On Using Summary Statistics from an External Calibration Sample to Correct for Covariate Measurement Error

Supplementary Materials

This supplementary material presents the R source code to perform the MI-EC algorithm we describe in the main article.

```
#Calculate the summary statistics\\
calsummstat <-function(inputdata){

    x=inputdata[,1]
    w=inputdata[,2]

    n=nrow(inputdata)

    xbar=mean(x)
    wbar=mean(w)
    xx=sum(x^2)/n
    ww=sum(w^2)/n
    xy=sum(x*w)/n

    sxx=xx-xbar*xbar
    sww=ww-wbar*wbar
    sxy=xy-xbar*wbar

    b0=xbar-(sxy/sww)*wbar
    b1=sxy/sww
    sigmasq=sxx-sxy*sxy/sww

    param=c(b0,b1,sigmasq,sxx,sww,sxy,xbar,wbar)
    return(param)

}

#Draw parameters from their predictive distribution based on the
calibration data \\
generateRandomdDrawofMEMParam <- function(calibdata,n) {
```

```
    ndraw=1

    #Calculate the summary statistics of X and W
    iniparam=calsummstat(calibdata)

    betahat0=iniparam[1]
    betahat1=iniparam[2]
    rss=n*iniparam[3]
    sxx=iniparam[4]

    sigmasq=rss/rchisq(ndraw,(n-2))

    tmp0=rnorm(1)
    beta0=betahat0 + sqrt(sigmasq/n)*tmp0

    tmp1=rnorm(1)
    beta1=betahat1 + sqrt(sigmasq/(n*sxx))*tmp1

    param=c(beta0, beta1, sigmasq)
    return(param)
}

#Draw parameters from their predictive distribution based on the
main study data. \\
generateRandomMultivarRegreParam <- function(maindata,n,p) {

    ndraw=1

    w=maindata[,1]

    wmat=mat.or.vec(n,2)
    wmat[,1]=1
    wmat[,2]=w

    umat=maindata[,2:(p+1)]

    ww=solve(t(wmat)%*%wmat)
    coeffhat=ww%*%(t(wmat)%*%umat)
    residual=umat-wmat%*%coeffhat
    rss=t(residual)%*%residual
    a=t(chol(ww))
    invrss=solve(rss) #calculate inverse covariance matrix

    #Draw covariance of U given W from inverse
    #wishart distribution with df = (n-(k+p)+1)
```

```
    #where k is dimension of wmat and p is dimension of umat
    df=n-(p+2)+1
    covmatrix=riwish(df, rss)
    covmatrix=as.matrix(covmatrix)

    betavar= kronecker(covmatrix, ww)
    vec.coeffhat=as.vector(coeffhat)
    beta = mvrnorm(ndraw, vec.coeffhat, betavar)

    #regression coefficients of U on W
    multivarRegrecoeff=matrix(beta,2,p)

    #residual covariance matrix of U given W
    multivarResidcov=covmatrix

    param=rbind(multivarRegrecoeff,multivarResidcov)
    return(param)
}

#generate parameter of the distribution of surrogate W from the
posterior distribution \\
generateErrorvarParam <- function(calibdata,maindata,NCALIB,NSAMPLE)
{

    #combine the information about W from calibration data
    #and main study data
    w=c(calibdata[,2],maindata[,1])

    n=NCALIB+NSAMPLE

    ndraw = 1

    #Draw sigmaxsq from inverse chi-square distribution
    #with df = (n-2)
    muw = mean(w)
    rss = sum((w-muw)^2)
    sigmaxsq = rss/rchisq(ndraw,(n-1))

    #Draw mux from normal distribution
    tmp = rnorm(1)
    mu = muw + sqrt(sigmaxsq/n)*tmp

    param = c(mu, sigmaxsq) # two parameters: mean and variance

    return(param)
```

```
}

#create the initinal mean and covariance matrix \\
createMatrixonW <- function(memParam, dmParam,wParam, P){

    m = P+2+1
    matrixonW=matrix(NA, nrow = m, ncol = m)

    matrixonW[1,1] = -(1 + wParam[1]^2/wParam[2])
    matrixonW[1,2] = wParam[1]/wParam[2]
    matrixonW[2,2] = - 1/wParam[2]

    matrixonW[1:(m-1),3:(m-1)]=dmParam

    matrixonW[1:2,m] = memParam[1:2]
    matrixonW[m,m] = memParam[3]

    matrixonW[3:(m-1),m] = dmParam[2,] * memParam[3] / memParam[2]

    return(matrixonW)
}

#complete initinal mean and covariance matrix \\
completeGmatrix <- function(G){

    size = nrow(G)

    for (i in 1:size){
        for (j in 1:size){
            if ( is.na(G[i,j]) ) {
                if ( is.na(G[j,i]) == FALSE ) {
                    G[i,j] = G[j,i]
                } else {
                    print(sprintf("ERROR: both elements at [%d,%d]
                    and [%d,%d] are null", i, j, j, i));
                }
            }
        }
    }

    if (isSymmetric(G)) {
        return(G)
    } else {
        return(NULL)
    }
```

```
}

#check the symmetry of the matrix created by the sweep operator
isSymmetric <- function(G){

    size = nrow(G)

    for (i in 1:size){
        for (j in 1:size){
            if ( abs(G[i, j] - G[j, i]) > 1e-10) {
                print(sprintf("ERROR: elements not matched at [%d,%d]
                and [%d,%d]", i, j, j, i));
                return(FALSE);
            }
        }
    }
    return(TRUE);
}

#perform the sweep operator \\
sweep <- function(matrixonW){

    size = nrow(matrixonW)

    curH = completeGmatrix(matrixonW)
    newH= matrix(NA, nrow=size, ncol=size)

    for (k in 3:(size-1)){

        for (i in 1:size){
            for (j in 1:size){

                if (i==k && j==k)   {
                    newH[i,j] = -1/curH[k,k]
                } else if (i==k || j==k) {
                    newH[i,j] = curH[i,j]/curH[k,k]
                } else {
                    newH[i,j] = curH[i,j]
                      - curH[i,k]*curH[k,j]/curH[k,k]
                }
            }
        }

        curH = newH;
        newH= matrix(NA, nrow=size, ncol=size)
```

```
    }

    param=curH[,size]
    return(param)
}

#Create imputed values for unobserved covariate X
generateMissingvalue <- function(param, maindata){

    nsample=nrow(maindata)

    nparam=length(param)

    #test whether the estimate of the residual variance is negative,
    #and print a warning message
    if (param[nparam] < 0) {

        print(sprintf("%s", "Warning: The estimate of the
        residual variance of mismeaured covariate given
        the observed data is negative."))
        }

    mux = param[1]
    for (i in 2:(nparam-1)){

        mux = mux + param[i]*maindata[,i-1]

    }

    #Generate X from its posterior distribution with mean mux
    #and variance sigmasqxx_wzy
    imputedX = mux + sqrt(param[nparam])*rnorm(nsample, mean=0, sd=1)

    return(imputedX)
}

#title function \\
printTitle <-function(){

    print(sprintf("%s", "########################################"))
    print(sprintf("%s", "## Loading required package: MIEC  ####"))
    print(sprintf("%s", "########################################"))
}

#main function \\
```

```
MIEC <- function(maindata,calibdata,NCALIB,NSAMPLE,M,N,K,S) {

    printTitle()

    MIimputedXbasedoncalib=c()
    multipleImputedX=c()
    twostageMIimputedX=c()

    P=K+S

    count = 0

    while (count < M) {

        #Step-1: draw parameters by regressing X on W based on
        #measurement error model
        memParam=generateRandomdDrawofMEMParam(calibdata,NCALIB)

        #Step-2: draw parameters by regressing (Y, Z) on W based on main
        #interested "disease" model
        dmParam=generateRandomMultivarRegreParam(maindata,NSAMPLE,P)

        #Step-3: generate draws of mean and variance of W
        wParam=generateErrorvarParam(calibdata,maindata,NCALIB,NSAMPLE)

        #Step-4: creating sweeping matrix on W by using parameters
        #obtained from step 1-3 and filling estimated covariance
        #parameter between U and X given W
        sweepMatrixonW=createMatrixonW(memParam, dmParam,wParam, P)

        #Step-5: calculate parameters of the imputation model for X
        #given (W,Y,Z) by the sweep operator
        impmodelParam = sweep(sweepMatrixonW)

        #Step-6: generate random draw for unknown X from its posterior
        #distribution, given W and Y
        varIndicator=length(impmodelParam)

        if (impmodelParam[varIndicator] >= 0){

            for (n in 1:N){

                secondStageDrawX = generateMissingvalue(impmodelParam,
                maindata)
```

```
                twostageMIimputedX=cbind(twostageMIimputedX,
                secondStageDrawX)

            }
            count=count+1
        }
    }

    #Output data with Y, Z, and multiply imputed X,
    #where maindata[,P+1] = U(Y,Z)
    twoStageMIimputeddata=cbind(maindata[,2:(P+1)], twostageMIimputedX)
    return(twoStageMIimputeddata)
}
```