Supplementary information

Multivariate R function

The function takes a single argument, a list. This list contains two further lists, `y` and `s`, and the vector `d`, all of length equal to the total number of studies. The list `s` contains the within-study covariance matrices S_{di}. The vector `d` contains the design of each study, (e.g. *AB*, *AC*, *BDE*). Each treatment should be allocated a letter, with *A* being the reference treatment. The notation closely follows that of the main paper. *Note that the program requires the treatments to be the first set letters of the alphabet. The code will not work if the treatments are named differently, or if the first set of letters are not used. For example, if there are six treatments then the treatments MUST be called A,B,C,D,E and F.*

Each design should be referred to in the same way across studies. For example, if the design involving treatments A and B is called "AB" in one study, then it should not be called "BA" in another. The order of the letters in design names need not be alphabetical (for example, the "BA" design is allowed) but the treatment effects must be specified for each treatment arm in the order used in the design name, relative to the first treatment. For example, if a design is referred to as "FEA" then the corresponding outcome vector will consist of treatment effects of treatment E relative to F, followed by treatment A relative to F. See the format of the data below for more information.

In the paper the stacking of the outcome data is performed by design. This is in order to keep the exposition of the statistical methods as clear as possible. However this is not necessary because all matrices (such as M1 and M2) are defined in such a way that the stacking can be performed in any order. The code therefore does not require the stacking of the data to be performed by design, but the analyst can enforce this by entering the data so that each design is entered into the outcome data lists in turn.

The format of the data

Note that we take each treatment comparison in turn when forming the estimated treatment effect vectors and within-study covariance matrices. For example, imagine in the first study that there are five treatment arms, three outcomes and the design is *ABCDE*. Then the first entry in d is "ABCDE" and the first entry in the y and s lists are the concatenation of 12 values, and a 12 by 12 matrix, respectively. The first three values in this entry of y are three treatment effects for the AB comparison, the next three values are three treatment effects for the AC comparison, the next three values are for the AD comparison and the final three values are for the AE comparison. The entries of the first entry of s contain the corresponding within-study covariance matrix.

Handling missing values

Missing values are indicated by NA in the estimated treatment effects `y`. They are handled by providing infinite within-study variances (diagonal entries of the matrices in `s`) and corresponding within-study correlations of zero.

Main output

The main results included in the output are:

est: the estimated treatment effects; this is the estimate of delta_mv.

Cov: the covariance matrix for the estimated treatment effects.

se: the standard error of the estimated treatment effects

Sigma_Omega and Sigma_Beta, the *(truncated*) estimates of the unknown variance components. These are the inconsistency and the between-study heterogeneity covariance matrices, respectively.

To see an example of what these results mean in practice, relate the results below to the trivariate results shown in Tables 1 and 2 in the main paper. Untruncated estimates (".u") of and estimates under the assumption of consistency ("_con") are also given (but are not provided in the main paper).

The Matrix and the MASS packages are required for the program to run.

R code: cut and paste into R.

```
fn_mv=function(all_data,impute=0){

one=strsplit(all_data$d[1], "")[[1]]

contrastsone=length(one)-1

p=length(all_data$y[[1]])/contrastsone

smat=bdiag(all_data$s)  # Create and fill matrix S

Y=matrix(unlist(all_data$y), ncol=1);  R= diag(c(1-1*(is.na(Y))))

Y[which(is.na(Y)==TRUE)]=impute

n=nrow(Y)/p ## p is the dimension of the multivariate meta so n is the number of constrasts

############## Create and fill UNIVARIATE network design matrix X  ##############

trts <- 0     # Number of treatments, including placebo

for(i in 1:26)  trts <- trts + grepl(LETTERS[i], paste(all_data$d,collapse=""))

X=NULL

P1=matrix(nrow=0, ncol=0)

for(i in 1:length(all_data$d))

{   # For each study d, create a small matrix [no. active trts in study i] x [total no. trts-1].

des_letters=strsplit(all_data$d[i], "")[[1]]

des_numbers=match(des_letters, LETTERS)

cons=length(des_numbers)-1

X1=matrix(0, nrow=cons, ncol=trts)

P1=bdiag(P1, matrix(0.5, nrow=cons, ncol=cons))
# Create the empty X1 matrix;

for(j in 1:cons)  X1[j, des_numbers[j+1]]=1  # Add 1's in the columns of active treatments;

X1[,des_numbers[1]]=-1                    # Add a -1 in the column of the (relative) reference trt;

X1 <- X1[, -1]

X=rbind(X, X1, deparse.level = 0)           # Add small X1 matrices to create full design matrix X

}

diag(P1)=1; P1=as.matrix(P1)

### P1 is matrix P1 from the previous 2015 paper, created differently to previous code now we have mv
data, and is called M1 in the present paper.

designs=NULL

for(i in 1:length(all_data$y))

{

the_number=length(all_data$y[[i]])/p

new_designs=rep(all_data$d[i], the_number)

designs=c(designs, new_designs)

}

P2 <- matrix(0, nrow=n, ncol=n)

for(i in 1:n)

for(j in 1:n)

{

if(designs[i]==designs[j] & sum(X[i,]==X[j,])==trts-1) P2[i,j]=1

if(designs[i]==designs[j] & sum(X[i,]==X[j,])!=trts-1) P2[i,j]=0.5

}


### P2 is matrix P2 from the previous 2015 paper, and is called M2 in the present paper.

############## Create and fill MULTIVARIATE network design matrix Xmv using X and call it X
##############

Xmv=NULL
```

```
total_con=nrow(X)

for(i in 1:total_con)

{

N=NULL

for(j in 1:p)

{

N[[j]]=matrix(X[i,], nrow=1)

}

X2=bdiag(N)

Xmv=rbind(Xmv, as.matrix(X2))

}

### Having created the necessary matrices, now we can start the estimation.

### All the above was just formatting the data and creating the necessary matrices to describe the data.

X=Xmv

W=solver(smat)

### Now X is the MV design matrix,

Hat=X%*%solve(t(X)%*%W%*%X)%*%t(X)%*%W

res=Y-Hat%*%Y

Q=W%*%res%*%t(res)%*%R

Q=matrix(block(Q, n, p), ncol=1)

### Q is the entire Q_net from 2015 but is now a p by p matrix, as appropriate for mvmeta.

I = diag(nrow(Y))

A=(t(I-Hat))%*%W

B=(t(I-Hat))%*%R

### Perform consistency estimation, also do some calculation for inconsistency analysis whilst we are at it.

E=matrix(block(B, n, p), ncol=1)

triple=extra_cross_prod(A, B, n, p, P1, P2)
```

```
C=triple$C; D=triple$D

### The above lines get the coefficients of the estimating equations. We now do the consistency estimation.

Sigma_con = (solve(C))%*%(Q-E)

Sigma_con = matrix(Sigma_con, ncol=p)

Sigma_con_trun=truncater(Sigma_con)

### Sigma_con_trun is the truncated consistency model's between-study covariance matrix. This is
tau2_beta_con in the univariate case!

### We now move onto the inconsistency model estimation, so we now require Qhet.

unique_d <- unique(all_data$d) # Each unique design

d <- length(unique_d) # Number of unique studies

Qhet=0; sumEd=0; sumCd=0

for(i in 1:d){  # For design i, record:

nd <- sum(unique_d[i]==all_data$d) #nd here is the number of studies not contrasts

which_d <- which(unique_d[i]==all_data$d)  # Which studies have design i?

yd <- matrix(unlist(all_data$y[which_d]), ncol=1)  ; Rd= diag(c(1-1*(is.na(yd))))

yd[which(is.na(yd)==TRUE)]=impute

cd <- nchar(unique_d[i])-1  # No. of treatments compared to reference treatment

icd <- as.matrix(diag(cd*p)) # Identity matrix Icd

xd <- matrix(rep(icd, nd), ncol=cd*p, byrow=T)  # Stacked identity matrices

sd <- bdiag(all_data$s[which_d]); wd=solver(sd)

estd= (ginv(as.matrix(t(xd)%*% wd%*%xd)))%*%t(xd)%*%wd%*%yd

resd=yd-xd%*%estd

Qd=block(wd%*%resd%*%t(resd)%*%Rd, nd*cd, p)

Qhet=Qhet+Qd

Hatd=xd%*%ginv(as.matrix(t(xd)%*%wd%*%xd))%*%t(xd)%*%wd

Id=diag(nrow(yd))

Ad=(t(Id-Hatd))%*%wd

Bd=(t(Id-Hatd))%*%Rd
sumEd=sumEd+block(Bd, nd*cd, p)

J=matrix(0.5, nrow=cd, ncol=cd)+0.5*diag(cd)

Js=rep(list(J), nd)

Pd=bdiag(Js)

Extrad=extra_cross_prod(Ad, Bd, nd*cd, p, Pd, Pd)

sumCd=sumCd+Extrad$C

}

Qhet=matrix(Qhet, ncol=1); sumEd=matrix(sumEd, ncol=1)

### The above lines get the estimating coefficients. We are now ready to perform the estimation!

Sigma = (solve(sumCd))%*%(Qhet-sumEd)

Omega = (solve(D))%*%(Q-E - C%*%Sigma)

### Thats the untruncated estimates in vector form

Sigma = matrix(Sigma, ncol=p)

Sigma_trun = truncater(Sigma)

Omega = matrix(Omega, ncol=p)

Omega_trun = truncater(Omega)

### Do the inference for the treatment effects!

Con_V = smat + P1%x% Sigma_con_trun; W_CON=solver(Con_V)

Incon_V = smat + P1%x% Sigma_trun + P2%x% Omega_trun; W_INCON=solver(Incon_V)

vcov_con=(solve(t(Xmv) %*% W_CON %*% Xmv)) ; vcov_incon=(solve(t(Xmv) %*% W_INCON %*% Xmv))

est_con=vcov_con%*%t(Xmv)%*%W_CON%*%Y

est=vcov_incon%*%t(Xmv)%*%W_INCON%*%Y

return(list(est=est, est_con=est_con, Cov= vcov_incon, Cov_con= vcov_con, se= diag(vcov_incon)^0.5,
se_con= diag(vcov_con)^0.5, Sigma_Omega=Omega_trun, Sigma_Beta=Sigma_trun,
Sigma_Beta_con=Sigma_con_trun, Sigma_Omega.u=Omega, Sigma_Beta.u=Sigma,
Sigma_Beta_con.u=Sigma_con))

}

block=function (Mat, n, p)

{
```

```
# Block is a function that computes the block trace.

the_sum=0

for(i in 1:n)

{

part=(((i-1)*p+1):((i-1)*p+p))

diag_block=Mat[part, part ]

the_sum=the_sum+diag_block

}

the_sum

}

extra_cross_prod=function (A, B, n, p, Ind1, Ind2)

{

# Extra_cross_prod calculates triple sums in the matrix equations

C=0; D=0

for(i in 1:n)

{

for(j in 1:n)

{

for(k in 1:n)

{

if(Ind1[i,j]>0 | Ind2[i,j]>0)

{

parti=(((i-1)*p+1):((i-1)*p+p))

partj=(((j-1)*p+1):((j-1)*p+p))

partk=(((k-1)*p+1):((k-1)*p+p))

Amat=A[partk, parti]

Bmat=t(B[partj, partk])
answer=Bmat%x%Amat

C=C+Ind1[i,j]*answer

D=D+Ind2[i,j]*answer

}

}

}

}

return(list(C=C, D=D))

}

solver=function(x){

#solver is a “matrix inverter” that can handle missing values

if(is.finite(sum(x)))

  {

  final.x <- solve(x)

  }

else{

keep.rows <- which(is.finite(rowMeans(x, na.rm = TRUE))) # Which rows are NOT all NA/Inf?

drop.rows <- which(!is.finite(rowMeans(x, na.rm = TRUE))) # Which rows are?

subset.x <- x[keep.rows, keep.rows] # Discard the rows with NA/Inf

inv.subset.x <- solve(subset.x)

inv.subset.vec <- as.vector(inv.subset.x)  # Break it down to a vector

final.x <- matrix(0, nrow = nrow(x), ncol = ncol(x))

# Populate the matrix with the elements of the inverted matrix:

k <- 1

for(j in keep.rows){

  for(i in keep.rows){

    final.x[i, j] <- inv.subset.vec[k]
    k <- k+1
  }
}
}
final.x
}
truncater=function (the_matrix)
{
#Truncater truncates covariance matrices
the_matrix=(t(the_matrix) + the_matrix)/2
p=nrow(the_matrix)
eig<-eigen(the_matrix) ; matrix_trun=0
for(i in 1:p)
matrix_trun<-matrix_trun+max(0, eig$values[i])*eig$vectors[,i]%*%t(eig$vectors[,i])
matrix_trun
}


#OAK DATA (used previously in Jackson et al 2015)


OAK=structure(list(d = c("BT", "BT", "BT", "BT", "AF", "AV", "BE",
"BE", "BE", "BE", "BE", "BE", "BE", "BV", "BV", "BV", "BV", "VI",
"HS", "BQ", "BJ", "BJ", "BJ", "AH", "AH", "AH", "AH", "AH", "AH",
"AH", "AH", "AH", "AH", "AH", "AH", "AH", "AH", "AG", "AG", "AG",
"AQ", "MD", "MD", "MD", "MD", "MD", "MD", "MD", "AD", "AD", "AD",
"AD", "AD", "AD", "BS", "BS", "BS", "BP", "BP", "BP", "AR", "CH",
"CH", "AK", "AK", "AK", "AL", "AL", "IH", "BH", "AE", "BI", "BU",
"VD", "BO", "AH", "ADH", "CVN", "ADM", "ADV", "GAL", "AHQ", "AGH",
"BIN", "COB", "BHV", "BDNV"), y = list(0.10789828, -0.18189296,
    0.61413822, -0.82603693, -0.14916168, -0.11419721, -1.4097723,
    -0.33240995, 0.06856271, -0.36464027, -0.75169228, -0.50324765,
    0.49719666, -0.2426402, -2.93518003, -0.63975477, -0.77748203,
    0.06109236, -1.19253525, -0.64570277, -0.8415101, 0.27921639,
    0.14999677, -0.19771863, -0.46940718, -0.95646705, -0.39313575,
    -0.20897436, -0.57880581, -0.02050818, -0.0678545, -0.44411973,
    -0.43769753, -0.65908944, -0.45814044, -0.43840988, -0.20789905,
    -0.51366242, -0.41349784, -4.28795429, -0.07497408, 0.05210705,
    -1.06152028, -0.53068704, -0.06273315, -0.28286799, -1.77579377,
    -0.77949258, -1.13556061, -0.75708101, 0.16615459, -1.69875885,
    0.16551812, -1.06159269, -0.20099705, -0.37731101, -0.69964482,
    0.35103883, -1.21532755, -0.07204235, 0.45557099, -2.44967895,
    -0.71066998, -0.31747873, -0.3271303, -0.19472813, -0.72629004,
    0.08966899, -0.80290605, -0.63473546, -1.01109974, -0.20138755,
    -0.8227403, 0.15080312, -0.928263, -0.17013936, c(-0.233571462,
    -0.025025514), c(-0.762857156, -0.433152233), c(-1.201925827,
    -0.684690514), c(-1.071267074, -1.313976021), c(-0.053579736,
    -0.177808846), c(-0.496204998, -0.267065054), c(-0.439873163,
    -0.321458035), c(0, -0.689955396), c(-1.84964968, -0.19611917
    ), c(0.267938107, -0.171802395), c(-1.19775, -0.2603804,
    -0.7811413)), s = list(0.0232674, 0.04839907, 0.06071628,
    0.10877984, 0.03430561, 0.07154501, 0.09614721, 0.0780995,
    0.09172009, 0.08170178, 0.15599594, 0.08253258, 0.1346979,
    0.10898273, 0.27169231, 0.07841072, 0.06908316, 0.10004665,
    0.0942214, 0.10213311, 0.14573089, 0.02595851, 0.06099994,
    0.05911098, 0.07779299, 0.06190854, 0.02386021, 0.0565442,
    0.19888567, 0.04349886, 0.20011511, 0.01123115, 0.1462782,
    0.0767004, 0.07894128, 0.03310878, 0.04103685, 0.04493278,
    0.13677825, 0.54134215, 0.06671351, 0.12921046, 0.05206926,
    0.04705471, 0.0727871, 0.10100018, 0.26737741, 0.21519022,
    0.06363698, 0.14288619, 0.10034509, 0.20238644, 0.11804995,
    0.01334994, 0.07862088, 0.21767532, 0.23581952, 0.0760269,
    0.10389677, 0.10006488, 0.12110769, 0.35594513, 0.05061601,
    0.02227884, 0.1023655, 0.09854855, 0.05777662, 0.01066845,
    0.08644658, 0.11056433, 0.05638952, 0.07885916, 0.1034265,
    0.28652648, 0.17041677, 0.17374879, structure(c(0.049754964,
    0.016393443, 0.016393443, 0.033062697), .Dim = c(2L, 2L)),
    structure(c(0.284852637, 0.083333333, 0.083333333, 0.263094018
    ), .Dim = c(2L, 2L)), structure(c(0.040793869, 0.014925373,
    0.014925373, 0.045750321), .Dim = c(2L, 2L)), structure(c(0.563756286,
    0.166666667, 0.166666667, 0.595918499), .Dim = c(2L, 2L)),
    structure(c(0.037832677, 0.0125, 0.0125, 0.037324696), .Dim = c(2L,
    2L)), structure(c(0.251905638, 0.066666667, 0.066666667,
    0.134522062), .Dim = c(2L, 2L)), structure(c(0.024831019,
    0.007874016, 0.007874016, 0.024362932), .Dim = c(2L, 2L)),
    structure(c(0.212669683, 0.076923077, 0.076923077, 0.231583482
    ), .Dim = c(2L, 2L)), structure(c(0.350038469, 0.142857143,
    0.142857143, 0.412386578), .Dim = c(2L, 2L)), structure(c(0.193227337,
    0.0625, 0.0625, 0.188114918), .Dim = c(2L, 2L)), structure(c(0.18278513,
    0.04, 0.04, 0.04, 0.16223164, 0.04, 0.04, 0.04, 0.1703878
    ), .Dim = c(3L, 3L)))), .Names = c("d", "y", "s"))
```

```
# Example data – the trivariate dataset in the main paper.
ms3v.data=structure(list(d=c("AB", "AB", "AC", "AC", "AC", "DC", "AD", "CB", "DB", "DB", "AEF", "AEF","CEF"),
y = list(c( -0.9942523 , -0.0833816 , 0 ),c( -0.8915981 , -0.4155155 , -0.4418328 ),
c( -0.4004776 , -0.3856625 , -0.6302778 ), c( -1.108663 , -0.3424903 , -0.3150811 ),
c( -1.514128 , -0.3856625 , -0.4624058 ),c( -0.3147107 , 0.0295588 , 0.3296765 ),
c( NA , -0.3424903 , -0.1690698 ),c( NA , -0.3424903 , -1.053661 ),
c( NA , 0.0582689 , 0.061369 ), c( NA , -0.0304592 , 0.120628 ),
c( -1.347074 , -0.7985077 , -0.3896016 , -1.347074 , -0.9162908 , -0.4670407 ),
c( -1.347074 , -0.644357 , -0.1872916 , -1.714798 , -0.6931472 , -0.3878512 ),
c( -0.4307829 , -0.7339692 , -0.3133934 , -0.5447271 , -0.4942963 , -0.1777004 )),
s = list(structure(c( 0.11977005129961, 0.0085905322892025, 0.01083667432968,
0.0085905322892025, 0.00985852395801 , 0.00310905104328 ,
0.01083667432968 , 0.00310905104328 , 0.1210483264 ), .Dim = c(3L, 3L)),
structure(c( 0.19288732907236 , 0.011587035122585 , 0.0141794739683064 ,
 0.011587035122585, 0.01113681306721 , 0.0034071302387916 ,
 0.0141794739683064 , 0.0034071302387916 , 0.12868606496656 ), .Dim = c(3L, 3L)),
structure(c( 0.02672943787396 , 0.00517419217634 , 0.0051437755823184 ,
0.00517419217634 , 0.01602563573776 , 0.0039828571779744 ,
0.0051437755823184 , 0.0039828571779744 , 0.12220505774656 ), .Dim = c(3L, 3L)),
structure(c( 0.017808635601 , 0.00280593203625 , 0.003212783473959 ,
0.00280593203625 , 0.007073651025 , 0.002024827118055 ,
0.003212783473959 , 0.002024827118055 , 0.07155619650001 ), .Dim = c(3L, 3L)),
structure(c( 0.023933637025 , 0.003307063035375 , 0.00376433837262 ,
0.003307063035375 , 0.00731132733969 , 0.0020805704158932 ,
0.00376433837262 , 0.0020805704158932 , 0.07309431331216 ), .Dim = c(3L, 3L)),
structure(c( 0.05082486186969 , 0.007815851274375 , 0.0049014512887698 ,
0.007815851274375 , 0.019230755625 , 0.00301498226595 ,
0.0049014512887698 , 0.00301498226595 , 0.05835635478436 ), .Dim = c(3L, 3L)),
structure(c( Inf , 0 , 0 , 0 , 0.01093473850249 , 0.0027620013646467 ,
0 , 0.0027620013646467 , 0.08612998213681 ), .Dim = c(3L, 3L)),
structure(c( Inf , 0 , 0 , 0 , 0.01822915823716 , 0.004551672243573 ,
0 , 0.004551672243573 , 0.14031055098025 ), .Dim = c(3L, 3L)),
structure(c( Inf , 0 , 0 , 0 , 0.00551624085796 , 0.001242093354894 ,
0 , 0.001242093354894 , 0.034528700761 ), .Dim = c(3L, 3L)),
structure(c( Inf , 0 , 0 , 0 , 0.00565176175524 , 0.0012529240911792 ,
0 , 0.0012529240911792 , 0.03429103982656 ), .Dim = c(3L, 3L)),
structure(c( 0.02776905624025 , 0.0040578586494875 , 0.002558991175299 , 0.00535175644249 ,
0.001601161732128 , 0.001254928240497 , 0.0040578586494875 , 0.00948751973521 ,
0.0014957691589962 , 0.001601161732128 , 0.00299401574976 , 0.0010637888279088 ,
0.002558991175299 , 0.0014957691589962 , 0.02911330012644 , 0.001254928240497 ,
0.0010637888279088 , 0.01307852980996 , 0.00535175644249 , 0.001601161732128 ,
 0.001254928240497 , 0.01971376059249 , 0.00355208168259 , 0.0021902640525666 ,
0.001601161732128 , 0.00299401574976 , 0.0010637888279088 , 0.00355208168259 ,
0.01024038754704 , 0.0015785921280024 , 0.001254928240497 , 0.0010637888279088 ,
0.01307852980996 , 0.0021902640525666 , 0.0015785921280024 , 0.03004266491524 ), .Dim = c(6L, 6L)),
structure(c( 0.04740313055076 , 0.00549393588549 , 0.0033057566969292 , 0.00966212327521 ,
0.002333120843004 , 0.0017245878726825 , 0.00549393588549 , 0.01018779347716 ,
0.0015325245514332 , 0.002333120843004 , 0.00352112878881 , 0.0011799066751785 ,
0.0033057566969292 , 0.0015325245514332 , 0.02846097213444 , 0.0017245878726825 ,
0.0011799066751785 , 0.01368092819025 , 0.00966212327521 , 0.002333120843004 ,
0.0017245878726825 , 0.05632681048929 , 0.0060151913481825 , 0.0036741280674186 ,
0.002333120843004 , 0.00352112878881 , 0.0011799066751785 ,  0.0060151913481825 ,
0.01027788412401 , 0.0015694539187482 , 0.0017245878726825 , 0.0011799066751785 ,
0.01368092819025 , 0.0036741280674186 , 0.0015694539187482 , 0.02958750890404 ), .Dim = c(6L, 6L)),
structure(c( 0.02381401426041 , 0.0056614646552475 , 0.0037749008591613 ,  0.01154601826576 ,
0.003606880499376 , 0.002878229119854 , 0.0056614646552475 , 0.02153500485361 ,
0.0035897295697407 , 0.003606880499376 ,  0.00704224750761 , 0.0025475490075903 ,
0.0037749008591613 , 0.0035897295697407 , 0.07387431588289 , 0.002878229119854 ,
0.0025475490075903 ,  0.03188863776121 , 0.01154601826576 , 0.003606880499376 ,
0.002878229119854 , 0.01926030474225 , 0.004775755917075 , 0.00330409329822 ,
0.003606880499376 , 0.00704224750761 , 0.0025475490075903 , 0.004775755917075 ,
0.01894702696324 , 0.003277111827816 , 0.002878229119854 , 0.0025475490075903 ,
0.03188863776121 , 0.00330409329822 , 0.003277111827816 , 0.069977179024 ), .Dim = c(6L, 6L)))),
.Names = c("d", "y", "s"))


require(MASS)

require(Matrix)

A=fn_mv(ms3v.data)

A

# TRIVARIATE RESULTS AS GIVEN IN THE MAIN PAPER.

B=fn_mv(OAK)

B

#OAK RESULTS (A UNIVARIATE EXAMPLE) AS GIVEN PREVIOUSLY BY JACKSON ET AL (2016)
```