brms and stan

C..J. Anderson

11/5/2019

In this document includes examples of how to use brms (for normal models) and how to get stan code from brms for use and re-use in stan. We will use the nels data again with 23 school, so we start by running code to get everything set up for model fitting. Among the things illustrated here

* Fitting normal (single level) model
* Fitting multilevel random effects model
* Posterior predictive checks
* waic and loo
* Extracting the stan code and data list produced by brms
* How to compile model using stan code such that it can be re-used
* Sampling from compile model
* A list of the many model families that brms can do

I have included libraries in case you want to also do maximum likelihood estimation of the HLM models.

We will start simple and work up to more complex models.

## Set up

library(brms)

library(rstan)

## Loading required package: StanHeaders

## Loading required package: ggplot2

## rstan (Version 2.19.2, GitRev: 2e1f913d3ca3)

## 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 improved execution time, we recommend calling  
## Sys.setenv(LOCAL\_CPPFLAGS = '-march=native')  
## although this causes Stan to throw an error on a few processors.

library(lme4)

library(lmerTest)

library(mvtnorm)  
library(optimx)

Recommended options for use with stan:

# for stan  
options(mc.cores = parallel::detectCores())  
rstan\_options(auto\_write = TRUE)  
#  
  
setwd("C:/Users/cja/Dropbox/edps 590BAY/Lectures/9 Ham\_Stan\_brms")  
nels <- read.table("school23\_data.txt",header=TRUE)  
head(nels

tail(nels)

# sort by schools:  
nels <- nels[order(nels$school,nels$student) , ]  
  
# Get information on number schools & create consecutive integer school.id  
school <- unique(nels$school) ##   
N <- length(school) ## need N and school.id  
school.int <- as.data.frame(cbind(school,seq(1:N))) ## for dataList & model   
names(school.int) <- c("school", "school.id") ##  
nels <- merge(nels,school.int,by="school") ##  
  
# total number of students  
n <- length(nels$math) ## need n for dataList & model  
  
# You don't need this but might want it  
nj <- matrix(999,nrow=N,ncol=1)  
for (j in 1:N) {  
 nj[j] <- nrow(subset(nels,nels$school.id==j))  
}

## brms to Estimate Mean and Variance

We’ll start simple (fastest of the bunch) and use default priors,

options(mc.cores = parallel::detectCores())  
t0<-Sys.time()  
summary(model.simple <- brm(math~ 1 , data=nels))

## Compiling the C++ model

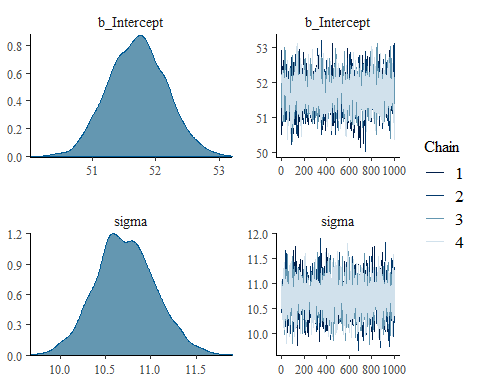
## Start sampling

## Family: gaussian   
## Links: mu = identity; sigma = identity   
## Formula: math ~ 1   
## Data: nels (Number of observations: 519)   
## Samples: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;  
## total post-warmup samples = 4000  
##   
## Population-Level Effects:   
## Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat  
## Intercept 51.71 0.46 50.82 52.62 3091 1.00  
##   
## Family Specific Parameters:   
## Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat  
## sigma 10.72 0.33 10.09 11.39 3503 1.00  
##   
## Samples were drawn using sampling(NUTS). For each parameter, Eff.Sample   
## is a crude measure of effective sample size, and Rhat is the potential   
## scale reduction factor on split chains (at convergence, Rhat = 1).

(tfinal<-Sys.time()-t0)

## Time difference of 1.485655 mins

plot(model.simple)



Note that this take a while and a considerable amount of time is spent getting the model into C++ (i.e., stan).

We can readily get the waic (widely applicable information criterion) for this as well as loo (leave one out):

w0 <- waic(model.simple)  
loo0 <- loo(model.simple)

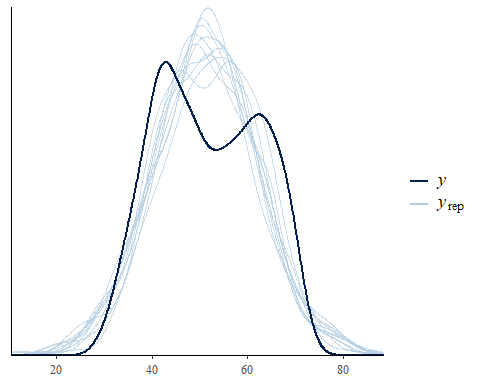
We can also easily obtain posterior prective checks and a quick and dirty way to find out what we can get pp checks on is to type what is below removing comment for “pp\_checks”. Knit doesn’t like to get error messages and gets upset when I try.

# types of pp\_checks available....  
#pp\_check(model.simple,type="dumb")

Here are some examples (they might look better with a more reasonable model)

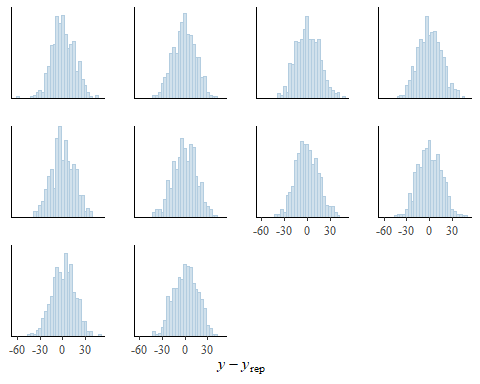
pp\_check(model.simple)

## Using 10 posterior samples for ppc type 'dens\_overlay' by default.

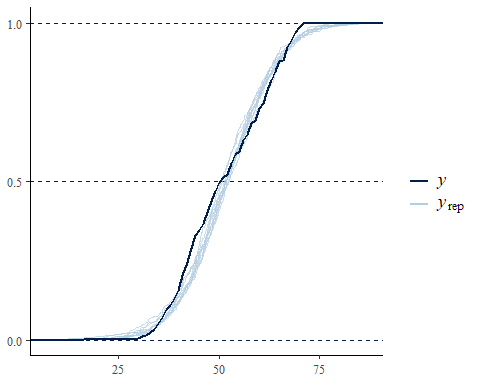


pp\_check(model.simple, type="error\_hist",nsamples=10)

## `stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.



pp\_check(model.simple, type="ecdf\_overlay",nsamples=10)



## brms and stan

Before going on to more interesting models, we’ll take a look at some useful things for \* Simulation studies (faster ones) \* Continue to learn the Stan language

To see the Stan code that brms produced

(simple.stan <-stancode(model.simple))

## // generated with brms 2.9.0  
## functions {  
## }  
## data {  
## int<lower=1> N; // number of observations  
## vector[N] Y; // response variable  
## int prior\_only; // should the likelihood be ignored?  
## }  
## transformed data {  
## }  
## parameters {  
## real temp\_Intercept; // temporary intercept  
## real<lower=0> sigma; // residual SD  
## }  
## transformed parameters {  
## }  
## model {  
## vector[N] mu = temp\_Intercept + rep\_vector(0, N);  
## // priors including all constants  
## target += student\_t\_lpdf(temp\_Intercept | 3, 51, 13);  
## target += student\_t\_lpdf(sigma | 3, 0, 13)  
## - 1 \* student\_t\_lccdf(0 | 3, 0, 13);  
## // likelihood including all constants  
## if (!prior\_only) {  
## target += normal\_lpdf(Y | mu, sigma);  
## }  
## }  
## generated quantities {  
## // actual population-level intercept  
## real b\_Intercept = temp\_Intercept;  
## }

We can also get the data list used with this stan code by using

dataList0 <- standata(model.simple)

We can use the stan code to help learn the stan language and how to optimize model code. Furthermore we can create a compiled version of this code and re-use the model with another data set. This would be a great time saver if you’re running a simulation study or you want to fit the same model to different data sets. For example, I could use if for the anorexia data by just changing the data list.

I made a few changes that I though might speed things up; in particular, I deleted the “if” statement and did away with temporary intercept. You can also change the priors (if you want or you could set them before running brms using the “set\_prior” command).

To compile the model without sampling use the command “stan\_model”,

m.0 <- stan\_model(model\_code = "data {  
 int<lower=1> N; // number of observations  
 vector[N] Y; // response variable  
 int prior\_only; // should the likelihood be ignored?  
 }  
 transformed data {  
 }  
 parameters {  
 real b\_Intercept; // temporary intercept  
 real<lower=0> sigma; // residual SD  
 }  
 transformed parameters {  
 }  
 model {  
 vector[N] mu = b\_Intercept + rep\_vector(0, N);  
 // priors including all constants  
 target += student\_t\_lpdf(b\_Intercept | 3, 51, 13);  
 target += student\_t\_lpdf(sigma | 3, 0, 13)  
 - 1 \* student\_t\_lccdf(0 | 3, 0, 13);  
 // likelihood including all constants  
 target += normal\_lpdf(Y | mu, sigma);  
 }  
 "  
)

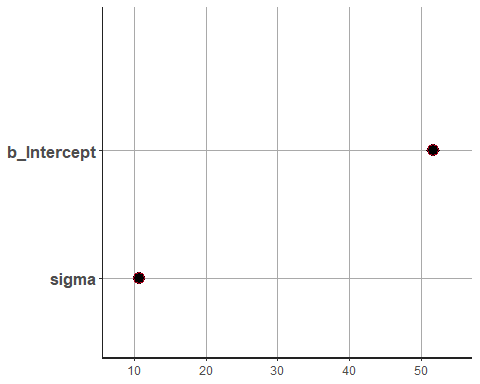
Note that this took a while.

I can now sample from this mode, using the command “sampling” and include all the options that I would normally include when running stan. For the nels data, using mostly the defaults….

# Now do sampling using the nels data list  
model.0 <- sampling(m.0,   
 data=dataList0,  
 chains=4,  
 core=4,  
 )  
stan\_plot(model.0)

## ci\_level: 0.8 (80% intervals)

## outer\_level: 0.95 (95% intervals)



We get what appears to be very precise estimates!

If for some more graphics,

stan\_trace(model.0)  
stan\_dens(model.0)  
stan\_hist(model.0)  
stan\_ac(model.0)  
stan\_scat(fit.2,pars=c("mu","sigma"))

Notice that samping is fast compared to creating the C++ version of our model.

It would be nice to save m.0, so that I can re-use it when ever I would like to; however, I haven’t figured this out yet.

## Random intercept model

Using the default when running lmer, the model failed to converge; however, I change the optimizer and it works. So using MLE,

modelx.lmer <- lmer(math~ 1 + homew + (1 |school.id), data=nels, REML=TRUE,   
 control = lmerControl(optimizer ="Nelder\_Mead"))  
summary(modelx.lmer)

## Linear mixed model fit by REML. t-tests use Satterthwaite's method [  
## lmerModLmerTest]  
## Formula: math ~ 1 + homew + (1 | school.id)  
## Data: nels  
## Control: lmerControl(optimizer = "Nelder\_Mead")  
##   
## REML criterion at convergence: 3729.3  
##   
## Scaled residuals:   
## Min 1Q Median 3Q Max   
## -2.5940 -0.7066 0.0052 0.6613 3.2075   
##   
## Random effects:  
## Groups Name Variance Std.Dev.  
## school.id (Intercept) 21.34 4.620   
## Residual 71.28 8.443   
## Number of obs: 519, groups: school.id, 23  
##   
## Fixed effects:  
## Estimate Std. Error df t value Pr(>|t|)   
## (Intercept) 46.3558 1.1628 33.0122 39.866 <2e-16 \*\*\*  
## homew 2.3999 0.2772 512.8988 8.658 <2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Correlation of Fixed Effects:  
## (Intr)  
## homew -0.437

And using brms, the model formula is the same,

modelx.brm <- brm(math ~ homew + (1 | school.id),  
 data=nels,  
 family=brmsfamily("gaussian"),  
 save\_all\_pars=TRUE,  
 cores=4,   
 silent=FALSE)

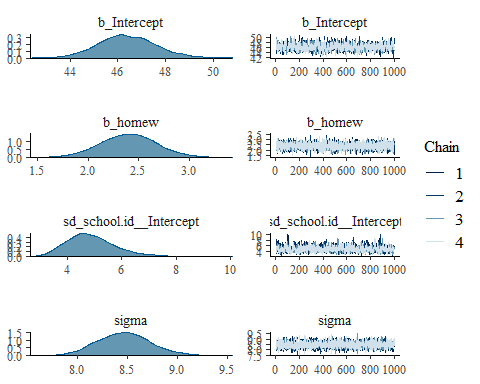
## Compiling the C++ model

## Start sampling

summary(modelx.brm)

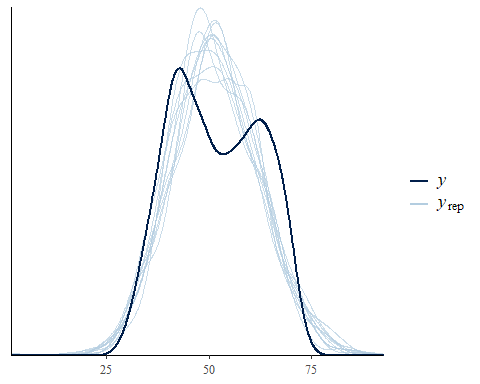
## Family: gaussian   
## Links: mu = identity; sigma = identity   
## Formula: math ~ homew + (1 | school.id)   
## Data: nels (Number of observations: 519)   
## Samples: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;  
## total post-warmup samples = 4000  
##   
## Group-Level Effects:   
## ~school.id (Number of levels: 23)   
## Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat  
## sd(Intercept) 4.88 0.92 3.40 6.99 1143 1.00  
##   
## Population-Level Effects:   
## Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat  
## Intercept 46.40 1.22 44.06 48.88 1102 1.00  
## homew 2.40 0.27 1.87 2.94 5743 1.00  
##   
## Family Specific Parameters:   
## Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat  
## sigma 8.47 0.27 7.96 9.02 6212 1.00  
##   
## Samples were drawn using sampling(NUTS). For each parameter, Eff.Sample   
## is a crude measure of effective sample size, and Rhat is the potential   
## scale reduction factor on split chains (at convergence, Rhat = 1).

plot(modelx.brm)



w1 <- waic(modelx.brm)   
pp\_check(modelx.brm)

## Using 10 posterior samples for ppc type 'dens\_overlay' by default.



## Much more complex model

And lets make this more complex by adding random slope, and some more fixed effects including a cross level interaction,

##########################################################################  
# Random intercept & slope with level 1 and level 2 predictors #  
##########################################################################  
nels$public <- ifelse(nels$schtype==1,1,0)  
nels$public.fac <- as.factor(nels$public)  
  
model2.lmer <- lmer(math~ 1 + homew + ses + public.fac + homew\*public.fac + (1 + homew|school.id),  
 data=nels, REML=TRUE, control = lmerControl(optimizer ="Nelder\_Mead"))  
summary(model2.lmer)

## Linear mixed model fit by REML. t-tests use Satterthwaite's method [  
## lmerModLmerTest]  
## Formula: math ~ 1 + homew + ses + public.fac + homew \* public.fac + (1 +   
## homew | school.id)  
## Data: nels  
## Control: lmerControl(optimizer = "Nelder\_Mead")  
##   
## REML criterion at convergence: 3598.3  
##   
## Scaled residuals:   
## Min 1Q Median 3Q Max   
## -2.3958 -0.6789 -0.0247 0.6389 2.9930   
##   
## Random effects:  
## Groups Name Variance Std.Dev. Corr   
## school.id (Intercept) 53.66 7.325   
## homew 15.49 3.936 -0.87  
## Residual 51.40 7.169   
## Number of obs: 519, groups: school.id, 23  
##   
## Fixed effects:  
## Estimate Std. Error df t value Pr(>|t|)   
## (Intercept) 47.5428 2.8214 21.5157 16.851 7.07e-14 \*\*\*  
## homew 2.3253 1.4741 18.6166 1.577 0.132   
## ses 2.6621 0.5071 496.4469 5.250 2.26e-07 \*\*\*  
## public.fac1 -0.8167 3.4978 21.7755 -0.233 0.818   
## homew:public.fac1 -0.7524 1.8288 18.8471 -0.411 0.685   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Correlation of Fixed Effects:  
## (Intr) homew ses pblc.1  
## homew -0.856   
## ses -0.068 0.000   
## public.fac1 -0.812 0.691 0.135   
## hmw:pblc.f1 0.692 -0.806 -0.026 -0.856

And now for the Bayesian model,

model2.brm <- brm(math~ 1 + homew + ses + public.fac + homew\*public.fac + (1 + homew|school.id),   
 data=nels, cores=4, save\_all\_pars=TRUE) # fit the model

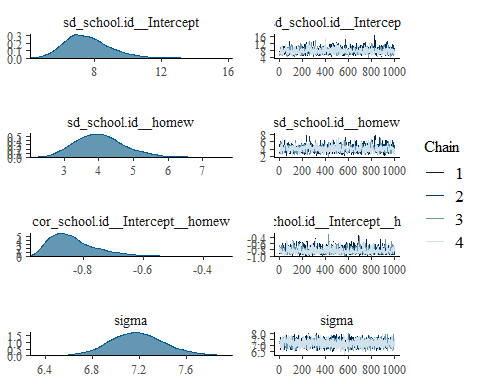
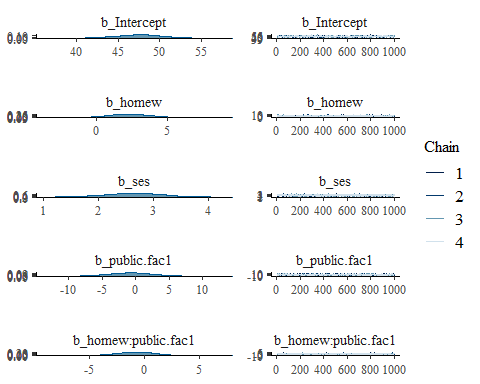
## Compiling the C++ model

## Start sampling

summary(model2.brm) # summary of model information

## Family: gaussian   
## Links: mu = identity; sigma = identity   
## Formula: math ~ 1 + homew + ses + public.fac + homew \* public.fac + (1 + homew | school.id)   
## Data: nels (Number of observations: 519)   
## Samples: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;  
## total post-warmup samples = 4000  
##   
## Group-Level Effects:   
## ~school.id (Number of levels: 23)   
## Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat  
## sd(Intercept) 7.58 1.38 5.33 10.64 1071 1.00  
## sd(homew) 4.06 0.75 2.81 5.68 997 1.00  
## cor(Intercept,homew) -0.83 0.08 -0.94 -0.63 1102 1.00  
##   
## Population-Level Effects:   
## Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat  
## Intercept 47.53 3.00 41.48 53.45 1164 1.00  
## homew 2.31 1.56 -0.84 5.34 1039 1.01  
## ses 2.64 0.52 1.62 3.64 3325 1.00  
## public.fac1 -0.75 3.75 -7.98 6.77 1009 1.00  
## homew:public.fac1 -0.79 1.96 -4.65 3.11 921 1.00  
##   
## Family Specific Parameters:   
## Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat  
## sigma 7.19 0.23 6.74 7.68 3372 1.00  
##   
## Samples were drawn using sampling(NUTS). For each parameter, Eff.Sample   
## is a crude measure of effective sample size, and Rhat is the potential   
## scale reduction factor on split chains (at convergence, Rhat = 1).

plot(model2.brm) # trace and density plots



bayes\_R2(model2.brm) # Bayes version of R2

## Estimate Est.Error Q2.5 Q97.5  
## R2 0.5544468 0.02120486 0.5100196 0.5937179

bayes\_factor(model2.brm,modelx.brm) # Computes bayes factor

## Iteration: 1  
## Iteration: 2  
## Iteration: 3  
## Iteration: 4  
## Iteration: 5  
## Iteration: 6  
## Iteration: 7  
## Iteration: 8  
## Iteration: 1  
## Iteration: 2  
## Iteration: 3  
## Iteration: 4  
## Iteration: 5

## Estimated Bayes factor in favor of bridge1 over bridge2: 140423799162854051618624048.00000

w2 <- waic(model2.brm)  
loo\_compare(w0,w1,w2)

## elpd\_diff se\_diff  
## model2.brm 0.0 0.0   
## modelx.brm -77.3 11.5   
## model.simple -189.1 14.7

## Other model families

There are lots!!

We specify them in brms pretty much how we would do these using glm; however, the range of models is greater with brms than with glm. The model families (from the brms documentation 8/29/2019) include

\* student(link = “identity”, link\_sigma = “log”, link\_nu = “logm1”)

\* bernoulli(link = “logit”) \* negbinomial(link = “log”, link\_shape = “log”)

\* geometric(link = “log”) \* lognormal(link = “identity”, link\_sigma = “log”)

\* shifted\_lognormal(link = “identity”, link\_sigma = “log”,

\* link\_ndt = “log”)skew\_normal(link = “identity”, link\_sigma = “log”,link\_alpha = “identity”)

\* exponential(link = “log”)weibull(link = “log”, link\_shape = “log”)

\* frechet(link = “log”, link\_nu = “logm1”)

\* gen\_extreme\_value(link = “identity”, link\_sigma = “log”,link\_xi = “log1p”)

\* exgaussian(link = “identity”, link\_sigma = “log”, link\_beta = “log”)

\* wiener(link = “identity”, link\_bs = “log”, link\_ndt = “log”,link\_bias = “logit”)

\* Beta(link = “logit”, link\_phi = “log”)

\* dirichlet(link = “logit”, link\_phi = “log”, refcat = NULL)

\* von\_mises(link = “tan\_half”, link\_kappa = “log”)

\* asym\_laplace(link = “identity”, link\_sigma = “log”,link\_quantile = “logit”)

\* hurdle\_poisson(link = “log”)

\* hurdle\_negbinomial(link = “log”, link\_shape = “log”,link\_hu = “logit”)

\* hurdle\_gamma(link = “log”, link\_shape = “log”, link\_hu = “logit”)

\* hurdle\_lognormal(link = “identity”, link\_sigma = “log”,link\_hu = “logit”)

\* zero\_inflated\_beta(link = “logit”, link\_phi = “log”,link\_zi = “logit”)

\* zero\_one\_inflated\_beta(link = “logit”, link\_phi = “log”,link\_zoi = “logit”, link\_coi = “logit”)

\* zero\_inflated\_poisson(link = “log”, link\_zi = “logit”)

\* zero\_inflated\_negbinomial(link = “log”, link\_shape = “log”,link\_zi = “logit”)

\* zero\_inflated\_binomial(link = “logit”, link\_zi = “logit”)

\* categorical(link = “logit”, refcat = NULL)

\* multinomial(link = “logit”, refcat = NULL)

\* cumulative(link = “logit”, link\_disc = “log”,threshold = c(“flexible”, “equidistant”))

\* sratio(link = “logit”, link\_disc = “log”, threshold = c(“flexible”,“equidistant”))

\* cratio(link = “logit”, link\_disc = “log”, threshold = c(“flexible”,“equidistant”))

\* acat(link = “logit”, link\_disc = “log”, threshold = c(“flexible”,“equidistant”))