Supplemental Materials

1. Sample SAS Codes and SAS macros

   /* The following are SAS function and macros are defined before running the example */
   /* The defined function will be stored in C:\SBR2021. */
   /* To run the example, DIA dataset “Chapter15_example..sas7bdat” */
   /* should be in the folder of C:\SBR2021. */

   libname mylib "C:\SBR2021";

   /*** Define MO function to get components for CR estimates ***/
   proc fcmp outlib=mylib.funcs.dia;
   /* compute Sigma_{mo} inv(Sigma_{oo}) (\mut - \mup) */
   /* assuming monotonous missing */
   subroutine MO(out[*], mut[*], mup[*], sigma[*,*], nmiss);
   outargs out;  /* last nmiss elements in OUT are filled */

   array Smo[1] / nosymbols;
   array Soo[1] / nosymbols;
   array tmp[1] / nosymbols;
   array difo[1] / nosymbols;
   array mvec[1] / nosymbols;
   array res[1] / nosymbols;

   p = dim(mut, 1);
   nM = nmiss;
   nO = p - nM;

   call dynamic_array(Smo, nM, nO);
   call dynamic_array(Soo, nO, nO);
   call dynamic_array(tmp, nM, nO);
   call dynamic_array(difo, nO);
   call dynamic_array(mvec, p);
   call dynamic_array(res, nM);

   /* this mvec is not necessary in monotone missing, but the code */
   /* works for arbitrary missing, in case we need it in the future. */
   do i = 1 to p;
     if (i <= nO) then mvec[i] = 0;
     else mvec[i] = 1;
   end;

   /* pick out Soo according to mvec[] */
   io = 1;
do i = 1 to p;
    if(mvec[i] eq 0) then do;
        jo = 1;
        do j = 1 to p;
            if(mvec[j] eq 0) then do;
                Soo[io,jo] = Sigma[i,j];
                jo = jo + 1;
                end;
                end;
            io = io + 1;
            end;
        end;
    end;
    call inv(Soo, Soo);

    /* pick out Smo according to mvec[] */
    im = 1;
    do i = 1 to p;
        if(mvec[i] eq 1) then do;
            jo = 1;
            do j = 1 to p;
                if(mvec[j] eq 0) then do;
                    Smo[im,jo] = Sigma[i,j];
                    jo = jo + 1;
                    end;
                    end;
                im = im + 1;
                end;
            end;
        end;
    end;
    call mult(Smo, Soo, tmp);

    /* compute the mean diffs for the observed elements */
    io = 1;
    do i = 1 to p;
        if(mvec[i] eq 0) then do;
            difo[io] = mut[i] - mup[i];
            io = io+1;
            end;
        end;
    end;
    call mult(tmp, difo, res);

    im = 1;
    do i = 1 to p;
        if(mvec[i] eq 1) then do;
out[i] = res[im];
im = im+1;
end;
end;
endsub;
run;

/* This macro does the needed transformation for MAR, J2R, CIR, and CR. */
/* Need to supply the dimension, as well as the beta and dirch prior information. */
/* nrep = number of repeated measures in the study */
/* p1hat_prior = beta(n1, n2), n1=# of completers, n2=# of dropouts */
/* npattern = numbers of dropouts at time 2, ..., NREP, and completers */
%macro CBI_Comp(nrep=, p1hat_prior=beta(36,7), npattern = (1 1 4 2 36) );
  parms p1hat;
prior p1hat ~ &p1hat_prior;
array p[&nrep];

parms p;
array npattern[&nrep] &npattern;
prior p ~ dirich(npattern);

beginnodata;
array diff[&nrep]; call subtractmatrix(mut, mup, diff);
array m[&nrep]; /* placeholder for MO subroutine outcome */
array d[&nrep,&nrep];

/* looping through nmiss = 1 to (n-1) scenarios */
do nM = 1 to (&nrep - 1);
  call MO(m, mut, mup, sigma, nM);
nO = &nrep - nM;
do i = 1 to nO; d[nO,i] = diff[i]; end; /* first nO observed cases */
do i = (nO+1) to &nrep; d[nO,i] = m[i]; end; /* the nM missing cases */
end;
do i = 1 to &nrep; d[&nrep,i] = diff[i]; end;

d_mar = diff[&nrep];  **** trt difference under MAR ****;
d_j2r = p1hat*diff[&nrep];  **** trt difference under J2R ****;
call mult(p, diff, d_cir);
d_cr = 0;
do i = 1 to &nrep;
  d_cr = d_cr + p[i] * d[i,&nrep];
end;
endnodata;
%mend;
/* Import Data: assume the DIA dataset “Chapter15_example..sas7bdat” is in C:\SBR2021. */
data DIA;
   set mylib.Chapter15_example;
   by patient visit;
   retain w1 w2 w4 w6 count;
   if first.patient then do; w1=.; w2=.; w4=.; w6=.; count=0; end;
   if visit=4 then w1=change;
   if visit=5 then w2=change;
   if visit=6 then w4=change;
   if visit=7 then w6=change;
   count=count+1;
   if last.patient then do; *Remove those with intermediate missing data; 
      if count = (visit-3) then output;
      else put "remove patient" patient;
   end;
run;

proc standard data=DIA out=DIA mean=0;
 var basval; * centralize baseline value so overall mean = 0 *
run;
data dia2;
   set dia;
   array y[4] w1 w2 w4 w6;
   array r[4];
   trtn = (therapy='DRUG');
do i = 1 to 4;
   if (y[i] eq .) then r[i] = 1;
   else r[i] = 0;
   end;
do i = 1 to 4;
   L = i;
   if (y[i] eq .) then do;
      L = (i - 1);
      leave;
      end;
   end;
drop i;
   rename w1=y1 w2=y2 w4=y3 w6=y4;
run;
data long;
   set dia2;
   array yy{4} y1-y4;
do time = 1 to 4; resp = yy{time}; output; end;
drop y1-y4;
run;

ods html; ods listing;
proc mixed data=long;
  class patient time trtn;
  model resp = time time*trtn basval*time / DDFM=KR;
  repeated time / subject = patient type=UN;
  lsmeans time*trtn time;
  estimate 'Mixed result' time*trtn 0 0 0 0 0 -1 1 / cl;
  title "Mixed model for all patients without trt";
run;

proc tabulate data=DIA2;
  class trtn L;
  var r1 - r4;
  table trtn, (L all)*n*F=7.;
  title "Number of Dropouts by Pattern";
run;

%let n = 4;
%let seed=201911;
options cmplib=mylib.funcs;
proc mcmc data=DIA2 nmc=200000 thin=1 seed=&seed outpost=DIA_OUT(keep=iteration p1hat d_mar d_cr d_j2r d_cir) ntu=2000 diag=none monitor=(d_mar d_cr d_j2r d_cir);
  array y[&n] y1-y&n;
  array r[&n] r1-r&n;
  array mu[&n];
  array Sigma[&n,&n];
  array S[&n,&n] ;
  array mup[&n];
  array mut[&n];
  array beta[&n];
  beginincnst; call identity(s); endcnst;
%
  %cbi_comp(nrep=&n, p1hat_prior=beta(63,20), npattern = (6 5 9 63));
  *** there were 63 completes out of the 83 patients in the drug group ***;
  *** the number of dropout patients at week 1, 2, 4 were 5, 6, and 9 ***;
  parms Sigma ;
  parm (mup:) 0;
  parm (mut:) 0;
  parm (beta:) 0;
prior Sigma ~ iwish(&n, S);
prior mup: mut: beta: ~ n(0, var=1e6);

do i = 1 to &n;
    if (trtn eq 0) then mu[i] = mup[i] + beta[i]*basval;
    else mu[i] = mut[i] + beta[i]*basval;
end;
model y ~ mvn(mu, Sigma);
title "MCMC results for CBI";
run;

2. Sample R-Stan Codes

library(rstan)
library(reshape2)
library(dplyr)

#Anti depression trial
#loading data file
dia_dat = read.csv("dia_data.csv", header=TRUE)

#centering baseline variable
dia_dat[,"basval"] = dia_dat[,"basval"] - mean(dia_dat[,"basval"])

#defining the treatment vector
treatVec = as.numeric(dia_dat$THERAPY == "DRUG")

#number of post assignment time points
nNumTimePoint = 4

#outcome values post assignment vectorized
yVec = c(t(as.matrix(dia_dat[,c("w1","w2","w4","w6")])))

#identifying indices of that are missing
missInd = which(is.na(yVec))

#filling missing values as -1000 so STAN won't fail
yVec[missInd] = -1000

#creating missing indicators matrix
RAll = (is.na(dia_dat[,c("w1","w2","w4","w6")]))*1
# Identifying the number missing for each period in the control group "f" vector
numMisPerC = apply(RAll[treatVec==0,],2,sum)
numMisPerC = c(numMisPerC[2:length(numMisPerC)]) - c(numMisPerC[1:(length(numMisPerC)-1)])
numMisPerC[length(numMisPerC)+1] = sum(1-treatVec) - sum(numMisPerC)

# Identifying the number missing for each period in the treatment group "f" vector
numMisPerT = apply(RAll[treatVec==1,],2,sum)
numMisPerT = c(numMisPerT[2:length(numMisPerT)]) - c(numMisPerT[1:(length(numMisPerT)-1)])
numMisPerT[length(numMisPerT)+1] = sum(treatVec) - sum(numMisPerT)

# Patterns of monotone missingness observed in the data
Rpattern = distinct(data.frame(RAll))
numMisPerRow = apply(Rpattern,1,sum)
Rpattern = as.matrix(Rpattern[numMisPerRow!=0,])

# Preparing the data that would be passed to STAN
datForStan = list(K = nNumTimePoint, # Number of post randomization time points
                  J = 1, # Number of covariates
                  numPatterns = nNumTimePoint - 1, # Number of monotone patterns
                  num_miss = length(missInd), # Number of missing outcomes values
                  N = dim(dia_dat)[1], # Number of units
                  allSize = length(yVec), # Number of timepoints*number of units
                  x = as.matrix(dia_dat$basval), # Covariate matrix
                  y = yVec, # Outcomes matrix
                  R = Rpattern, # Patterns observed in the data
                  W = treatVec, # Treatment indicator
                  miss_index = missInd, # Missingness index
                  priorMiss = rep(1,nNumTimePoint), # Prior used for the Dirichlet distribution of missing values
                  numMissPer0 = as.vector(numMisPerC), # Number of missing values in the control group per period
                  numMissPer1 = as.vector(numMisPerT-1) # Number of missing values in the treatment group per period
)

# Running STAN with LKJ prior for covariance matrix
fitCalc <- stan(file = "BaseLine.stan", # Stan program
data = datForStan, # Named list of data
chains = 3, # Number of Markov chains
warmup = 500, # Number of warmup iterations per chain
iter = 4000, # Total number of iterations per chain
cores = 1, # Number of cores (could use one per chain)
refresh = 0, # No progress shown
control = list(max_treedepth = 15))
#obtaining posterior draws
matrix_of_draws <- data.frame(as.matrix(fitCalc))

#posterior samples of the different estimands
resultsUse = matrix_of_draws[,c("MAR","CR","J2R","CIR","RTB")]

#summarizing the posterior using mean, sd and credible intervals
sumRes = cbind(apply(resultsUse,2,mean),apply(resultsUse,2,sd),apply(resultsUse,2,quantile,c(0.025)),apply(resultsUse,2,quantile,c(0.975)))

columns name
colnames(sumRes) = c("mean","sd","lower_95_CI","upper_95_CI")

#summarizing results into a table
write.csv(sumRes,"c:\BSWG\resultsDIA_dat.csv")

FUNCTION: BaseLine.Stan
functions{
//function to insert samples missing values x_miss into their missing places using the missing_index

//x_obs is the vector of all observed values
vector mergeMissing( int[] miss_indexes , vector x_obs , vector x_miss) {
    int N = dims(x_obs)[1];
    int N_miss = dims(x_miss)[1];
    vector[N] merged;

    merged = x_obs;
    for ( i in 1:N_miss )
        merged[ miss_indexes[i] ] = x_miss[i];

    return merged;//returning merged vector of observed and imputed missing
}

//function to compute the covariance matrix of the missing data conditional on the observed data
//varCovMat is the covariance matrix, diffVec is the difference between observed outcome to mean

real obsMissCondCov(matrix varCovMat, vector diffVec, int[] colObsMis) {
    int nCols = dims(colObsMis)[1];
    int nColMis = sum(colObsMis);
    int nColObs = nCols - sum(colObsMis);
    vector[nColObs] diffVecP;
    int io;
```c++
int im;
int jo;
int jm;
matrix[nColObs,nColObs] Soo;
matrix[nColMis,nColObs] Smo;
vector[nColMis] res;

io = 1;
im = 1;
for(i in 1:nCols)
{
    jo = 1;
    jm = 1;
    for(j in 1:nCols)
    {
        if(colObsMis[i] == 0 && colObsMis[j] == 0)
        {
            Soo[io,jo] = varCovMat[i,j];
            jo += 1;
        }
        if(colObsMis[i] == 1 && colObsMis[j] == 0)
        {
            Smo[im,jm] = varCovMat[i,j];
            jm += 1;
        }
    }
    if(colObsMis[i] == 0)
    {
        diffVecP[io] = diffVec[i];
        io += 1;
    } else {
        im += 1;
    }
}

res = Smo * inverse(Soo) * diffVecP;
return(res[nColMis]);
}
```

data {
    int<lower=1> K; //number of time points
    int<lower=0> num_miss; //number of missing values
}
int<lower=1> J; //number of covariates
int<lower=0> N; //number of units
int<lower=0> allSize; //number of units*number of time points
int<lower=1> numPatterns; //number of missing data patterns
vector[J] x[N]; //covariates matrix
vector[N*K] y; //vectorized outcome
int numMissPer0[K]; //number of missing values in control
int numMissPer1[K]; //number of missing values in treatment
int W[N]; //treatment indicator
int miss_index[num_miss]; //missing indices in vectorized outcome
vector[K] priorMiss; //prior for Dirichlet distribution "f"
int R[numPatterns,K]; //types montone missing data patterns
}

parameters {
  matrix[K, J] beta; //slope parameters for covariates in each time point
  vector[K] alpha0; //mean at average slope value for control for each time point
  vector[K] alpha1; //mean at average slope value for treated for each time point
  cholesky_factor_corr[K] L_Omega; //two matrices to use for LKJ prior
  vector<lower=0>[K] L_sigma;
  vector[num_miss] y_impute; //imputed values vector
  simplex[K] propMisPer0; //proportions of missing values in each time point control
  simplex[K] propMisPer1; //proportions of missing values in each time point treated
}

model {
  vector[K] mu[N]; //mean vector for MMRM model
  matrix[K, K] L_Sigma; //covariance matrix sampling for LKJ prior
  vector[K] y_mergeM[N]; //imputed and observed vectorized outcomes
  vector[N*K] y_merge; //imputed and observed outcomes in matrix format

  //mean vector for each unit for all time points post randomization
  for (n in 1:N)
  {
    mu[n,] = W[n] * alpha1 + (1-W[n]) * alpha0 + beta * x[n,];
  }

  //Covariance matrix with LKJ prior
  L_Sigma = diag_pre_multiply(L_sigma, L_Omega);

  to_vector(beta) ~ normal(0, 100); //prior distributions for slope parameter
  alpha0 ~ normal(0, 100); //prior for mean at average covariates controls
  alpha1 ~ normal(0, 100); //prior for mean at average covariates treatment
  L_Omega ~ lkj_corr_cholesky(1); //prior LKJ
  L_sigma ~ student_t(3, 0, 5); //prior LKJ
//merging imputed outcomes with observed outcome
y_merge = mergeMissing(miss_index, to_vector(y), y_impute);

//reshaping the vectorized outcome into a matrix
for (j in 0:(N-1))
    y_mergeM[j+1] = y_merge[(K*j +1):(K*(j +1))];

//MMRM model
y_mergeM ~ multi_normal_cholesky(mu, diag_pre_multiply(L_sigma, L_Omega));

//Multinomial for the control missing values
numMissPer0 ~ multinomial(propMisPer0);
//Multinomial for the treated missing values
numMissPer1 ~ multinomial(propMisPer1);

//priors for number of missing values in control and treated
propMisPer0 ~ dirichlet(priorMiss);
propMisPer1 ~ dirichlet(priorMiss);
}
generated quantities {
    //Covariance matrix samples
    matrix[K, K] Omega;
    matrix[K, K] Sigma;
    //hypothetical outcomes "plug-in" method
    real MAR;
    real CR;
    real J2R;
    real CIR;
    real RTB;
    real resCR;

    //Covariance matrix
    Omega = multiply_lower_tri_self_transpose(L_Omega);
    Sigma = quad_form_diag(Omega, L_sigma);

    {
        //calculating hypothetical outcomes based on the formulas in the paper Equation (3)
        vector[K] diffVec;
        diffVec = alpha1 - alpha0;
        CR = propMisPer1[K]*diffVec[K];
        for(i in 1:numPatterns)
        {
            resCR = obsMissCondCov(Sigma, diffVec, R[i,]);
        }
    }
CR += propMisPer1[i]*resCR;
}
}
J2R = propMisPer1[K]*(alpha1[K] - alpha0[K]);
CIR = sum(propMisPer1[1:K].*(alpha1[1:K] - alpha0[1:K]));
RTB = propMisPer1[K]*alpha1[K] - propMisPer0[K] * alpha0[K];
MAR = alpha1[K] - alpha0[K];