Supplementary Code S1.

Timothée Bonnet

December 21, 2021

Contents

1 Computing environment

To apply this code we need the following packages:

```
library(MCMCglmm) # To fit the animal models

## Loading required package:  Matrix
## Loading required package:  coda
## Loading required package:  ape

library(tidyverse) # To wrangle tables in meta-analysis

## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.4     v dplyr   1.0.7
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   2.0.1     v forcats 0.5.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x tidyr::expand() masks Matrix::expand()
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## x tidyr::pack()   masks Matrix::pack()
## x tidyr::unpack() masks Matrix::unpack()

library(brms) # To fit meta-analytic model

## Loading required package:  Rcpp
## Loading 'brms' package (version 2.16.1).  Useful instructions
## can be found by typing help('brms').  A more detailed introduction
## to the package is available through vignette('brms_overview').
##
## Attaching package:  'brms'
## The following object is masked from 'package:MCMCglmm':
##
##     me
## The following object is masked from 'package:stats':
##
##     ar
```

This document was created in knitr, using the following versions:

```
sessionInfo()
```

```
## R version 4.1.2 (2021-11-01)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 20.04.3 LTS
##
## Matrix products: default
## BLAS:   /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.9.0
## LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.9.0
##
## locale:
##  [1] LC_CTYPE=en_AU.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_AU.UTF-8        LC_COLLATE=en_AU.UTF-8
##  [5] LC_MONETARY=en_AU.UTF-8    LC_MESSAGES=en_AU.UTF-8
##  [7] LC_PAPER=en_AU.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_AU.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] brms_2.16.1     Rcpp_1.0.7      forcats_0.5.1   stringr_1.4.0
##  [5] dplyr_1.0.7     purrr_0.3.4     readr_2.0.1     tidyr_1.1.3
##  [9] tibble_3.1.4    ggplot2_3.3.5   tidyverse_1.3.1 MCMCglmm_2.32
## [13] ape_5.5         coda_0.19-4     Matrix_1.4-0    knitr_1.34
##
## loaded via a namespace (and not attached):
##   [1] cubature_2.0.4.2     minqa_1.2.4          colorspace_2.0-2
##   [4] ellipsis_0.3.2       ggridges_0.5.3       rsconnect_0.8.24
##   [7] estimability_1.3     corpcor_1.6.10       markdown_1.1
##  [10] base64enc_0.1-3      fs_1.5.0             rstudioapi_0.13
##  [13] rstan_2.21.2         farver_2.1.0         DT_0.19
##  [16] fansi_0.5.0          mvtnorm_1.1-2        lubridate_1.7.10
##  [19] xml2_1.3.2           codetools_0.2-18     bridgesampling_1.1-2
##  [22] splines_4.1.2        shinythemes_1.2.0    bayesplot_1.8.1
##  [25] projpred_2.0.2       jsonlite_1.7.2       nloptr_1.2.2.2
##  [28] broom_0.7.9          dbplyr_2.1.1         shiny_1.6.0
##  [31] compiler_4.1.2       httr_1.4.2           emmeans_1.6.3
##  [34] backports_1.2.1      assertthat_0.2.1     fastmap_1.1.0
##  [37] cli_3.0.1            later_1.3.0          prettyunits_1.1.1
##  [40] htmltools_0.5.2      tools_4.1.2          igraph_1.2.6
##  [43] gtable_0.3.0         glue_1.4.2           reshape2_1.4.4
##  [46] posterior_1.1.0      V8_3.4.2             cellranger_1.1.0
##  [49] vctrs_0.3.8          nlme_3.1-152         crosstalk_1.1.1
##  [52] tensorA_0.36.2       xfun_0.26            ps_1.6.0
##  [55] lme4_1.1-27.1        rvest_1.0.1          mime_0.11
##  [58] miniUI_0.1.1.1       lifecycle_1.0.0      gtools_3.9.2
##  [61] MASS_7.3-54          zoo_1.8-9            scales_1.1.1
##  [64] colourpicker_1.1.0   hms_1.1.0            promises_1.2.0.1
##  [67] Brobdingnag_1.2-6    parallel_4.1.2       inline_0.3.19
##  [70] shinystan_2.5.0      curl_4.3.2           gamm4_0.2-6
##  [73] gridExtra_2.3        StanHeaders_2.21.0-7 loo_2.4.1
##  [76] stringi_1.7.4        highr_0.9            dygraphs_1.1.1.6
##  [79] checkmate_2.0.0      pkgbuild_1.2.0       boot_1.3-28
##  [82] rlang_0.4.11         pkgconfig_2.0.3      matrixStats_0.61.0
##  [85] distributional_0.2.2 evaluate_0.14        lattice_0.20-44
##  [88] rstantools_2.1.1     htmlwidgets_1.5.4    processx_3.5.2
##  [91] tidyselect_1.1.1     plyr_1.8.6           magrittr_2.0.1
##  [94] R6_2.5.1             generics_0.1.0       DBI_1.1.1
##  [97] pillar_1.6.2         haven_2.4.3          withr_2.4.2
## [100] mgcv_1.8-36          xts_0.12.1           abind_1.4-5
## [103] modelr_0.1.8         crayon_1.4.1         utf8_1.2.2
## [106] tzdb_0.1.2           grid_4.1.2           readxl_1.3.1
## [109] callr_3.7.0          threejs_0.3.3        reprex_2.0.1
## [112] digest_0.6.27        xtable_1.8-4         httpuv_1.6.3
## [115] RcppParallel_5.1.4   stats4_4.1.2         munsell_0.5.0
## [118] shinyjs_2.0.0
```

2 Animal model fitting

Zero-inflated over-dispersed Poisson animal models for lifetime breeding success can be fitted by the following function:

```
univar_LBS_zi_prior1 <- function(maindir = "ArchiveDataAndCode/ArchivedData",
                                 datadir,
                                 mult=100,
                                 modvar=list(fixed=list("inbreeding", "sexU"),
                                             random = c("id", "dam", "cohort")))
{
  #load main data
  fit <- read.csv(file =  paste0(maindir, "/", datadir, "/data_", datadir, ".csv"),
                  stringsAsFactors = FALSE)
  #load the inverse relatedness matrix
  load(paste0(maindir, "/", datadir, "/Ainv_", datadir))

  #Making sure there are not missing values:
  for (i in 1:length(modvar$fixed))
  {
    fit <- fit[!is.na(fit[,modvar$fixed[[i]] ] ), ]
  }

  #turning the list of fixed and random effects into fomula:
  Fform <- "LBS ~ -1 + trait + at.level(trait,1):stcohort "
  for(i in 1:length(modvar$fixed))
  {
    Fform <- paste0(Fform, "+ trait:", modvar$fixed[[i]])
  }
  Fform <- as.formula(Fform)

  Rform <- "~ "
  for(i in 1:length(modvar$random))
  {
    Rform <- paste0(Rform, ifelse(i==1," ","+"), " us(trait):", modvar$random[[i]])
  }
  Rform <- as.formula(Rform)

  nbfixef <- 1 + 2 + 2*length(modvar$fixed)
  # The number of fixed effects is not twice the length of modvar£fixed,
  # because we need to count 2 parameters for the intercepts,
  # and 1 for the effect of cohort fitted only to the Poisson part.
  prior_a_c <- priortype(dimvar = 2, # effective number of traits
                         #(2 because 1 Poisson and 1 Bernoulli traits)
           extranu = 0, # positive values make the prior more
           # informative for random effects
           nre = length(modvar$random), #number of random effects
           rfix = 2, # fix the residual variance for the Bernoulli trait to 1
           nbfixef=nbfixef, # number of fixed effects
           varfixef=10, # variance for fixed effect prior
           mufixef=0) # mean for fixed effect prior

  m0 <- MCMCglmm(fixed =  Fform, #fixed effect formula
                 random = Rform, #random effect formula
                 rcov=~idh(trait):units, #residual formula
                 prior=prior_a_c, #priors
                 family="zipoisson",#GLMM family,
                 #here zero-inflated over-dispersed Poisson
                 data = fit[!is.na(fit$LBS),],#phenotypic data
                 ginverse = list(id=ainv),#inverse of the additive
                 #genetic relatedness matrix, derived from pedigree
                 thin = 10*mult,#thining of MCMC, to save RAM
                 nitt = 13000*mult, #number of iterations
                 burnin = 3000*mult)#burnin, discarding the first MCMC samples

  #rename and save the model
  filename <- modname <- name_model(datadir=datadir,
                                    mod="univar_LBS_ZIfull_priorB_",
                                    mult=mult)
  assign(x = modname, value = m0)
  save(file = filename, list = modname)
}#end univar_LBS_zi_prior1
```

Where the function ‘priortype()‘ is:

```
priortype <- function(dimvar = 1,
                      extranu=0,
                      nre=3,
                      rfix=2,
                      nbfixef=NULL,
                      varfixef=10,
                      mufixef=0)
{
  nu <- ifelse(dimvar==1,
               0.002+extranu,
               dimvar+extranu)

  Gelement <- list(V=diag(dimvar),
                   nu=nu,
                   alpha.mu=rep(0,times=dimvar),
                   alpha.V=diag(dimvar)*1000)

  G <- list()

  for (i in 1:nre)
  {
    G[[paste0("G",i)]] <- Gelement
  }
  if(is.null(rfix))
  {
    Rcompo = list(V=diag(dimvar), nu=nu)
  }else{
    Rcompo = list(V=diag(dimvar), nu=nu, fix=rfix)
  }

  if(is.null(nbfixef))
   {prior <- list(G=G,
       R=Rcompo)
  }else{
    B <- list(mu=rep(mufixef, nbfixef), V=diag(nbfixef)*varfixef)
    prior <- list(B=B, G=G, R=Rcompo)
  }
  return(prior)
}
```

And the function ‘name_model()‘ provides a time stamp to save the model.

```
name_model <- function(datadir, mod, mult)
{
  tmst <- paste0(strsplit(x = date(),
                          split = c(" "))[[1]][c(4,3,2,5)],
                 collapse = "-")#time stamp
  mn <- paste0(mod,datadir, "_m",mult,"_",tmst)
  return(mn)
}
```

For instance, we can run a model fitted to the snow vole data set and run for only 13000 iterations:

```
univar_LBS_zi_prior1(datadir = "svG",
                         mult=1,
                         modvar=list(fixed=list("inbreeding", "sexU"),
                                     random = c("id", "dam", "cohort")))
```

To reproduce the paper results, ‘mult‘ needs to be changed to 100, but the model will then take a long time to run (from a few hours to a few days depending on population and on computer).

3 Back-transformation

We define utility functions to work with zero-inflated over-dispersed Poisson models:

```
## function for expectation of a zero-inflated Poisson model
Ey<-function(l1,l2){
  plogis(-l2)*exp(l1)
}

## population mean given MC samples of latent variables
barW<-function(MC){
  l1<-MC[,1]
  l2<-MC[,2]
  return(mean(Ey(l1,l2)))
}
```

Given one posterior sample from a zero-inflated over-dispersed animal model, we calculate the value of $V_A(w)$.

```
# function to give the additive genetic variance of relative
# fitness given ZIP model M for MCMC sample s.  This
# is itself done by MC integration, with nMC samples
Va_w_zip<-function(m, s=1, nMC=50000, h=0.02,
                  fixedpred=c("inbreeding", "Qgg"),
                   ranpred = c("id", "dam", "cohort"),
                   focalvar = "id")
{
  require(mvtnorm)

  Xmat <- as.matrix(m$X)
  muP <- Xmat[1:(nrow(Xmat)/2),grep("traitLBS", colnames(Xmat))] %*%
    m$Sol[s,grep("traitLBS", colnames(m$Sol))]
  muZI <- Xmat[(1+(nrow(Xmat)/2)):nrow(Xmat),grep("traitzi_LBS", colnames(Xmat))] %*%
    m$Sol[s,grep("traitzi_LBS", colnames(m$Sol))]

  mu<-cbind(muP, muZI)

  PVP <- m$VCV[s,"traitLBS.units"] +   var(mu[,1])

  for (i in 1:length(ranpred))
  {
    PVP <- PVP + m$VCV[s,paste0("traitLBS:traitLBS.", ranpred[i])]
  }

  PVZI <- m$VCV[s,"traitzi_LBS.units"] +
    var(mu[,2])
  for (i in 1:length(ranpred))
  {
    PVZI <- PVZI + m$VCV[s,paste0("traitzi_LBS:traitzi_LBS.", ranpred[i])]
  }

  PcovZIP <- cov(mu[,1], mu[,2])
  for (i in 1:length(ranpred))
  {
    PcovZIP <- PcovZIP + m$VCV[s,paste0("traitzi_LBS:traitLBS.", ranpred[i])]
  }

  sigma <-matrix(c(PVP,
                   PcovZIP,PcovZIP,
                   PVZI),2,2)

  MC_latent_smaples<-rmvnorm(nMC,colMeans(mu),sigma)

  # mean fitness
  muW<-barW(MC_latent_smaples)

  # values of mean fitness with perturbations of mean latent
  # values for use in getting derivatives by finite differences

  muW1<-barW(MC_latent_smaples+cbind(rep(h,nMC),0))
  muW2<-barW(MC_latent_smaples+cbind(0,rep(h,nMC)))

  # derivatives of mean relative fitness w.r.t. mean latent values
  d1<-(muW1/muW-1)/h
  d2<-(muW2/muW-1)/h

  # genetic variance in relative fitness from the linear
  # projection of the distribution of latent values for the
  #two components of the zip model

  sigma_a<-matrix(m$VCV[s,c(paste0(c("traitLBS:traitLBS.",
                                     "traitzi_LBS:traitLBS.",
                                     "traitLBS:traitzi_LBS.",
                                     "traitzi_LBS:traitzi_LBS."), focalvar))],2,2)

  d<-matrix(c(d1,d2),2,1)

  return(as.numeric(t(d)%*%sigma_a%*%d))
} #end Va_w_zip()
```

We now apply the previous function to all posterior samples of the model to integrate the calculation of $V_A(w)$:

```
Va_w_zip_posterior <- function(m, nMC=50000, h=0.02,
                               fixedpred=c("inbreeding", "Qgg"),
                               ranpred = c("id", "dam", "cohort"), va="id")
{
  length_posterior <- length(m$Sol[,1])
  ranpredunits <- c(ranpred, "units")
  # create a MCMC matrix for the transformed ZIP values
  btzip <- as.mcmc(matrix(data = NA, nrow = length_posterior,
                          ncol = length(ranpredunits),
                          dimnames = list(1:length_posterior, ranpredunits) ) )
  #copy MCMC attributes to the new matrix
  attr(btzip, "mcpar") <- attr(m$VCV, "mcpar")

  btva <- vector(length = length_posterior)
  for(i in 1:length_posterior)
  {
    btva[i] <- Va_w_zip(m = m, s = i,
                        fixedpred=fixedpred, nMC = nMC, h = h,
                        ranpred = ranpred, focalvar = va)
  }
  btzip[, va] <- btva

  m$btva <- btva
  return(m)
}#end Va_w_zip_posterior()
```

For random effects other than the additive genetic variance, we need a different set of functions to account for non-additive effects (which are by definition excluded from the calculation of the additive genetic variance):

```
#### Non genetic variance components ####
# function to give the genetic variance of relative
# fitness given ZIP model M for MCMC sample s.  This
# is itself done by MC integration, with nMC samples
Vp_w_zip<-function(m, s=1, nMC1=500, nMC2=1000,
                   fixedpred=c("inbreeding", "Qgg"),
                   ranpred = c("id","dam", "cohort"),
                   focalvar = "dam"
)
{
  require(mvtnorm)
```

```
Xmat <- as.matrix(m$X)
muP <- Xmat[1:(nrow(Xmat)/2),grep("traitLBS", colnames(Xmat))] %*%
  m$Sol[s,grep("traitLBS", colnames(m$Sol))]
muZI <- Xmat[(1+(nrow(Xmat)/2)):nrow(Xmat),grep("traitzi_LBS", colnames(Xmat))] %*%
  m$Sol[s,grep("traitzi_LBS", colnames(m$Sol))]

mu<-cbind(muP, muZI)

PVP <- m$VCV[s,"traitLBS.units"] +  var(mu[,1])
for (i in 1:length(ranpred))
{
  PVP <- PVP + m$VCV[s,paste0("traitLBS:traitLBS.", ranpred[i])]
}

PVZI <- m$VCV[s,"traitzi_LBS.units"] +  var(mu[,2])
for (i in 1:length(ranpred))
{
  PVZI <- PVZI + m$VCV[s,paste0("traitzi_LBS:traitzi_LBS.", ranpred[i])]
}

PcovZIP <- cov(mu[,1], mu[,2])
for (i in 1:length(ranpred))
{
  PcovZIP <- PcovZIP + m$VCV[s,paste0("traitzi_LBS:traitLBS.", ranpred[i])]
}


sigma <-matrix(c(PVP,
                 PcovZIP,PcovZIP,
                 PVZI),2,2)

MC_latent_smaples<-rmvnorm(nMC1,colMeans(mu),sigma)

# mean fitness
muW<-barW(MC_latent_smaples)

# genetic variance in relative fitness from the linear
# projection of the distribution of latent values for the
#two components of the zip model
if(focalvar=="units")
{
  focalsigma <- matrix(data = c(m$VCV[s,paste0("traitLBS.",focalvar)],
                           0,0,
                           m$VCV[s,paste0("traitzi_LBS.",focalvar)]),
                  2,2)

}else{
  focalsigma <- matrix(m$VCV[s,c(paste0(c("traitLBS:traitLBS.",
                                     "traitzi_LBS:traitLBS.",
                                     "traitLBS:traitzi_LBS.",
                                     "traitzi_LBS:traitzi_LBS."), focalvar))],2,2)
}

focsamp <- rmvnorm(nMC1, colMeans(mu), focalsigma)
compsigma <- sigma - focalsigma

focexpectations <- vector(length = nMC1)
for (i in 1:nrow(focsamp))
```

```
  {
    margvar <- rmvnorm(nMC2, c(focsamp[i,1], focsamp[i,2]), compsigma)
    focexpectations[i] <- mean(apply(margvar, MARGIN = 1,
                                     FUN = function(x) Ey(x[1], x[2])))
  }

  return(var(focexpectations)/(muW^2))
} #end Vp_w_zip()


Vp_w_zip_posterior <- function(m, nMC1=500, nMC2=1000,
                               fixedpred=c("inbreeding", "Qgg"),
                               ranpred = c("id", "dam", "cohort"), va="id")
{
  length_posterior <- length(m$Sol[,1])
  if("units" %in% ranpred) {stop("units should not appear in ranpred.")}
  ranpredunits <- c(ranpred, "units")
  ranpredunits <- ranpredunits[ranpredunits!=va]

  # create a MCMC matrix for the transformed ZIP values
  btzip <- as.mcmc(matrix(data = NA, nrow = length_posterior,
                          ncol = length(ranpredunits),
                          dimnames = list(1:length_posterior, ranpredunits) ) )
  #copy MCMC attributes to the new matrix
  attr(btzip, "mcpar") <- attr(m$VCV, "mcpar")

  #compute the back-transformed effects for each variance component
  for(vp in 1:length(ranpredunits))
  {
    btvp <- vector(length = length_posterior)
    for(i in 1:length_posterior)
    {
      btvp[i] <- Vp_w_zip(m = m, s = i,
                          fixedpred=fixedpred, nMC1 = nMC1, nMC2=nMC2,
                          ranpred = ranpred, #not allowed to contain units
                          focalvar = ranpredunits[vp])
    }
    btzip[, ranpredunits[vp]] <- btvp
  }

  m$btzip <- btzip
  return(m)
}#end Vp_w_zip_posterior()
```

Finally, we define a function that calculates $V_A(w)$ as well as the variance components for all the other random effects, and appending the results as a new slot in the model.

```
Vallcomp_w_zip_posterior <- function(m,
                                     #the model
                                     nMC1=1000,
    # number of first level MCMC sample for other random effect numerical integration
                                     nMC2=1000,
    # number of second level MCMC sample for other random effect numerical integration
                                     nMC=50000,
    # number of MCMC sample for VA(w) numerical integration
                                     h=0.02,
    #size of step for numerical integration
                               fixedpred=c("inbreeding", "Qgg"),
    #fixed effects
                               ranva = "id",
    #name of the additive genetic variance random effect
                               ranpred = c("id", "dam", "cohort")
    #all random effects
      )
{
  length_posterior <- length(m$Sol[,1])

  if("units" %in% c(ranva, ranpred)) {
    stop("units should not appear in ranpred or ranva.")}
  ranpredunits <- c(ranpred, "units")

  btzip <- as.mcmc(matrix(data = NA, nrow = length_posterior,
                          ncol = length(ranpredunits),
                          dimnames = list(1:length_posterior, ranpredunits ) ) )
  #copy MCMC attributes to the new matrix
  attr(btzip, "mcpar") <- attr(m$VCV, "mcpar")

  #compute the back-transformed effects for each variance component
  btva <- vector(length = length_posterior)
  for(i in 1:length_posterior)
  {
    btva[i] <- Va_w_zip(m = m, s = i,
                        fixedpred=fixedpred, nMC = nMC, h = h,
                        ranpred = ranpred, focalvar = ranva)
  }
  btzip[,ranva] <- btva

  for(vp in 2:length(ranpredunits))
  {
    btvp <- vector(length = length_posterior)
    for(i in 1:length_posterior)
    {
      btvp[i] <- Vp_w_zip(m = m, s = i,
                          fixedpred=fixedpred, nMC1 = nMC1, nMC2=nMC2,
                          ranpred = ranpred, focalvar = ranpredunits[vp])
    }
    btzip[, ranpredunits[vp]] <- btvp
  }
  m$btzip <- btzip
  return(m)
}#end Vallcomp_w_zip_posterior()
```

4 Extracting $V_A(w)$

After running and backtransforming several replicated models for each populations and saving them to a ”Models/” directory, we can load them and merge them:

Loading all models:

```
modlf <- list.files("Models/", full.names = TRUE)

zipmodlistall <- list()
for (i in 1:length(modlf))
{
  mnam <- load(modlf[i])
  m0 <- get(x = as.character(mnam))
  zipmodlistall[[i]] <- m0[c("btzip", "Sol", "VCV")]
  rm(list=as.character(mnam))
}
```

Combining the posterior of model replicates from the same population:

```
startpop <- unlist(gregexpr("priorB_", modlf[[1]])) + nchar("priorB_")
allmodelpops <- as.character(sapply(modlf, function(x)
  {
  unlist(strsplit(x=substr(x, startpop, stop = startpop+2), split = "_"))[1]
  }))

upops <- unique(allmodelpops)

zipmodlist <- list()

for (i in 1:length(upops))
{
  which_models <- which(allmodelpops==upops[i])
  zipmodlist[[ upops[i] ]] <- zipmodlistall[[ which_models[1] ]]
  if(length(which_models)>1)
  {
    for (j in 1:length(which_models))
      {
        zipmodlist[[ upops[i] ]]$Sol <-
          rbind(zipmodlist[[ upops[i] ]]$Sol,
                zipmodlistall[[ which_models[j] ]]$Sol)
        zipmodlist[[ upops[i] ]]$VCV <-
          rbind(zipmodlist[[ upops[i] ]]$VCV,
                zipmodlistall[[ which_models[j] ]]$VCV)
        zipmodlist[[ upops[i] ]]$btzip <-
          rbind(zipmodlist[[ upops[i] ]]$btzip,
                zipmodlistall[[ which_models[j] ]]$btzip)
      }
  }
}#end for upops
```

We can extract all back-transformed variance components in each population as a list with:

```
alldistzip <- lapply(zipmodlist,
                     function(x){

                        as.list(as.data.frame(x$btzip))
                            }
                     )
```

From there, we can extract $V_A(w)$ in each population with:

```
VAsamp <- lapply(alldistzip, function(x) {x$id})
```

5 Meta-analytic estimates

To meta-analyse the values of a given parameter across populations we use the following function:

```r
MetaModel <- function(vals,
            # list of population specific posterior distributions for a parameter
                      nb_samples=4000,
                      #how many posterior samples in each population?
                      allpops)
  # labels for the populations
{
  require(tidyverse)
  require(brms)

  mcmc_meta <- data.frame(ID=rep(allpops, each=nb_samples),
                        value=unlist(lapply(vals, function(x) x[1:nb_samples])))

dfs_meta <-
    mcmc_meta %>%
    unnest_legacy() %>%
    group_by(ID) %>%
    sample_n(100) %>%
    mutate(Sample = 1:n()) %>%
    ungroup() %>%
    group_by(Sample) %>%
    nest_legacy(.key = "Df") %>%
    pluck("Df")

prior_n <- c(
    #prior(normal(0, 20), "b"),
    prior(normal(0, 20), "b", lb = 0),
    prior(cauchy(0, 1), "sd")
)

# Formula (for whichever "Parameter" you are interested in)
form <- brmsformula(value ~ 1 + (1|ID),
                    sparse = TRUE)

# Call to brm_multiple to fit the multiple imputation data
# Again, change parameters to your will
mod <-
    brm_multiple(formula    = form,
                 data       = dfs_meta,
                 save_ranef = FALSE,
                 chains     = 3,
                 iter       = 3000,
                 warmup     = 2000,
                 thin       = 50,
                 prior      = prior_n)
return(mod)
}#end MetaModel
```

We can run the meta-model with:

```r
MetaModelVA <- MetaModel(vals = VAsamp,
                        nb_samples=min(unlist(lapply(VAsamp, length))),
                        allpops = names(VAsamp))
```

The meta-average is the parameter ”Population-Level Effects: Intercept”, the standard deviation among populations is ”Group-Level Effects: sd(Intercept)”.

Increase the number of iterations if the model does not converge.