Fitting mlVAR Models with Non-ignorable Missingness in Stan and JAGS

Yanling Li, Julie Wood, Linying Ji, Sy-Miin Chow, Zita Oravecz

The Pennsylvania State University

Fitting mlVAR Models with Non-ignorable Missingness in Stan and JAGS

This tutorial provides illustrations and code examples for handling non-ignorable (or not missing at random (NMAR)) missingness via the Bayesian selection model approach when fitting mlVAR models in Stan and JAGS. Note that although this tutorial and the tutorial in our paper focus on different missing mechanisms (i.e., NMAR and MAR, respectively), most code are identical, especially code in Step 0, which only involves some data preparation work. We suggest that readers first read the code examples in our paper and then code examples in this file to better understand the differences between fitting mlVAR models under MAR and NMAR missing mechanisms.

Since NMAR refers to the scenarios where the missingness depends on information that is unobserved in the data, we specify the missing data mechanism below, using the notations in our paper.

$$\begin{aligned} p_{1,i,t} &= P(R_{1,i,t} = 1 | y_{1,i,t}, y_{2,i,t}) = logit^{-1}(\lambda_{10} + \lambda_{11} y_{1,i,t} + \lambda_{12} y_{2,i,t}) \\ p_{2,i,t} &= P(R_{2,i,t} = 1 | y_{1,i,t}, y_{2,i,t}) = logit^{-1}(\lambda_{20} + \lambda_{21} y_{1,i,t} + \lambda_{22} y_{2,i,t}) \end{aligned} \tag{1}$$

where $y_{1,i,t}$ and $y_{2,i,t}$ represent person i's level of positive affect (PA) and negative affect (NA) at time point t; $R_{1,i,t}$ and $R_{2,i,t}$ are the missing indicators, following bernoulli distributions with probability $p_{1,i,t}$ and $p_{2,i,t}$, respectively. That is, the probability of missingness in PA (or NA) is assumed to be associated with concurrent levels of PA and NA.

R Code for Fitting mlVAR Models in Stan

Step 0: Package installation and data preparation.

First, install and load all necessary packages. One can refer to `https://github.com/stan-dev/rstan/wiki/RStan-Getting-Started` for more detailed instructions on installation of the `rstan` package.

We then load the empirical data (the data format can be found in our paper) and create some new variables to store data information. Specifically, scores on PA and NA are stored in `Y`, a `P` × `T` × 2 (i.e., 217 × 74 × 2) array, with three dimensions representing persons, time points, and outcome variables (i.e., PA an NA) respectively. Between-level predictors (i.e., levels of perceived stress and extraversion) are stored in `X`, a `P` × 3 (i.e., 217 × 3) matrix in which the first column is 1 for each person (for the intercept), and the second and third columns represent levels of perceived stress and extraversion respectively. Since some missing entries in `Y` are due to the fact that participants have different numbers of time points (i.e., the data array is filled up with "`NA`"s after a person's last observation), we create a vector, `TimePoint`, to store the number of time points for each person to ensure that we do not label data points after the last observation as missing values. We also create a set of indicator variables to help locate missing data. Specifically, we use the `ifelse()` function to create an array of missing indicators, `R`. We then use the `which()` function to obtain the indices where the entry is missing. Since `Y` is a three-dimensional array, the location indices matrix `Y_miss` should have three columns, representing the location of missing entries for persons, time points, and outcome variables, respectively. Then we store the three columns in `Y_miss` into `Y_miss1`, `Y_miss2` and `Y_miss3` separately, which will be used to implement some index manipulation in the model script. Finally, we replace all missing entries in `Y` with our chosen arbitrary constant values (i.e., 99), because Stan does not allow for any missing entries in the data set. Then we pass all data, including the missingness indicators and the mean vector `mus_1` and covariance matrix `Sigma_VAR_1` of the distribution for the first observation[1], to Stan as a list variable, named as `stan_data` below for our example.

```
library(rstan)
library(coda)
```

[1] Note that the distribution for the first observation is slightly misspecified, which may cause some biases in model parameters, especially when the number of time points is small. We will discuss other possible specifications in the Discussion section.

```
# load data
load("mlVAR.Rdata")
ID <- unique(data$ID)
P <- length(ID)

# create a vector to store the number of time points for each person
TimePoint <- c()
for (pp in 1:P){
  tmp <- data[data$ID==ID[pp],]
  tt <- nrow(tmp)
  del <- 0
  while(sum(is.na(tmp[tt,c("y1","y2")]))==2){
    del <- del + 1
    tt <- tt - 1
  }
  TimePoint[pp] <- nrow(tmp) - del
}

# store dependent variables and covariates into new variables, Y and X
Y <- array(rep(NA,P*max(TimePoint)*2), c(P,max(TimePoint),2))
X <- matrix(NA, nrow=P, ncol=3)
for(pp in 1:P){
  tmp <- data[data$ID==ID[pp],]
  X[pp,1] <- 1
  X[pp,2] <- unique(tmp$x1)
  X[pp,3] <- unique(tmp$x2)
  for (tt in 1:TimePoint[pp]){
    Y[pp,tt,1] <- tmp$y1[tt]
    Y[pp,tt,2] <- tmp$y2[tt]
  }
}

# missing indicators
R <- ifelse(is.na(Y),1,0) # missing indicators (1: missing)

# create a set of indicator variables to locate missing data
Y_miss <- which(is.na(Y), arr.ind=TRUE)
for(u in 1:dim(Y_miss)[1]){
# NAs caused by different number of time points should not
# be caught by Y_miss
  if(Y_miss[u,2] > TimePoint[Y_miss[u,1]]){
    Y_miss[u,2] <- NA
  }
}
Y_miss <- na.omit(Y_miss)
```

```
Y_miss1 <- Y_miss[,1] # which person has missing entries
Y_miss2 <- Y_miss[,2] # which time point has missing entries
Y_miss3 <- Y_miss[,3] # which variable has missing entries

# replace missing entries with 99
Y[is.na(Y)] <- 99

stan_data <- list(Y = Y,
                  R = R,
                  P = dim(Y)[1],
                  T = max(TimePoint),
                  TimePoint = TimePoint,
                  X = X,
                  K = dim(X)[2],
                  Y_miss1 = Y_miss1,
                  Y_miss2 = Y_miss2,
                  Y_miss3 = Y_miss3,
                  N_miss = dim(Y_miss)[1],
                  mus_1 = c(2,2),
                  Sigma_VAR_1 = diag(2))
```

The following steps are almost identical to the steps described in our paper, except that we add some lines of code to handle NMAR missingness. In the following illustrations, we will only focus on code related to NMAR missing data handling.

Step 1: Defining the “data”, “parameters”, and “transformed parameters” blocks.

The array of missing indicators (i.e., `R`) is declared in the “data” block, and the regression coefficients in Equation 1 are declared in the “parameter” block.

```
modelString = "
data {
int<lower=1> P;         // number of persons
int<lower=1> T;         // maximum number of time points
vector[2] Y[P,T];       // dependent variables
int R[P,T,2];        // missing indicators
int TimePoint[P];       // number of time points for each person
int<lower=1> N_miss;    // total number of missing entries
int Y_miss1[N_miss];    // persons who had missing data
int Y_miss2[N_miss];    // time points with missing data
int Y_miss3[N_miss];    // variables with missing data
int<lower=1> K;    // number of between-level predictors (including intercept)
matrix[P,K] X;    // between-level predictors
```

```
vector[2] mus_1;         // mean vector for the 1st observation
matrix[2,2] Sigma_VAR_1; // covariance matrix for the 1st observation
} // end of the data block

parameters {
vector[N_miss] Y_impute; // a vector of imputed missing values
matrix[P,9] b; // person-specific parameters
matrix[K,9] Coeff; // between-level regression coefficients
vector<lower=0>[9] sigma; // random effect standard deviations
// cholesky factor of the random effect correlation matrix
cholesky_factor_corr[9] L;
matrix[3,2] lambda;// regression coefficients in the missing data model
} // end of the parameters block

transformed parameters {
// population means of person-specific parameters
matrix[P,9] bmu;

// random innovation covariance matrices
vector<lower=-1,upper=1>[P] corr_noise;
vector[2] sd_noise[P];
matrix[2,2] Sigma_VAR[P];

// an array storing observed values in the original data set
// and imputed missing values (i.e., elements in Y_impute)
vector[2] Y_merge[P,T];

for (pp in 1:P) {
// calculate population means of person-specific parameters
bmu[pp] = X[pp,]*Coeff;

// parameter back-transformations
corr_noise[pp] = (exp(2*b[pp,9])-1)/(exp(2*b[pp,9])+1);
sd_noise[pp,1] = exp(b[pp,7]);
sd_noise[pp,2] = exp(b[pp,8]);
// make random innovation covariance matrices
Sigma_VAR[pp,1,1] = sd_noise[pp,1]*sd_noise[pp,1];
Sigma_VAR[pp,2,2] = sd_noise[pp,2]*sd_noise[pp,2];
Sigma_VAR[pp,1,2] = sd_noise[pp,1]*corr_noise[pp]*sd_noise[pp,2];
Sigma_VAR[pp,2,1] = Sigma_VAR[pp,1,2];
} // close loop over persons

// replace 99 in Y_merge with parameters in Y_impute
Y_merge = Y;
for (u in 1:N_miss){
```

```
Y_merge[Y_miss1[u],Y_miss2[u],Y_miss3[u]] = Y_impute[u];
}
} // end of the transformed parameters block
```

Step 2: Defining the “model” and “generated quantities" blocks.

The missing data model is specified in the “model” block. Specifically, the probability of missingness (i.e., `pr`) is specified based on Equation 1, and then a Bernoulli distribution with probability `pr` is assigned to the missing indicator `R`. Finally, normal priors are assigned to regression coefficients in Equation 1.

```
model {
vector[2] mus[P,T];
vector[2] logodds[P,T];
vector[2] pr[P,T]; // probability of missingness

for (pp in 1:P){
// the 1st observation
Y_merge[pp,1,1:2] ~ multi_normal(mus_1, Sigma_VAR_1);

// the likelihood function for dependent variables
for (tt in 2:TimePoint[pp]){
mus[pp,tt,1] = b[pp,1]+b[pp,3]*(Y_merge[pp,tt-1,1]-
b[pp,1])+b[pp,5]*(Y_merge[pp,tt-1,2]-b[pp,2]);

mus[pp,tt,2] = b[pp,2]+b[pp,6]*(Y_merge[pp,tt-1,1]-
b[pp,1])+b[pp,4]*(Y_merge[pp,tt-1,2]-b[pp,2]);

Y_merge[pp,tt] ~ multi_normal(mus[pp,tt], Sigma_VAR[pp]);

// missing data model
for(dd in 1:2){
logodds[pp,tt,dd] = lambda[1,dd]+lambda[2,dd]*Y_merge[pp,tt,1]+
                     lambda[3,dd]*Y_merge[pp,tt,2];
pr[pp,tt,dd] = exp(logodds[pp,tt,dd])/(1+exp(logodds[pp,tt,dd]));
R[pp,tt,dd] ~ bernoulli(pr[pp,tt,dd]);
}
} // close loop over time points

// population distributions on person-specific parameters
b[pp,1:9] ~ multi_normal_cholesky(bmu[pp,1:9],diag_pre_multiply(sigma, L));
} // close loop over persons
// priors on between-level regression coefficients
for(kk in 1:K){
for (ww in 1:9){
Coeff[kk,ww] ~ normal(0,10);
}}

// priors on elements in the random effect covariance matrix
L ~ lkj_corr_cholesky(1);
sigma ~ cauchy(0, 10);

// priors on parameters in the missing data model
for(kk in 1:3){
for(dd in 1:2){
lambda[kk,dd] ~ normal(0,10);
}}
} // end of the model block

generated quantities {
matrix[9,9] bcorr; // random effect correlation matrix
matrix[9,9] bcov; // random effect covariance matrix

bcorr = multiply_lower_tri_self_transpose(L);
bcov = quad_form_diag(bcorr, sigma);
} // end of the generated quantities block

" # end of "modelString"
```

Step 3: Running the Stan model and extracting the results.

In this step, we add `lambda` to the parameter list to extract their posteriors. This allows us to investigate the relationship between missingness in PA (or NA) and values of PA and NA.

```
writeLines(modelString, con = "mlvar.stan")
parameterlist <- c("Coeff","sigma","bcov","lambda","b","corr_noise","Sigma_VAR")
VARmodel <- stan(file = 'mlvar.stan', data = stan_data, seed = 1,
                 chains = 2, cores = 2, iter = 25000, warmup = 5000,
                 pars = parameterlist)
codaSamples <- mcmc.list(lapply(1:ncol(VARmodel), function(x){
  mcmc(as.array(VARmodel)[, x, ])
}))
resulttable <- zcalc(codaSamples)
```

R Code for Fitting mlVAR Models in JAGS

Step 0: Package installation and data preparation.

First, install JAGS from `http://mcmc-jags.sourceforge.net`, and then install and load all necessary packages.

The data preparation work is similar to Stan, except that different indicator variables are generated due to different restrictions in Stan and JAGS. Specifically, JAGS does not allow for specifications of multivariate distributions on partially missing records (i.e., only PA or NA is missing at a particular time point), which we have in the current application. To work around this, we create a set of indicator variables to locate missing data. Briefly, the location of time points with fully missing (i.e., both PA and NA are missing at a particular time point) or partially missing records for each person is stored in `Tmiss`, and the location of time points with fully observed records (i.e., both PA and NA are observed at a particular time point) for each person is stored in `Tseen`. The number of time points for each person in `Tmiss` and `Tseen` is stored in `nmiss` and `nseen` respectively. Later we will show how these indicator variables will play a role in the model fitting process. Note that if there are no partially missing records in the data set, we can skip the above procedure. All data, including the missingness indicators, the scale matrix of the IW prior, and the mean vector `mus_1` and precision matrix `prec_VAR_1` of the distribution for the first observation, are passed to JAGS as a list variable, named as `jags_data` below for our example.

```
library(rjags)

# load data
load("mlVAR.Rdata")
ID <- unique(data$ID)
P <- length(ID)

# create a vector to store the number of time points for each person
TimePoint <- c()
for (pp in 1:P){
  tmp <- data[data$ID==ID[pp],]
  tt <- nrow(tmp)
  del <- 0
  while(sum(is.na(tmp[tt,c("y1","y2")]))==2){
    del <- del + 1
    tt <- tt - 1
  }
  TimePoint[pp] <- nrow(tmp) - del
}

# store dependent variables and covariates into new variables, Y and X
Y <- array(rep(NA,P*max(TimePoint)*2), c(P,max(TimePoint),2))
X <- matrix(NA, nrow=P, ncol=3)
for(pp in 1:P){
  tmp <- data[data$ID==ID[pp],]
  X[pp,1] <- 1
  X[pp,2] <- unique(tmp$x1)
  X[pp,3] <- unique(tmp$x2)
  for (tt in 1:TimePoint[pp]){
    Y[pp,tt,1] <- tmp$y1[tt]
    Y[pp,tt,2] <- tmp$y2[tt]
  }
}

# missing indicators
R <- ifelse(is.na(Y),1,0) # missing indicators (1: missing)

# create a set of indicator variables to locate missing data
Index <- matrix(NA, nrow = dim(Y)[1], ncol = dim(Y)[2])
Tmiss <- matrix(NA, nrow = dim(Y)[1], ncol = dim(Y)[2])
Tseen <- matrix(NA, nrow = dim(Y)[1], ncol = dim(Y)[2])
nmiss <- c()
nseen <- c()

for(pp in 1:dim(Y)[1]){
  for(tt in 1:TimePoint[pp]){
    # if two variables are both observed, return 0;
    # if at least one variable is missing, return 1;
    Index[pp,tt] <- ifelse(sum(is.na(Y[pp,tt,]))==0, 0, 1)
  }
}

for(pp in 1:dim(Y)[1]){
  # number of time points with (partially) missing records
  nmiss[pp] <- length(which(Index[pp,] %in% 1))
  if(nmiss[pp] != 0){
  # location of time points with (partially) missing records
```

```
    Tmiss[pp, 1:nmiss[pp]] <- which(Index[pp,] %in% 1)
  }
  # number of time points with fully observed records
  nseen[pp] <- length(which(Index[pp,] %in% 0))
  # location of time points with fully observed records
  Tseen[pp, 1:nseen[pp]] <- which(Index[pp,] %in% 0)
}

# P1 only contains person IDs with missing records
P1 <- c(1:dim(Y)[1])
P1 <- P1[-which(nmiss %in% 0)]

# the scale matrix of the inverse Wishart prior
W <- diag(9)

jags_data <- list(Y = Y,
                  R = R,
                  P = dim(Y)[1],
                  P1 = P1,
                  TimePoint = TimePoint,
                  X = X,
                  K = dim(X)[2],
                  W = W,
                  nmiss = nmiss,
                  nseen = nseen,
                  Tmiss = Tmiss,
                  Tseen = Tseen,
                  mus_1 = c(2,2),
                  prec_VAR_1 = diag(2))
```

The following steps are almost identical to the steps described in our paper, except that we add some lines of code to handle NMAR missingness, which are identical to the relevant Stan code presented above. The only difference is that users do not need to declare data and parameters before using them in the model script.

Step 1: Defining models and priors.

```
modelString = "
model{

for (pp in P1) {
# the likelihood function for dependent variables
for (tt in Tmiss[pp,1:nmiss[pp]]){
```

```
mus[pp,tt,1] <- b[pp,1]+b[pp,3]*(Y[pp,tt-1,1]-b[pp,1])+b[pp,5]*(Y[pp,tt-1,2]-b[pp,2])
mus[pp,tt,2] <- b[pp,2]+b[pp,6]*(Y[pp,tt-1,1]-b[pp,1])+b[pp,4]*(Y[pp,tt-1,2]-b[pp,2])
Y[pp,tt,1] ~ dnorm(mus[pp,tt,1], pow(sd_noise[pp,1], -2))
Y[pp,tt,2] ~ dnorm(mus[pp,tt,2], pow(sd_noise[pp,2], -2))
} # close loop over Tmiss
} # close loop over persons with missing data

for (pp in 1:P) {
# the 1st observation
Y[pp,1,1:2] ~ dmnorm(mus_1, prec_VAR_1)

# the likelihood function for dependent variables
for (tt in Tseen[pp,2:nseen[pp]]){
mus[pp,tt,1] <- b[pp,1]+b[pp,3]*(Y[pp,tt-1,1]-b[pp,1])+b[pp,5]*(Y[pp,tt-1,2]-b[pp,2])
mus[pp,tt,2] <- b[pp,2]+b[pp,6]*(Y[pp,tt-1,1]-b[pp,1])+b[pp,4]*(Y[pp,tt-1,2]-b[pp,2])
Y[pp,tt,1:2] ~ dmnorm(mus[pp,tt,1:2], prec_VAR[pp,1:2,1:2])
} # close loop over Tseen

# missing data model
for (tt in 1:TimePoint[pp]){
for(dd in 1:2){
logodds[pp,tt,dd] <- lambda[1,dd]+lambda[2,dd]*Y[pp,tt,1]+lambda[3,dd]*Y[pp,tt,2]
pr[pp,tt,dd] <- exp(logodds[pp,tt,dd])/(1+exp(logodds[pp,tt,dd]))
R[pp,tt,dd] ~ dbern(pr[pp,tt,dd])
}
} # close loop over time points

# population distributions on person-specific parameters
bmu[pp,1:9] <- X[pp,]%*%Coeff
b[pp,1:9] ~ dmnorm(bmu[pp,1:9], bpre[1:9,1:9])

# parameter back-transformations
corr_noise[pp] <- (exp(2*b[pp,9])-1)/(exp(2*b[pp,9])+1)
sd_noise[pp,1] <- exp(b[pp,7])
sd_noise[pp,2] <- exp(b[pp,8])
# make random innovation precision matrices
Sigma_VAR[pp,1,1] <- sd_noise[pp,1]*sd_noise[pp,1]
Sigma_VAR[pp,2,2] <- sd_noise[pp,2]*sd_noise[pp,2]
Sigma_VAR[pp,1,2] <- sd_noise[pp,1]*corr_noise[pp]*sd_noise[pp,2]
Sigma_VAR[pp,2,1] <- Sigma_VAR[pp,1,2]
prec_VAR[pp,1:2,1:2] <- inverse(Sigma_VAR[pp,1:2,1:2])

} # close loop over persons

# priors on between-level regression coefficients
```

```
for (kk in 1:K){
for (ww in 1:9){
Coeff[kk,ww] ~ dnorm(0, .01)
}}

# priors on the random effect precision matrix
bpre ~ dwish(W, 10)

# priors on parameters in the missing data model
for(kk in 1:3){
for(dd in 1:2){
lambda[kk,dd] ~ dnorm(0, .01)
}}

# obtain the random effect covariance matrix and
# random effect standard deviations
bcov <- inverse(bpre)
for(ww in 1:9){
sigma[ww] <- sqrt(bcov[ww,ww])
}

} # end of the "model" block
" # end of "modelString"
```

Step 2: Running the JAGS model and extracting the results.

```
writeLines(modelString, con = "mlvar.txt")
inits1 <- list(.RNG.name = "base::Wichmann-Hill", .RNG.seed = 1)
inits2 <- list(.RNG.name = "base::Wichmann-Hill", .RNG.seed = 2)
jagsModel <- jags.model(file = "mlvar.txt", data = jags_data,
                        inits = list(inits1,inits2),
                        n.chains = 2, n.adapt = 4000)
update(jagsModel, n.iter = 1000)
parameterlist <- c("Coeff","sigma","bcov","lambda","b","corr_noise","Sigma_VAR")
codaSamples <- coda.samples(jagsModel,
                            variable.names = parameterlist,
                            n.iter = 20000)
resulttable <- zcalc(codaSamples)
```