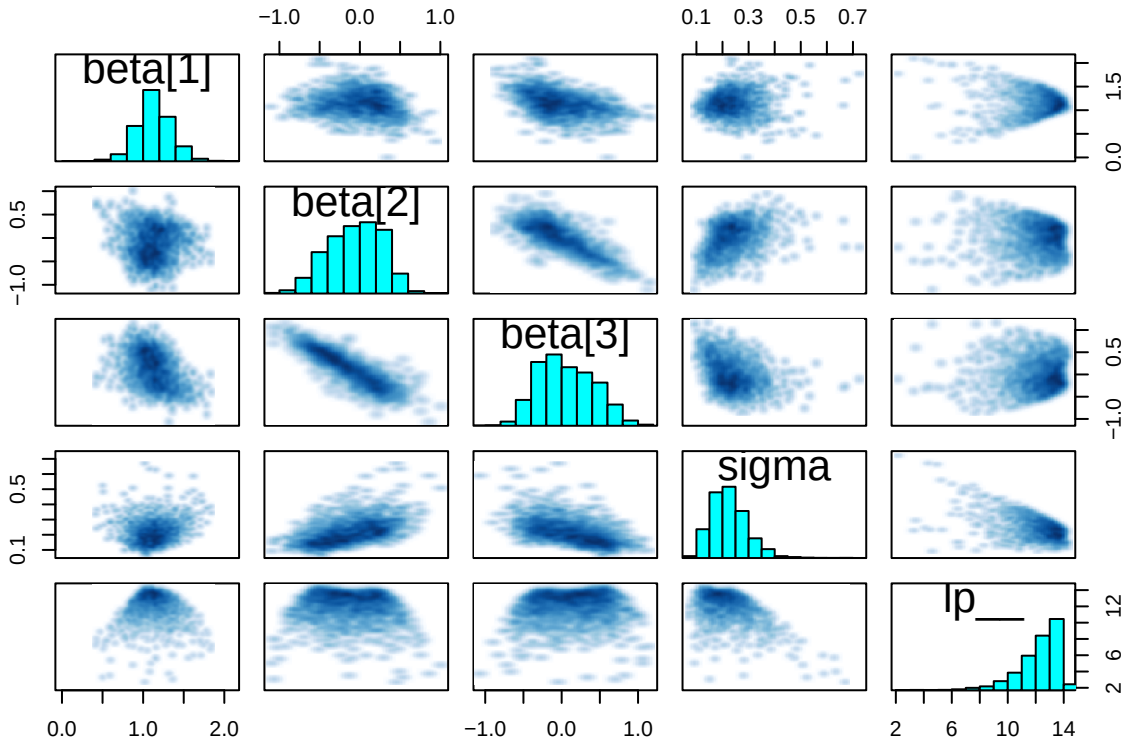
```
fit3_stan_6 <- sampling(robust_model_dso, c(dat3_list, nu = 6))

fit3_stan_6
```

```
## Inference for Stan model: anon_model.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##          mean se_mean   sd  2.5%   25%   50%   75% 97.5% n_eff Rhat
## beta[1]  1.13    0.01 0.21  0.73  1.00  1.13  1.26  1.57  1622    1
## beta[2] -0.06    0.01 0.33 -0.72 -0.31 -0.03  0.20  0.51   879    1
## beta[3]  0.06    0.01 0.36 -0.55 -0.22  0.03  0.33  0.75   846    1
## sigma    0.23    0.00 0.07  0.12  0.18  0.22  0.27  0.38  1087    1
## lp__    12.22    0.04 1.56  8.26 11.50 12.62 13.36 14.02  1225    1
##
## Samples were drawn using NUTS(diag_e) at Mon Jun  9 23:43:36 2025.
## 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).
## %]  (Warmup)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.023 seconds (Warm-up)
## Chain 1:                0.021 seconds (Sampling)
## Chain 1:                0.044 seconds (Total)
## Chain 1:
```

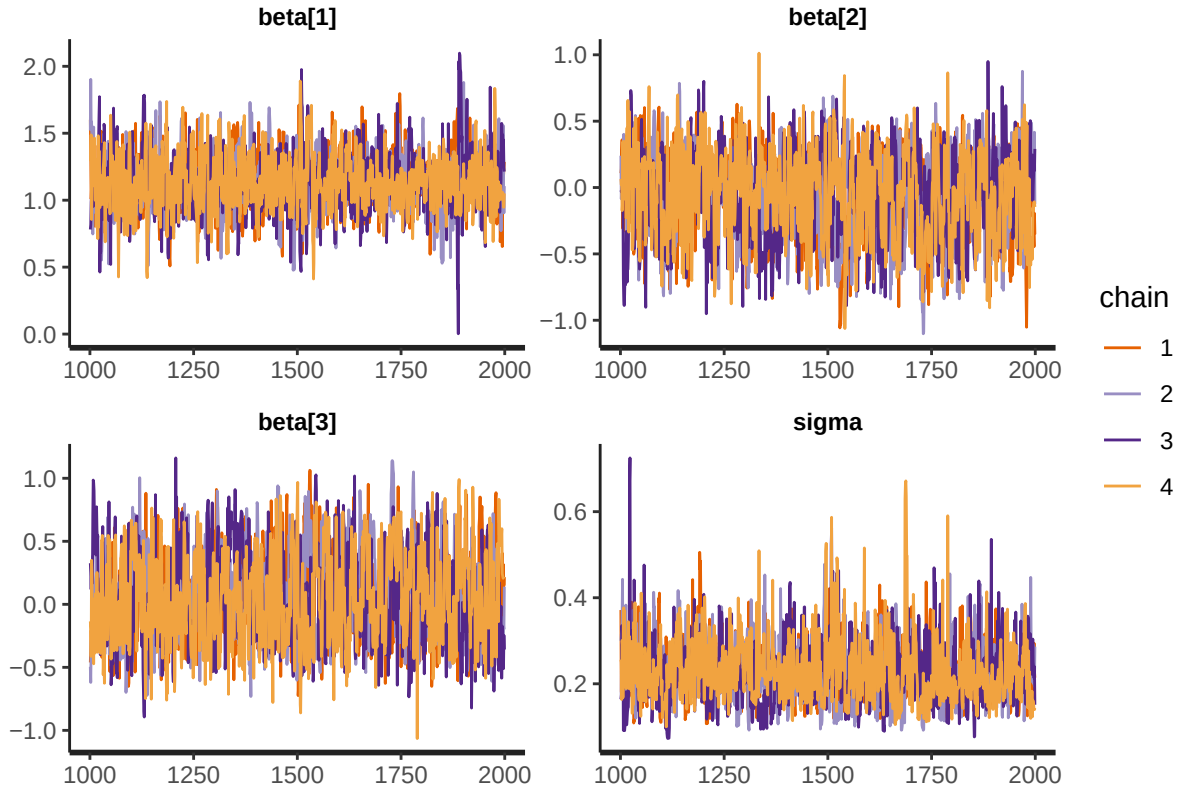
```
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)Chain
## 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.026 seconds (Warm-up)
## Chain 2:                0.03 seconds (Sampling)
## Chain 2:                0.056 seconds (Total)
## Chain 2:
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.024 seconds (Warm-up)
## Chain 3:                0.028 seconds (Sampling)
## Chain 3:                0.052 seconds (Total)
## Chain 3:
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.024 seconds (Warm-up)
## Chain 4:                0.022 seconds (Sampling)
## Chain 4:                0.046 seconds (Total)
## Chain 4:
```

```
pairs(fit3_stan_6, pars = c('beta','sigma','lp__'))
```

```
## Warning in par(usr): argument 1 does not name a graphical parameter
## Warning in par(usr): argument 1 does not name a graphical parameter
## Warning in par(usr): argument 1 does not name a graphical parameter
## Warning in par(usr): argument 1 does not name a graphical parameter
## Warning in par(usr): argument 1 does not name a graphical parameter
```




```
traceplot(fit3_stan_6)
```



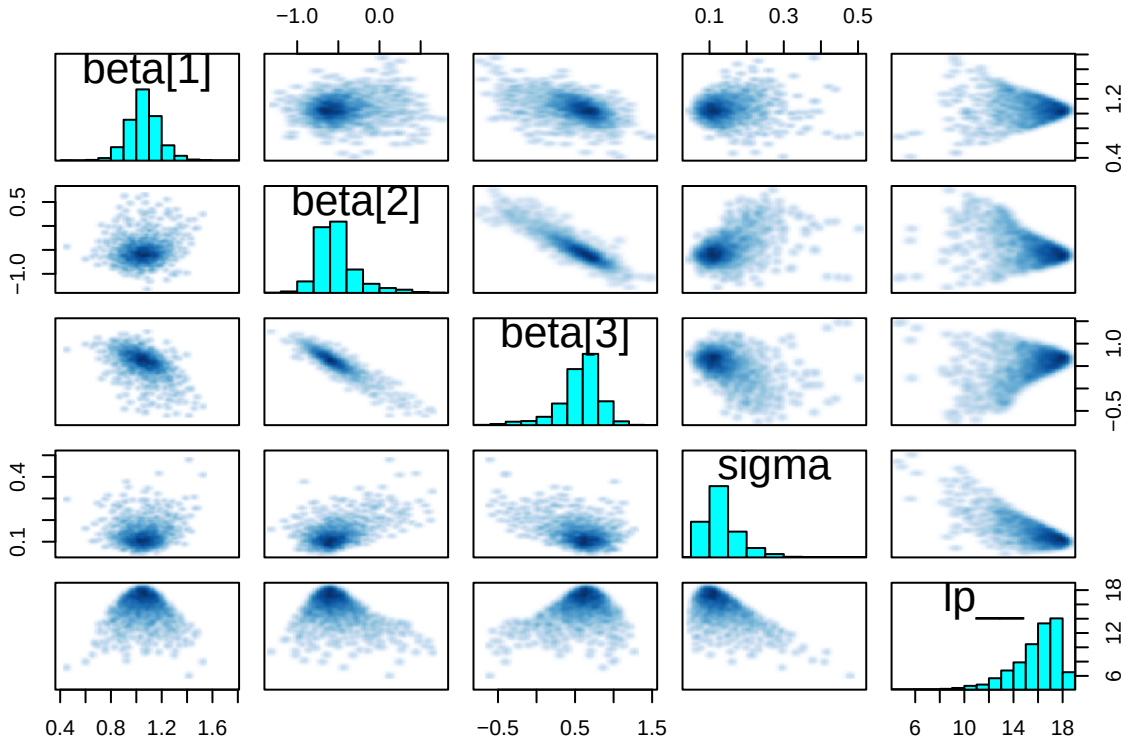

```
if(interactive) Fit3d(fun(fit3_stan_6), col = 'orange')

# Let's try more kurtosis
```

```
fit3_stan_3 <- sampling(robust_model_dso, c(dat3_list, nu = 3))
```

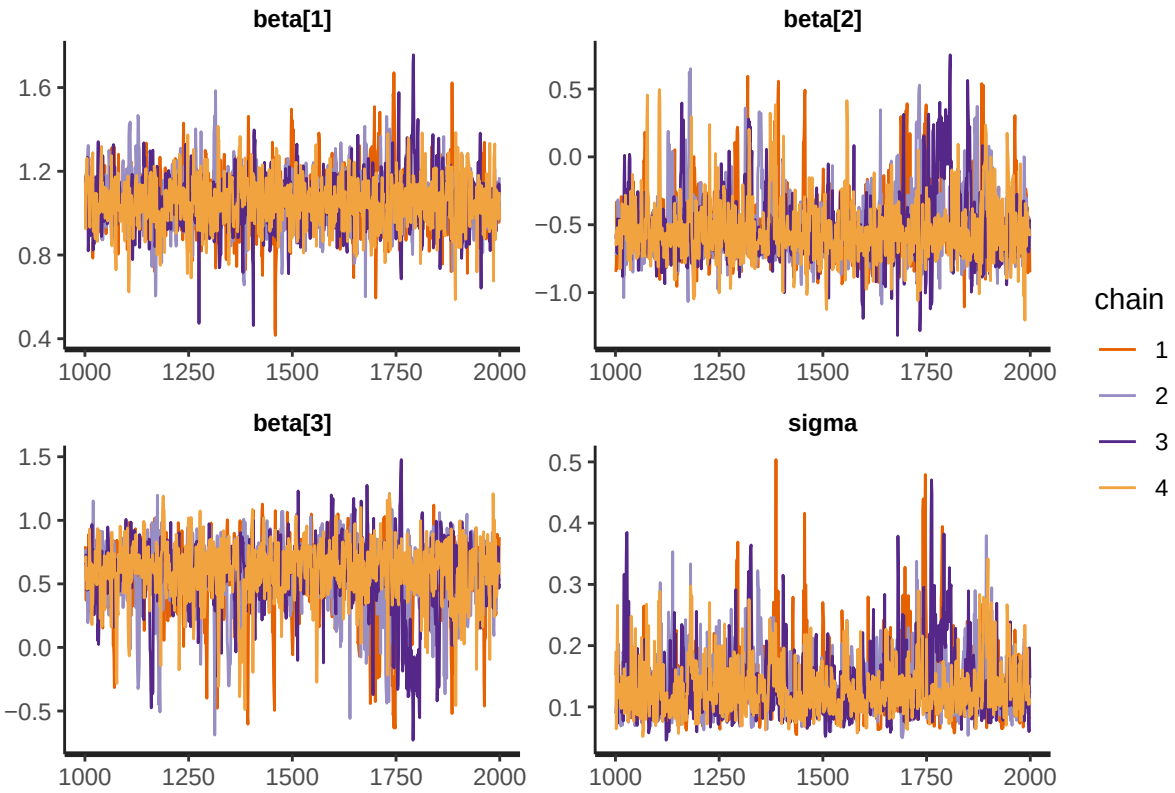
```
## Warning: Tail Effective Samples Size (ESS) is too low, indicating posterior variances and tail quanti
## Running the chains for more iterations may help. See
## https://mc-stan.org/misc/warnings.html#tail-ess
```

```
pairs(fit3_stan_3, pars = c('beta','sigma','lp__'))
```

```
## Warning in par(usr): argument 1 does not name a graphical parameter
## Warning in par(usr): argument 1 does not name a graphical parameter
## Warning in par(usr): argument 1 does not name a graphical parameter
## Warning in par(usr): argument 1 does not name a graphical parameter
## Warning in par(usr): argument 1 does not name a graphical parameter
```




```
traceplot(fit3_stan_3)
```




```
fit3_stan_3
```

```
## hain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.028 seconds (Warm-up)
## Chain 2:                0.03 seconds (Sampling)
## Chain 2:                0.058 seconds (Total)
## Chain 2:
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.029 seconds (Warm-up)
## Chain 3:                0.034 seconds (Sampling)
## Chain 3:                0.063 seconds (Total)
## Chain 3:
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.029 seconds (Warm-up)
## Chain 4:                0.032 seconds (Sampling)
## Chain 4:                0.061 seconds (Total)
## Chain 4:
```

```
if(interactive) Fit3d(fun(fit3_stan_3), col = 'orange')

## Adding fit indices ----------------------------------------

robust_model <-
```

```
  "
data {
  int N;   // number of observations
  int P;   // number of columns of X matrix (including intercept)
  matrix[N,P] X;   // X matrix including intercept
  vector[N] y;  // response
  int nu;   // degrees for freedom for student_t
}
parameters {
  vector[P] beta;   // default uniform prior if nothing specied in model
  real <lower=0> sigma;
}
model {
  y ~ student_t(nu, X * beta, sigma);
}
generated quantities {
  // computes log likelihood at each point to compute WAIC
  vector[N] log_lik;
  for(n in 1:N) {  // index in for loop need not be declared
    log_lik[n] = student_t_lpdf(y[n] | nu, X[n,] * beta , sigma);
  }
}
"
system.time(
  robust_model_dso <- stan_model(model_code = robust_model)
)
```

```
## Trying to compile a simple C file
```

```
fitlist <- list(
fit3_stan_3 = sampling(robust_model_dso, c(dat3_list, nu = 3)),
fit3_stan_6 = sampling(robust_model_dso, c(dat3_list, nu = 6)),
fit3_stan_100 = sampling(robust_model_dso, c(dat3_list, nu = 100))
)
```

```
## : 1001 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.025 seconds (Warm-up)
## Chain 3:                0.033 seconds (Sampling)
## Chain 3:                0.058 seconds (Total)
## Chain 3:
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.029 seconds (Warm-up)
## Chain 4:                0.024 seconds (Sampling)
## Chain 4:                0.053 seconds (Total)
## Chain 4:
## :               0.027 seconds (Sampling)
```

```
## Chain 4:                0.054 seconds (Total)
## Chain 4:
```

```
lapply(fitlist, print, pars = c('beta','sigma'))
```

```
## 58  1175 1.00
## beta[2] -0.09    0.01 0.34 -0.75 -0.34 -0.06 0.16  0.49   548 1.01
## beta[3]  0.09    0.02 0.38 -0.56 -0.20  0.05 0.37  0.80   543 1.01
## sigma    0.22    0.00 0.07  0.10  0.17  0.21 0.26  0.38   712 1.01
##
## Samples were drawn using NUTS(diag_e) at Mon Jun  9 23:44:10 2025.
## 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: anon_model.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##          mean se_mean   sd  2.5%   25%   50%   75% 97.5% n_eff Rhat
## beta[1]  1.19       0 0.23  0.73  1.05  1.19  1.33  1.64  2073    1
## beta[2]  0.21       0 0.18 -0.14  0.10  0.21  0.32  0.57  2145    1
## beta[3] -0.24       0 0.19 -0.62 -0.36 -0.24 -0.13  0.13  2064    1
## sigma    0.27       0 0.06  0.18  0.23  0.27  0.31  0.42  1561    1
##
## Samples were drawn using NUTS(diag_e) at Mon Jun  9 23:44:11 2025.
## 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).
## 0 [ 80%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.025 seconds (Warm-up)
## Chain 3:                0.033 seconds (Sampling)
## Chain 3:                0.058 seconds (Total)
## Chain 3:
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.029 seconds (Warm-up)
## Chain 4:                0.024 seconds (Sampling)
## Chain 4:                0.053 seconds (Total)
## Chain 4:
## :               0.027 seconds (Sampling)
## Chain 4:                0.054 seconds (Total)
## Chain 4:
```

```
library(loo)
```

```
## This is loo version 2.8.0
## - Online documentation and vignettes at mc-stan.org/loo
## - As of v2.0.0 loo defaults to 1 core but we recommend using as many as possible. Use the 'cores' ar
##
## Attaching package: 'loo'
```

```
## The following object is masked from 'package:rstan':
##
##     loo
```

```
loo(extract_log_lik(fitlist[[1]]))
loo(extract_log_lik(fitlist[[2]]))
```

```
## Warning: Some Pareto k diagnostic values are too high. See help('pareto-k-diagnostic') for details.
```

```
## aws (r_eff=1).
##
## Pareto k diagnostic values:
##                          Count Pct.    Min. ESS
## (-Inf, 0.7]   (good)     15    93.8%   1417
##    (0.7, 1]   (bad)       1     6.2%   <NA>
##    (1, Inf)   (very bad)  0     0.0%   <NA>
## See help('pareto-k-diagnostic') for details.
```

```
loo(extract_log_lik(fitlist[[3]]))
```

```
## Warning: Some Pareto k diagnostic values are too high. See help('pareto-k-diagnostic') for details.
```

```
##
## Computed from 4000 by 16 log-likelihood matrix.
##
##          Estimate  SE
## elpd_loo     -4.9 3.5
## p_loo         4.9 2.9
## looic         9.7 7.0
## ------
## MCSE of elpd_loo is NA.
## MCSE and ESS estimates assume independent draws (r_eff=1).
##
## Pareto k diagnostic values:
##                          Count Pct.    Min. ESS
## (-Inf, 0.7]   (good)     15    93.8%   1693
##    (0.7, 1]   (bad)       0     0.0%   <NA>
##    (1, Inf)   (very bad)  1     6.2%   <NA>
## See help('pareto-k-diagnostic') for details.
```

```
## Hierarchical model ----------------------------------------------------------------------
data(package='spida2')

### Generic mixed model in Stan ------------------------------------------------

# with flat priors for hyperparameters


mixed_model <- "
data {
  int N;
  int P;
  int Q;
  int J;   // number of clusters
  matrix[N,P] X;
  matrix[N,Q] Z;
  vector[N] y;
```

```
  int id[N];
}
transformed data {
  vector[Q] zero;
  for(q in 1:Q) zero[q] = 0.0;
}
parameters {
  vector[P] gamma; // fixed effects
  matrix[J,Q] u;  // between cluster random effects -- centered
  cov_matrix[Q] G;  // G matrix: variance of u
  real<lower=0> sigma;   // within-cluster variance
}
model {
  vector[N] eta;
  for(j in 1:J) u[j] ~ multi_normal(zero, G);
  eta = X * gamma + rows_dot_product(Z, u[id,]);
  y ~ normal(eta, sigma);
}
"
mixed_model_dso2 <- stan_model(model_code = mixed_model)
```

```
## Trying to compile a simple C file
```

```
## Running /usr/lib/R/bin/R CMD SHLIB foo.c
## using C compiler: 'gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0'
## gcc -I"/usr/share/R/include" -DNDEBUG   -I"/home/georges/R/x86_64-pc-linux-gnu-library/4.5/Rcpp/inclu
## In file included from /home/georges/R/x86_64-pc-linux-gnu-library/4.5/RcppEigen/include/Eigen/Core:19
##                  from /home/georges/R/x86_64-pc-linux-gnu-library/4.5/RcppEigen/include/Eigen/Dense:1
##                  from /home/georges/R/x86_64-pc-linux-gnu-library/4.5/StanHeaders/include/stan/math/p
##                  from <command-line>:
## /home/georges/R/x86_64-pc-linux-gnu-library/4.5/RcppEigen/include/Eigen/src/Core/util/Macros.h:679:10
##   679 | #include <cmath>
##       |          ^~~~~~
## compilation terminated.
## make: *** [/usr/lib/R/etc/Makeconf:202: foo.o] Error 1
```

```
head(hs)
```

```
##   school mathach    ses    Sex Minority Size   Sector PRACAD DISCLIM
## 1   1317  12.862  0.882 Female       No  455 Catholic   0.95  -1.694
## 2   1317   8.961  0.932 Female      Yes  455 Catholic   0.95  -1.694
## 3   1317   4.756 -0.158 Female      Yes  455 Catholic   0.95  -1.694
## 4   1317  21.405  0.362 Female      Yes  455 Catholic   0.95  -1.694
## 5   1317  20.748  1.372 Female       No  455 Catholic   0.95  -1.694
## 6   1317  18.362  0.132 Female      Yes  455 Catholic   0.95  -1.694
```

```
head( Xmat <- model.matrix(~(ses+cvar(ses,school))*Sector,hs) )
```

```
##   (Intercept)    ses cvar(ses, school) SectorPublic ses:SectorPublic
## 1           1  0.882         0.3453333            0               0
## 2           1  0.932         0.3453333            0               0
## 3           1 -0.158         0.3453333            0               0
## 4           1  0.362         0.3453333            0               0
## 5           1  1.372         0.3453333            0               0
## 6           1  0.132         0.3453333            0               0
##   cvar(ses, school):SectorPublic
```

```
## 1                              0
## 2                              0
## 3                              0
## 4                              0
## 5                              0
## 6                              0
```

```
head( Zmat <- model.matrix(~1 + ses, hs) )
```

```
##   (Intercept)    ses
## 1           1  0.882
## 2           1  0.932
## 3           1 -0.158
## 4           1  0.362
## 5           1  1.372
## 6           1  0.132
```

```
id <- as.numeric(as.factor(hs$school))  # this works for the reason as.character of a factor does not!

dat <- list(N = nrow(Xmat),
            X = Xmat,
            Z = Zmat,
            id = id,
            J = max(id),
            N = nrow(Xmat),
            P = ncol(Xmat),
            Q = ncol(Zmat),
            y = hs$mathach)

system.time(
  mod <- sampling(mixed_model_dso2 , dat)
)  # takes about 120 secs
```

```
## Warning: There were 4 divergent transitions after warmup. See
## https://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
## to find out why this is a problem and how to eliminate them.
```

```
## Warning: Examine the pairs() plot to diagnose sampling problems
```

```
## Warning: Bulk Effective Samples Size (ESS) is too low, indicating posterior means and medians may be
## Running the chains for more iterations may help. See
## https://mc-stan.org/misc/warnings.html#bulk-ess
```

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

```
##    user  system elapsed
##   0.570   0.294   9.382
## ain 4:
## Chain 4: Gradient evaluation took 0.000412 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 4.12 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
```

```
## Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 4.233 seconds (Warm-up)
## Chain 2:                3.056 seconds (Sampling)
## Chain 2:                7.289 seconds (Total)
## Chain 2:
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 4.273 seconds (Warm-up)
## Chain 4:                3.056 seconds (Sampling)
## Chain 4:                7.329 seconds (Total)
## Chain 4:
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 4.898 seconds (Warm-up)
## Chain 1:                2.897 seconds (Sampling)
## Chain 1:                7.795 seconds (Total)
```

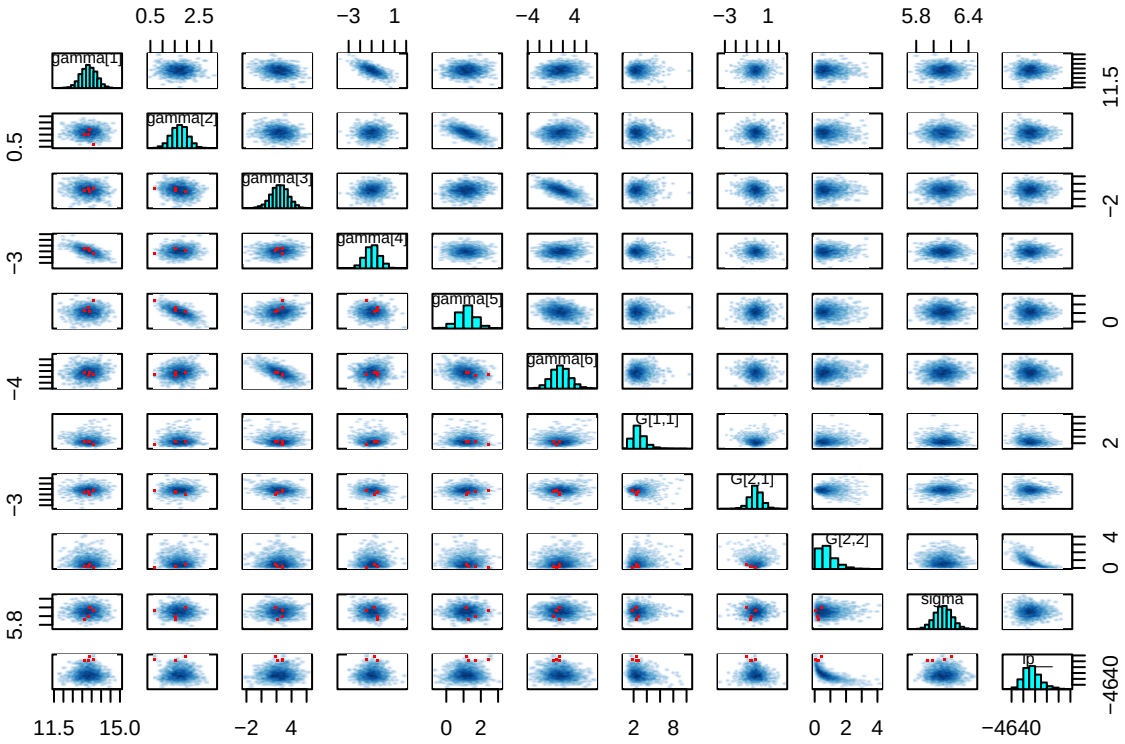
```
## Chain 1:
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 5.318 seconds (Warm-up)
## Chain 3:                3.319 seconds (Sampling)
## Chain 3:                8.637 seconds (Total)
## Chain 3:
```

```
print(mod, pars = c('gamma','G','sigma'))
pairs(mod, pars = c('gamma','G','sigma','lp__'))
```

```
## the following parameters were dropped because they are duplicative
## G[1,2]
```

```
## Warning in par(usr): argument 1 does not name a graphical parameter
```

```
## Warning in par(usr): argument 1 does not name a graphical parameter
## Warning in par(usr): argument 1 does not name a graphical parameter
## Warning in par(usr): argument 1 does not name a graphical parameter
## Warning in par(usr): argument 1 does not name a graphical parameter
## Warning in par(usr): argument 1 does not name a graphical parameter
## Warning in par(usr): argument 1 does not name a graphical parameter
## Warning in par(usr): argument 1 does not name a graphical parameter
## Warning in par(usr): argument 1 does not name a graphical parameter
## Warning in par(usr): argument 1 does not name a graphical parameter
## Warning in par(usr): argument 1 does not name a graphical parameter
```




```
#
# Different parametrization for the G matrix
#
# Note that this does not work well at all without
```

```
# hyperpriors.  See (cholesky parametrization fails.pdf)

mixed_model_2 <- "
data {
  int N;
  int P;
  int Q;
  int J;   // number of clusters
  matrix[N,P] X;
  matrix[N,Q] Z;
  vector[N] y;
  int id[N];
}
transformed data {
  vector[Q] zero;
  for(q in 1:Q) zero[q] = 0.0;
}
parameters {
  vector[P] gamma; // fixed effects
  matrix[J,Q] u;  // between cluster random effects -- centered
  cholesky_factor_corr[Q] Lg;  // correlation matrix: variance of u
  vector<lower=0>[Q]  g; // sdd of random effects
  real<lower=0> sigma;   // within-cluster variance
}
transformed parameters {
  cholesky_factor_cov[Q] L;
  cov_matrix[Q] G;

  L = diag_pre_multiply(g, Lg);
  G = L * L';
}
model {
  vector[N] eta;
  // here we can add a prior on g and sigma
  // sigma ~ cauchy(0,2.5);
  // gamma ~
  Lg ~ lkj_corr_cholesky(2.0);
  for(j in 1:J) u[j] ~ multi_normal_cholesky(zero, L);
  eta = X * gamma + rows_dot_product(Z, u[id,]);
  y ~ normal(eta, sigma);
}
"
mixed_model_2_dso <- stan_model(model_code = mixed_model_2)
```

```
## Trying to compile a simple C file
```

```
system.time(
  mod_2 <- sampling(mixed_model_2_dso , dat)
)  # takes about 180 secs
```

```
## Warning: There were 1 divergent transitions after warmup. See
## https://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
## to find out why this is a problem and how to eliminate them.
```

```
## Warning: There were 4 chains where the estimated Bayesian Fraction of Missing Information was low. S
```

```
## https://mc-stan.org/misc/warnings.html#bfmi-low

## Warning: Examine the pairs() plot to diagnose sampling problems

## Warning: The largest R-hat is NA, indicating chains have not mixed.
## Running the chains for more iterations may help. See
## https://mc-stan.org/misc/warnings.html#r-hat

## Warning: Bulk Effective Samples Size (ESS) is too low, indicating posterior means and medians may be
## Running the chains for more iterations may help. See
## https://mc-stan.org/misc/warnings.html#bulk-ess

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

## elapsed
##   0.526   0.354  14.025
## ain 4:
## Chain 4: Gradient evaluation took 0.000308 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 3.08 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
```

```
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 4.449 seconds (Warm-up)
## Chain 4:                3.189 seconds (Sampling)
## Chain 4:                7.638 seconds (Total)
## Chain 4:
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 4.951 seconds (Warm-up)
## Chain 1:                3.008 seconds (Sampling)
## Chain 1:                7.959 seconds (Total)
## Chain 1:
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 4.439 seconds (Warm-up)
## Chain 2:                3.821 seconds (Sampling)
## Chain 2:                8.26 seconds (Total)
## Chain 2:
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 6.085 seconds (Warm-up)
## Chain 3:                7.197 seconds (Sampling)
## Chain 3:                13.282 seconds (Total)
## Chain 3:
```

```
#
# Viewing predicted model on new values
# assuming that fixed effects are called gamma
#
# with flat priors for hyperparameters
#
mod_2
```

```
## 6    -0.30    -0.02     0.20     0.97
## u[17,1]      0.10    0.02  0.84    -1.60    -0.47     0.13     0.66     1.70
## u[17,2]      0.47    0.06  0.63    -0.33     0.03     0.28     0.76     2.08
## u[18,1]      1.47    0.01  0.78    -0.04     0.97     1.46     1.99     3.04
## u[18,2]      0.05    0.01  0.51    -0.95    -0.21     0.02     0.28     1.20
## u[19,1]      0.63    0.01  0.81    -1.01     0.08     0.64     1.18     2.20
## u[19,2]     -0.16    0.02  0.48    -1.30    -0.38    -0.07     0.10     0.70
## u[20,1]      0.60    0.01  0.77    -0.89     0.08     0.60     1.14     2.13
## u[20,2]      0.08    0.01  0.49    -0.88    -0.15     0.03     0.28     1.23
## u[21,1]     -0.62    0.02  0.85    -2.30    -1.17    -0.62    -0.05     1.04
## u[21,2]      0.23    0.02  0.53    -0.67    -0.06     0.12     0.44     1.56
## u[22,1]     -0.23    0.03  1.17    -2.62    -0.98    -0.23     0.53     2.05
## u[22,2]     -0.21    0.02  0.52    -1.49    -0.43    -0.11     0.08     0.64
```

```
## u[23,1]      1.15    0.02  0.82    -0.47     0.59     1.15     1.70     2.79
## u[23,2]     -0.08    0.01  0.48    -1.20    -0.30    -0.04     0.16     0.87
## u[24,1]      1.14    0.02  0.93    -0.71     0.50     1.12     1.77     3.00
## u[24,2]     -0.28    0.03  0.57    -1.69    -0.52    -0.15     0.05     0.64
## u[25,1]      0.22    0.02  0.92    -1.63    -0.39     0.22     0.83     2.07
## u[25,2]      0.12    0.01  0.47    -0.78    -0.12     0.07     0.32     1.21
## u[26,1]     -2.94    0.02  0.94    -4.88    -3.56    -2.92    -2.32    -1.16
## u[26,2]      0.06    0.01  0.54    -1.08    -0.24     0.04     0.35     1.21
## u[27,1]      0.52    0.02  0.81    -1.01    -0.01     0.52     1.06     2.17
## u[27,2]      0.20    0.02  0.51    -0.68    -0.08     0.11     0.40     1.49
## u[28,1]     -0.31    0.02  0.90    -2.14    -0.90    -0.30     0.32     1.41
## u[28,2]     -0.06    0.01  0.47    -1.10    -0.28    -0.03     0.18     0.91
## u[29,1]     -0.99    0.02  0.82    -2.63    -1.55    -0.97    -0.45     0.62
## u[29,2]      0.28    0.03  0.49    -0.48    -0.02     0.16     0.51     1.47
## u[30,1]      3.31    0.02  0.85     1.69     2.72     3.30     3.87     5.03
## u[30,2]     -0.18    0.02  0.60    -1.59    -0.47    -0.11     0.16     0.93
## u[31,1]      1.01    0.02  0.98    -0.94     0.36     1.00     1.65     2.97
## u[31,2]      0.02    0.01  0.50    -1.08    -0.21     0.01     0.24     1.13
## u[32,1]     -0.54    0.02  0.90    -2.34    -1.14    -0.55     0.07     1.28
## u[32,2]     -0.33    0.04  0.58    -1.87    -0.56    -0.18     0.02     0.47
## u[33,1]     -0.45    0.02  1.00    -2.49    -1.11    -0.43     0.21     1.47
## u[33,2]      0.08    0.01  0.53    -0.98    -0.16     0.04     0.28     1.31
## u[34,1]     -1.14    0.02  0.96    -3.06    -1.78    -1.12    -0.49     0.66
## u[34,2]      0.04    0.01  0.48    -0.97    -0.18     0.03     0.27     1.07
## u[35,1]     -1.64    0.01  0.81    -3.26    -2.17    -1.61    -1.08    -0.09
## u[35,2]     -0.12    0.01  0.49    -1.27    -0.35    -0.06     0.15     0.84
## u[36,1]     -0.44    0.02  0.85    -2.11    -1.01    -0.43     0.13     1.25
## u[36,2]      0.10    0.01  0.47    -0.82    -0.13     0.05     0.31     1.19
## u[37,1]     -2.00    0.02  1.02    -4.09    -2.65    -1.98    -1.29    -0.08
## u[37,2]      0.07    0.01  0.51    -1.02    -0.18     0.05     0.33     1.16
## u[38,1]      1.29    0.02  0.93    -0.52     0.66     1.28     1.89     3.15
## u[38,2]      0.05    0.01  0.49    -0.93    -0.18     0.01     0.27     1.12
## u[39,1]     -1.08    0.02  0.97    -2.99    -1.74    -1.08    -0.41     0.78
## u[39,2]      0.14    0.01  0.51    -0.82    -0.12     0.07     0.36     1.36
## u[40,1]     -0.77    0.02  0.93    -2.62    -1.38    -0.76    -0.15     1.02
## u[40,2]     -0.02    0.01  0.47    -1.07    -0.24    -0.01     0.20     0.96
## Lg[1,1]      1.00     NaN  0.00     1.00     1.00     1.00     1.00     1.00
## Lg[1,2]      0.00     NaN  0.00     0.00     0.00     0.00     0.00     0.00
## Lg[2,1]     -0.08    0.01  0.39    -0.76    -0.38    -0.10     0.19     0.69
## Lg[2,2]      0.91    0.00  0.11     0.59     0.87     0.95     0.99     1.00
## g[1]         1.53    0.01  0.26     1.08     1.34     1.51     1.69     2.08
## g[2]         0.48    0.05  0.30     0.07     0.23     0.43     0.68     1.14
## sigma        6.12    0.00  0.10     5.94     6.06     6.12     6.19     6.31
## L[1,1]       1.53    0.01  0.26     1.08     1.34     1.51     1.69     2.08
## L[1,2]       0.00     NaN  0.00     0.00     0.00     0.00     0.00     0.00
## L[2,1]      -0.04    0.01  0.20    -0.47    -0.15    -0.03     0.07     0.34
## L[2,2]       0.44    0.04  0.29     0.07     0.20     0.38     0.63     1.08
## G[1,1]       2.40    0.02  0.82     1.17     1.80     2.28     2.87     4.33
## G[1,2]      -0.07    0.01  0.32    -0.77    -0.22    -0.04     0.10     0.54
## G[2,1]      -0.07    0.01  0.32    -0.77    -0.22    -0.04     0.10     0.54
## G[2,2]       0.32    0.05  0.37     0.00     0.05     0.18     0.46     1.30
## lp__     -4580.86    4.78 28.65 -4624.53 -4602.89 -4585.68 -4562.41 -4514.98
##          n_eff Rhat
## gamma[1]  1320 1.00
```

```
## gamma[2]  3279 1.00
## gamma[3]  1143 1.00
## gamma[4]  1283 1.00
## gamma[5]  3037 1.00
## gamma[6]  1117 1.00
## u[1,1]    3025 1.00
## u[1,2]    4246 1.00
## u[2,1]    2322 1.00
## u[2,2]    3151 1.01
## u[3,1]    2778 1.00
## u[3,2]    1480 1.00
## u[4,1]    3115 1.00
## u[4,2]     970 1.01
## u[5,1]    2491 1.00
## u[5,2]    3527 1.00
## u[6,1]    2777 1.00
## u[6,2]     297 1.03
## u[7,1]    2118 1.00
## u[7,2]     235 1.03
## u[8,1]    2797 1.00
## u[8,2]    3197 1.00
## u[9,1]    2661 1.00
## u[9,2]    2683 1.00
## u[10,1]   3405 1.00
## u[10,2]   5198 1.00
## u[11,1]   2940 1.00
## u[11,2]    968 1.01
## u[12,1]   3069 1.00
## u[12,2]   1857 1.01
## u[13,1]   1526 1.00
## u[13,2]    134 1.04
## u[14,1]   2735 1.00
## u[14,2]    565 1.01
## u[15,1]   1635 1.00
## u[15,2]   3188 1.00
## u[16,1]   3196 1.00
## u[16,2]   2519 1.00
## u[17,1]   1845 1.00
## u[17,2]    123 1.05
## u[18,1]   2742 1.00
## u[18,2]   2865 1.00
## u[19,1]   3171 1.00
## u[19,2]    944 1.01
## u[20,1]   3340 1.00
## u[20,2]   2179 1.00
## u[21,1]   2591 1.00
## u[21,2]    568 1.02
## u[22,1]   2206 1.00
## u[22,2]    587 1.01
## u[23,1]   1847 1.00
## u[23,2]   3671 1.00
## u[24,1]   3450 1.00
## u[24,2]    277 1.03
## u[25,1]   2527 1.00
```

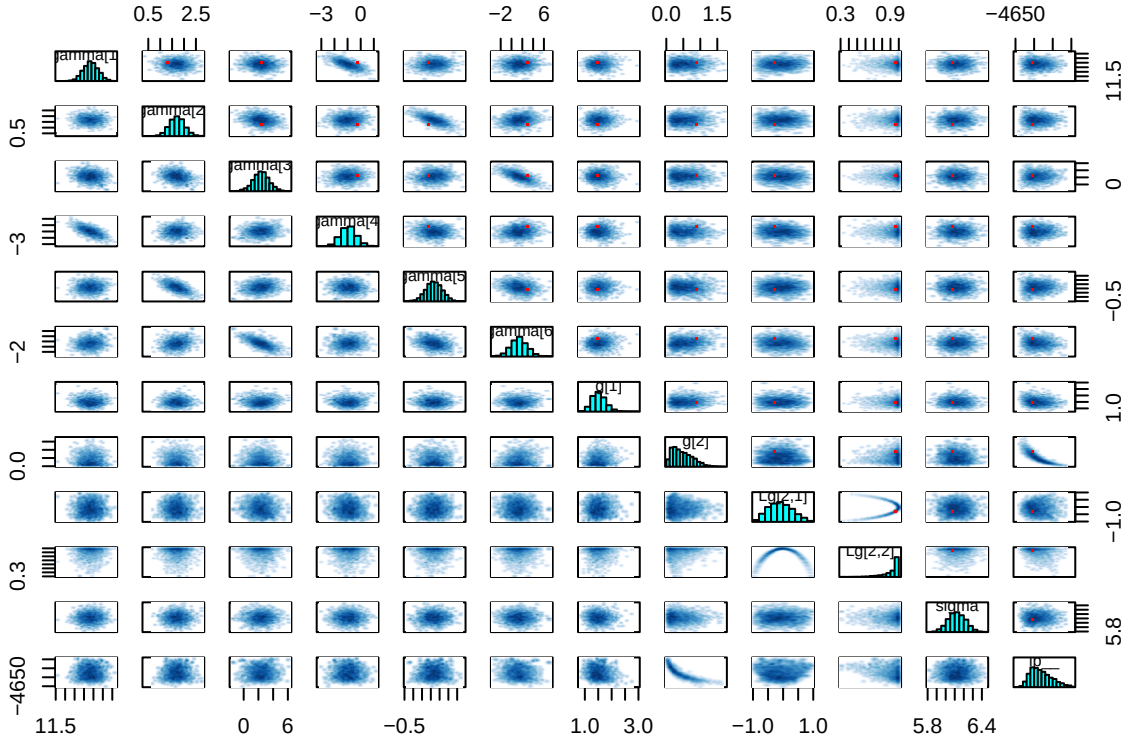
```
## u[25,2]   1992 1.01
## u[26,1]   1898 1.00
## u[26,2]   2050 1.00
## u[27,1]   2702 1.00
## u[27,2]    448 1.02
## u[28,1]   2233 1.00
## u[28,2]   4632 1.00
## u[29,1]   2356 1.00
## u[29,2]    242 1.03
## u[30,1]   2736 1.00
## u[30,2]   1524 1.01
## u[31,1]   3575 1.00
## u[31,2]   4246 1.00
## u[32,1]   2445 1.00
## u[32,2]    254 1.03
## u[33,1]   2153 1.00
## u[33,2]   4075 1.00
## u[34,1]   2299 1.00
## u[34,2]   3449 1.00
## u[35,1]   3159 1.00
## u[35,2]   1126 1.01
## u[36,1]   3103 1.00
## u[36,2]   4065 1.00
## u[37,1]   2914 1.00
## u[37,2]   2939 1.00
## u[38,1]   2363 1.00
## u[38,2]   3665 1.00
## u[39,1]   3683 1.00
## u[39,2]   1611 1.01
## u[40,1]   2180 1.00
## u[40,2]   3840 1.00
## Lg[1,1]    NaN  NaN
## Lg[1,2]    NaN  NaN
## Lg[2,1]    883 1.00
## Lg[2,2]    598 1.01
## g[1]      1736 1.00
## g[2]        43 1.12
## sigma     4990 1.00
## L[1,1]    1736 1.00
## L[1,2]     NaN  NaN
## L[2,1]     889 1.01
## L[2,2]      44 1.12
## G[1,1]    1873 1.00
## G[1,2]     906 1.01
## G[2,1]     906 1.01
## G[2,2]      66 1.09
## lp__        36 1.14
##
## Samples were drawn using NUTS(diag_e) at Mon Jun  9 23:45:50 2025.
## 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).
```

```
pairs(mod_2, pars = c('gamma','g','Lg','sigma','lp__'))
```

```
## the following parameters were dropped because they are constant
## Lg[1,1] Lg[1,2]
```

```
## Warning in par(usr): argument 1 does not name a graphical parameter
```

```
## Warning in par(usr): argument 1 does not name a graphical parameter
## Warning in par(usr): argument 1 does not name a graphical parameter
## Warning in par(usr): argument 1 does not name a graphical parameter
## Warning in par(usr): argument 1 does not name a graphical parameter
## Warning in par(usr): argument 1 does not name a graphical parameter
## Warning in par(usr): argument 1 does not name a graphical parameter
## Warning in par(usr): argument 1 does not name a graphical parameter
## Warning in par(usr): argument 1 does not name a graphical parameter
## Warning in par(usr): argument 1 does not name a graphical parameter
## Warning in par(usr): argument 1 does not name a graphical parameter
## Warning in par(usr): argument 1 does not name a graphical parameter
```




```
#
# Creating the prediction data frame
#
pred.school <- expand.grid(Sector = levels(hs$Sector),
                           ses.mean = c(-1,0,1))
pred.school$school <- 1:nrow(pred.school)
pred <- expand.grid( school = 1:nrow(pred.school),
                     ses.dev = seq(-1.1,1.1,.1))
pred <- merge(pred, pred.school)
pred$ses <- with(pred, ses.mean + ses.dev)
Xpred <- model.matrix(~(ses+cvar(ses,school))*Sector,pred)
head(Xpred)
```

```
##   (Intercept)  ses cvar(ses, school) SectorPublic ses:SectorPublic
## 1           1 -2.1                -1            0                 0
## 2           1 -1.2                -1            0                 0
## 3           1 -1.4                -1            0                 0
## 4           1 -0.5                -1            0                 0
## 5           1 -0.7                -1            0                 0
## 6           1 -1.8                -1            0                 0
##   cvar(ses, school):SectorPublic
## 1                              0
## 2                              0
## 3                              0
## 4                              0
## 5                              0
## 6                              0
```

```
get_posterior_mean(mod)[1:6,5]
```

```
##   gamma[1]   gamma[2]   gamma[3]   gamma[4]   gamma[5]   gamma[6]
## 13.3345719  1.6936880  2.3973526 -0.9394396  1.1694461  1.6023237
```

```
pred$y <- Xpred %*% get_posterior_mean(mod)[1:6,5]

library(lattice)
xyplot(y ~ ses | Sector, pred, groups = school, type = 'l')
```