Matrix normal sampling with by function in brms

Jordan S. Martin

6/29/2021

Data are first simulated with a phylogenetic correlation of -0.5 for group 1 and +0.5 for group 2. Updated brms code is then presented using a matrix normal parameterization for by function random effects, i.e. `(1|G|gr(phylo, cov = A, by = group))`, and the computational time and median estimates of this code are compared to the current brms code for a single random dataset.

```
##############################################################################
#generate Stan code for brms model with Kronecker product using by function
##############################################################################

#simulate scaling matrix
A = rethinking::rlkjcorr(1, 100, 1)

#GROUP A
#simulate correlated phylogenetic effects
r_G = -0.5 #phylo correlation
v_G =  0.5 #phylo variance

G_cor1 <- matrix(c(1,r_G,r_G,1), nrow=2, ncol=2)
G_sd1  <- c(sqrt(v_G),sqrt(v_G))
G1 <- diag(G_sd1) %*% G_cor1 %*% diag(G_sd1)

r_G =  0.5 #phylo correlation
v_G =  0.5 #phylo variance

G_cor2 <- matrix(c(1,r_G,r_G,1), nrow=2, ncol=2)
G_sd2  <- c(sqrt(v_G),sqrt(v_G))
G2 <- diag(G_sd2) %*% G_cor2 %*% diag(G_sd2)

Gblock = as.matrix(Matrix::bdiag(G1,G2))

Kron.prod1 <- Gblock %x% A
P <- matrix(mvtnorm::rmvnorm(1, mean=rep(0,nrow(A)*4), sigma=Kron.prod1), ncol = 4)
P1 = P[c(1:50),1:2];cor(P1)

##            [,1]       [,2]
## [1,]  1.0000000 -0.5762668
## [2,] -0.5762668  1.0000000
P2 = P[c(51:100),3:4];cor(P2)

##           [,1]      [,2]
## [1,] 1.0000000 0.4190752
## [2,] 0.4190752 1.0000000
```

```r
#group id
P = rbind(P1,P2)

#Gaussian responses
v_res = 0.5 #residual variance (assume independent errors)
t1 = 0 + P[,1] + rnorm(nrow(A), 0, sqrt(v_res))
t2 = 0 + P[,2] + rnorm(nrow(A), 0, sqrt(v_res))

library(brms); library(rstan)
rstan_options(auto_write = TRUE)
options(mc.cores = parallel::detectCores())

df = data.frame(t1,t2, phylo = seq(1:nrow(A)),
                group = rep(c(1, 2), each = nrow(A)/2))

rownames(A) = seq(1:nrow(A))

#get stan data list
stan_data = make_standata(formula = bf(t1 ~ 0 + (1|G| gr(phylo, cov = A, by = group))) +
                                    bf(t2 ~ 0 + (1|G| gr(phylo, cov = A, by = group)) ),
                          data = df, data2 = list(A = A))
```

```
#removed function block w/ Kronecker product function
write(
  "functions {
 /* compute correlated group-level effects with 'by' variables
  * in the presence of a within-group covariance matrix
  * Args:
  *   z: matrix of unscaled group-level effects (M_1 x N_1) or (traits x ids)
  *   SD: matrix of standard deviation parameters (M_1 x Nby_1) or (traits x number of by-factor levels
  *   L: an array of cholesky factor correlation matrices (LCs x number of by-factor levels)
  *   Jby: index which grouping level belongs to which by level (index 1 or 2)
  *   Lcov: cholesky factor of within-group correlation matrix (LC of A)
  * Returns:
  *   matrix of scaled group-level effects
  */
  matrix scale_r_cor_by_cov(matrix z, matrix SD, matrix[] L,
                            int[] Jby, matrix Lcov) {
    matrix[rows(L[1]), cols(L[1])] LC[size(L)];
    matrix[cols(z), rows(z)] r_by[size(LC)];
    matrix[cols(z),rows(z)] r;


    for (i in 1:size(LC)) {
    //create VCVs for each group
      LC[i] = diag_pre_multiply(SD[, i], L[i]);
    //create scaled effects for each group VCV
      r_by[i] = Lcov * z' * LC[i]';
    }
    for (j in 1:cols(z)) {
    //select appropriate effects based on group
    r[j] = r_by[Jby[j]][j];
    }
  return r;
  }
}

data {
  int<lower=1> N;  // total number of observations
  int<lower=1> N_t1;  // number of observations
  vector[N_t1] Y_t1;  // response variable
  int<lower=1> N_t2;  // number of observations
  vector[N_t2] Y_t2;  // response variable
  int<lower=1> nresp;  // number of responses
  int nrescor;  // number of residual correlations
  // data for group-level effects of ID 1
  int<lower=1> N_1;  // number of grouping levels
  int<lower=1> M_1;  // number of coefficients per level
  int<lower=1> J_1_t1[N_t1];  // grouping indicator per observation
  int<lower=1> J_1_t2[N_t2];  // grouping indicator per observation
  int<lower=1> Nby_1;  // number of by-factor levels
  int<lower=1> Jby_1[N_1];  // by-factor indicator per observation
  matrix[N_1, N_1] Lcov_1;  // cholesky factor of known covariance matrix
  // group-level predictor values
  vector[N_t1] Z_1_t1_1;
```

```
  vector[N_t2] Z_1_t2_2;
  int<lower=1> NC_1;  // number of group-level correlations
  int prior_only;  // should the likelihood be ignored?
}
transformed data {
  vector[nresp] Y[N];  // response array
  for (n in 1:N) {
    Y[n] = transpose([Y_t1[n], Y_t2[n]]);
  }
}
parameters {
  real<lower=0> sigma_t1;  // dispersion parameter
  real<lower=0> sigma_t2;  // dispersion parameter
  cholesky_factor_corr[nresp] Lrescor;  // parameters for multivariate linear models
  matrix<lower=0>[M_1, Nby_1] sd_1;  // group-level standard deviations
  matrix[M_1, N_1] z_1;  // standardized group-level effects
  cholesky_factor_corr[M_1] L_1[Nby_1];  // cholesky factor of correlation matrix
}
transformed parameters {
  matrix[N_1, M_1] r_1;  // actual group-level effects
  // using vectors speeds up indexing in loops
  vector[N_1] r_1_t1_1;
  vector[N_1] r_1_t2_2;
  // compute actual group-level effects
  r_1 = scale_r_cor_by_cov(z_1, sd_1, L_1, Jby_1, Lcov_1);
  r_1_t1_1 = r_1[, 1];
  r_1_t2_2 = r_1[, 2];
}
model {
  // likelihood including constants
  if (!prior_only) {
    // initialize linear predictor term
    vector[N_t1] mu_t1 = rep_vector(0.0, N_t1);
    // initialize linear predictor term
    vector[N_t2] mu_t2 = rep_vector(0.0, N_t2);
    // multivariate predictor array
    vector[nresp] Mu[N];
    vector[nresp] sigma = transpose([sigma_t1, sigma_t2]);
    // cholesky factor of residual covariance matrix
    matrix[nresp, nresp] LSigma = diag_pre_multiply(sigma, Lrescor);
    for (n in 1:N_t1) {
      // add more terms to the linear predictor
      mu_t1[n] += r_1_t1_1[J_1_t1[n]] * Z_1_t1_1[n];
    }
    for (n in 1:N_t2) {
      // add more terms to the linear predictor
      mu_t2[n] += r_1_t2_2[J_1_t2[n]] * Z_1_t2_2[n];
    }
    // combine univariate parameters
    for (n in 1:N) {
      Mu[n] = transpose([mu_t1[n], mu_t2[n]]);
    }
    target += multi_normal_cholesky_lpdf(Y | Mu, LSigma);
```

```
  }
  // priors including constants
  target += exponential_lpdf(sigma_t1 | 1);
  target += exponential_lpdf(sigma_t2 | 1);
  target += lkj_corr_cholesky_lpdf(Lrescor | 1);
  target += exponential_lpdf(to_vector(sd_1) | 1);
  target += std_normal_lpdf(to_vector(z_1));
  target += lkj_corr_cholesky_lpdf(L_1[1] | 1);
  target += lkj_corr_cholesky_lpdf(L_1[2] | 1);
}
generated quantities {
  // residual correlations
  corr_matrix[nresp] Rescor = multiply_lower_tri_self_transpose(Lrescor);
  vector<lower=-1,upper=1>[nrescor] rescor;
  // compute group-level correlations
  corr_matrix[M_1] Cor_1_1 = multiply_lower_tri_self_transpose(L_1[1]);
  vector<lower=-1,upper=1>[NC_1] cor_1_1;
  // compute group-level correlations
  corr_matrix[M_1] Cor_1_2 = multiply_lower_tri_self_transpose(L_1[2]);
  vector<lower=-1,upper=1>[NC_1] cor_1_2;
  // extract upper diagonal of correlation matrix
  for (k in 1:nresp) {
    for (j in 1:(k - 1)) {
      rescor[choose(k - 1, 2) + j] = Rescor[j, k];
    }
  }
  // extract upper diagonal of correlation matrix
  for (k in 1:M_1) {
    for (j in 1:(k - 1)) {
      cor_1_1[choose(k - 1, 2) + j] = Cor_1_1[j, k];
    }
  }
  // extract upper diagonal of correlation matrix
  for (k in 1:M_1) {
    for (j in 1:(k - 1)) {
      cor_1_2[choose(k - 1, 2) + j] = Cor_1_2[j, k];
    }
  }
}
", "m_by.stan")
```

```
#########################################################################
#compare times and estimated phylogenetic correlations
#########################################################################

#current brms
#########################################################################
start_time <- Sys.time() #time model

current = brm(formula = bf(t1 ~ 0 + (1|G| gr(phylo, cov = A, by = group))) +
                        bf(t2 ~ 0 + (1|G| gr(phylo, cov = A, by = group)) ),
                      data = df, data2 = list(A = A),
                      prior = c(prior("exponential(1)", class = "sd", resp = "t1"),
                                prior("exponential(1)", class = "sigma", resp = "t1"),
                                prior("exponential(1)", class = "sd", resp = "t2"),
                                prior("exponential(1)", class = "sigma", resp = "t2"),
                                prior("lkj(1)", class = "cor")))
Sys.time() - start_time

## Time difference of 2.718511 mins

post1 = posterior_samples(current)
median(post1$`cor_phylo__t1_Intercept:group1__t2_Intercept:group1`)

## [1] -0.671249

median(post1$`cor_phylo__t1_Intercept:group2__t2_Intercept:group2`)

## [1] 0.3909447

#updated code
#########################################################################
start_time = Sys.time()

m2 = stan_model("m_by.stan")
mod2 <- sampling(m2, data= stan_data, init = 0, iter = 2000, warmup = 1000)
Sys.time() - start_time

## Time difference of 1.109381 mins

post2 = extract(mod2)
median(post2$cor_1_1)

## [1] -0.7865274

median(post2$cor_1_2)

## [1] 0.6761183
```