***Multivariate Behavioral Research* Supplemental material**

**A Bayesian power analysis procedure considering uncertainty in**

**effect size estimates from a meta-analysis**

**Han Du & Lijuan Wang**

**R code for the pas function**

pas= function(effectsize,nactual,nplan,random=TRUE,cpower=0.8,nrep=10,type,adjust=FALSE,adaptstep=500,burninstep=1000,mcstep=50000,nchain=3,thinstep=1,M=40000)

# "effectsize" is a vector containing the observed effect sizes from a meta-analysis.

# "nactual" is a vector containing actual sample sizes of the studies included in the meta-analysis

# or a matrix containing actual group sample sizes of the studies included in the meta-analysis .

# "nplan" is a scalar for the planned sample size or a vector containing the planned group sample sizes.

# "random" indicates whether a random-effects model or a fixed-effects model is used.

# "cpower" specifies the target power level.

# "nrep" specifies the number of replications for calculating the average assurance level

# "type" specifies the effect size type with type='d' for testing standardized group mean differences and type='r' for testing Pearson's correlations.

# "adjust" indicates whether adjustments are needed for obtained unbiased standardized mean differences

# when type ='r', "nactual" is a vector and "nplan" is is a scalar.

# when type ='d', "nactual" is a matrix and "nplan" is is a vector.

# If the g values instead of d values are read in for "effectsize", adjust = T; the default is adjust =F.

# "adaptstep" specifies the number of iterations for adaptation.

# "burninstep" specifies the number of burn-in iterations

# "mcstep" specifies the number of iterations to monitor after burn-in and thinning

# "nchain" specifies the number of parallel MCMC chains

# "thinstep" specifies the thinning interval for monitors

# "M" specifies the number of sets of drawn parameter values from the posterior distribution with the post-burn-in iterations

{

mcmc\_dr = function( d, n1,n2,mcstep = mcstep, adaptstep = adaptstep, burninstep = burninstep,nchain = nchain,thinstep = thinstep) {

require(rjags)

#mcmc model

model= "

model {

for (i in 1:Ntotal){

d[i] ~ dnorm( delta[i], 1/( (n1[i]+n2[i])/(n1[i]\*n2[i]) + delta[i]^2/(2\*(n1[i]+n2[i]) ) ) )

delta[i] ~ dnorm( mu\_delta, 1/tau^2)

}

mu\_delta ~ dnorm(0, 1/10000)

tau ~ dunif(0, 100)

}

"

writeLines(model, con="model\_dr.txt" )

#data

dataList = list(

d = d,

n1 = n1,

n2 = n2,

Ntotal = length(d)

)

# Initialize the chains

parameters = c( "mu\_delta", "tau") # The model parameters

nIter = ceiling( ( mcstep \* thinstep ) / nchain )

jagsModel = jags.model( "model\_dr.txt" , data=dataList , n.chains=nchain , n.adapt=adaptstep )

cat( "Burning in the MCMC chain...\n" )

update( jagsModel , n.iter=burninstep)

cat( "Sampling final MCMC chain...\n" )

codaSamples = coda.samples( jagsModel , variable.names=parameters, n.iter=nIter , thin=thinstep )

mcmcChain = as.matrix( codaSamples )

return( mcmcChain[1:mcstep,])

} #get posterior distribution

#------------------------------------------------------------------------------

mcmc\_df = function( d, n1,n2,mcstep = mcstep, adaptstep = adaptstep, burninstep = burninstep,nchain = nchain,thinstep = thinstep) {

require(rjags)

#mcmc model

modelString = "

model {

for (i in 1:Ntotal){

d[i] ~ dnorm( mu\_delta,1/( (n1[i]+n2[i])/(n1[i]\*n2[i])+ mu\_delta^2/(2\*(n1[i]+n2[i]) ) ) )

}

mu\_delta ~ dnorm(0,1/10000)

}

"

writeLines( modelString , con="model\_df.txt" )

# data:

dataList = list(

d = d,

n1 = n1,

n2 = n2,

Ntotal = length(d)

)

# Initialize the chain

parameters = c( "mu\_delta") # The model parameters

nIter = ceiling( ( mcstep \* thinstep ) / nchain )

jagsModel = jags.model( "model\_df.txt" , data=dataList , n.chains=nchain , n.adapt=adaptstep )

cat( "Burning in the MCMC chain...\n" )

update( jagsModel , n.iter=burninstep)

cat( "Sampling final MCMC chain...\n" )

codaSamples = coda.samples( jagsModel , variable.names=parameters , n.iter=nIter , thin=thinstep )

mcmcChain = as.matrix( codaSamples )

return( mcmcChain[1:mcstep,])

} #get posterior distribution

#------------------------------------------------------------------------------

mcmc\_zr = function( r, n,mcstep = mcstep, adaptstep = adaptstep, burninstep = burninstep,nchain = nchain,thinstep = thinstep) {

require(rjags)

# mcmc model

modelString = "

model {

for ( i in 1:Ntotal ) {

z\_r[i] ~ dnorm( z\_rho[i],1/v[i])

z\_rho[i] ~ dnorm( mu\_zrho, 1/tau^2)

}

mu\_zrho ~ dnorm(0,1/10000)

tau ~ dunif(0, 100)

}

"

writeLines( modelString , con="model\_zr.txt" )

# data:

v <- 1/(n-3)

z\_r <- 1/2\*log( (1+r)/(1-r) )

dataList = list(

z\_r = z\_r,

v = v,

Ntotal = length(z\_r)

)

# Initialize the chain

parameters = c( "mu\_zrho", "tau") # The model parameters

nIter = ceiling( ( mcstep \* thinstep ) / nchain )

jagsModel = jags.model( "model\_zr.txt" , data=dataList, n.chains=nchain , n.adapt=adaptstep )

cat( "Burning in the MCMC chain...\n" )

update( jagsModel , n.iter=burninstep)

cat( "Sampling final MCMC chain...\n" )

codaSamples = coda.samples( jagsModel , variable.names=parameters ,n.iter=nIter , thin=thinstep )

mcmcChain = as.matrix( codaSamples )

return( mcmcChain[1:mcstep,])

} #get posterior distribution

#------------------------------------------------------------------------------

mcmc\_zf = function( r, n,mcstep = mcstep, adaptstep = adaptstep, burninstep = burninstep,nchain = nchain,thinstep = thinstep) {

require(rjags)

# mcmc model

modelString = "

model {

for ( i in 1:Ntotal ) {

z\_r[i] ~ dnorm(mu\_zrho, 1/v[i])

}

mu\_zrho ~ dnorm(0,1/10000)

}

"

# save model

writeLines( modelString , con="model\_zf.txt" )

# data:

v <- 1/(n-3)

z\_r <- 1/2\*log( (1+r)/(1-r) )

dataList = list(

z\_r = z\_r,

v = v,

Ntotal = length(z\_r)

)

# Initialize the chain

parameters = c( "mu\_zrho") # The model parameters

nIter = ceiling( ( mcstep \* thinstep ) / nchain )

jagsModel = jags.model( "model\_zf.txt" , data=dataList, n.chains=nchain , n.adapt=adaptstep )

cat( "Burning in the MCMC chain...\n" )

update( jagsModel , n.iter=burninstep)

cat( "Sampling final MCMC chain...\n" )

codaSamples = coda.samples( jagsModel , variable.names=parameters ,n.iter=nIter , thin=thinstep )

mcmcChain = as.matrix( codaSamples )

return( mcmcChain[1:mcstep,])

} #get posterior distribution

#------------------------------------------------------------------------------

EBpower\_dr = function( mcmcChain\_sd , nplan1, nplan2 ,N,nmcmc = M) {

chainLength = NROW( mcmcChain\_sd)

stepIdxVec = sample(chainLength , nmcmc)

nSim = 0

power <- matrix(data=NA, nrow=length(stepIdxVec), ncol=N)

for ( stepIdx in stepIdxVec ) {

nSim = nSim + 1

# Get parameter values for this simulation:

mu\_delta <- mcmcChain\_sd[stepIdx,1]

tau <- mcmcChain\_sd[stepIdx,2]

#Simulate effect sizes

delta <- rnorm( N, mean=mu\_delta, sd=tau)

lamda <- delta\*sqrt( nplan1\*nplan2/(nplan1+nplan2) )

#Calculate power

for (i in 1:N){

ncp <-lamda[i]

t <- qt(0.975,df=nplan1+nplan2-2)

power[nSim,i] <- 1-pt(t,df=nplan1+nplan2-2,ncp=ncp)+pt(-t,df=nplan1+nplan2-2,ncp=ncp)

}

}

return( power)

} # end of function power

##------------------------------------------------------------------------------

EBpower\_df = function( mcmcChain\_sd , nplan1, nplan2 ,N,nmcmc = M) {

chainLength = NROW( mcmcChain\_sd)

stepIdxVec = sample(chainLength-N+1 , nmcmc)

nSim = 0

power <- matrix(data=NA,nrow=length(stepIdxVec),ncol=N)

for ( stepIdx in stepIdxVec ) {

nSim = nSim + 1

# Get parameter values for this simulation and simulate effect sizes

mu\_delta <- mcmcChain\_sd[stepIdx:(stepIdx-1+N)]

lamda <- mu\_delta\*sqrt( nplan1\*nplan2/(nplan1+nplan2) )

#Calculate power

for (i in 1:N){

ncp <- lamda[i]

t <- qt( 0.975, df=nplan1+nplan2-2)

power[nSim,i] <- 1-pt(t,df=nplan1+nplan2-2,ncp=ncp)+pt(-t,df=nplan1+nplan2-2,ncp=ncp)

}

}

return( power)

} # end of function power

#------------------------------------------------------------------------------

EBpower\_zr = function( mcmcChain\_sd, nplan, N,nmcmc=M ) {

chainLength = NROW( mcmcChain\_sd)

stepIdxVec = sample(chainLength , nmcmc)

nSim = 0

power <- matrix(data=NA, nrow=length(stepIdxVec), ncol=N)

for ( stepIdx in stepIdxVec ) {

nSim = nSim + 1

# Get parameter values for this simulation:

mu\_zrho <- mcmcChain\_sd[stepIdx,1]

tau <- mcmcChain\_sd[stepIdx,2]

# Simulate effect sizes

z\_rho <- rnorm( N, mean=mu\_zrho, sd=tau)

v <- 1/(nplan-3)

# Calculate power

lamda <- z\_rho/sqrt(v)

for (i in 1:N){

power[nSim,i] <- 1-pnorm(1.96-lamda[i])+pnorm(-1.96-lamda[i])

}

}

return( power )

} # end of function power

#------------------------------------------------------------------------------

EBpower\_zf = function( mcmcChain\_sd, nplan, N,nmcmc = M ) {

chainLength = NROW( mcmcChain\_sd)

stepIdxVec = sample(chainLength-N+1 , nmcmc)

nSim = 0

power <- matrix(data=NA, nrow=length(stepIdxVec), ncol=N)

for ( stepIdx in stepIdxVec ) {

nSim = nSim + 1

# Get parameter values for this simulation and simulate effect sizes

mu\_zrho <- mcmcChain\_sd[stepIdx:(stepIdx-1+N)]

v <- 1/(nplan-3)

# calculate power

lamda <- mu\_zrho/sqrt(v)

for (i in 1:N){

power[nSim,i] <- 1-pnorm(1.96-lamda[i])+pnorm(-1.96-lamda[i])

}

}

return( power )

} # end of function power

#------------------------------------------------------------------------------

#Function part ends

#------------------------------------------------------------------------------

if (type=="d"){

# Get posterior distribution

n1 <- nactual[,1]

n2 <- nactual[,2]

d <- effectsize

if (adjust=="TRUE"){

d <- ( 1-3/(4\*(n1+n2-2)-1) )\*d

}

d <- d

if (random==TRUE){

mcmcChainpro1 = mcmc\_dr(d, nactual[,1],nactual[,2], mcstep = mcstep, adaptstep = adaptstep, burninstep = burninstep,nchain = nchain,thinstep = thinstep)

a <- c()

e <- c()

# Get power distribution

for (i in 1:nrep){

powerPro1 = EBpower\_dr(

# posterior distribution of parameters

mcmcChainpro1,

# planned sample sizes

nplan1 = nplan[1],

nplan2 = nplan[2],

# Number of simulations to run

nmcmc = M,

N = length(effectsize)

)

a[i] = sum(powerPro1>=cpower)/length(powerPro1)

e[i] = mean(powerPro1)

}

}else{

mcmcChainpro2 = mcmc\_df( d, nactual[,1],nactual[,2], mcstep = mcstep\*length(d), adaptstep = adaptstep, burninstep = burninstep,nchain = nchain,thinstep = thinstep)

a<-c()

e<-c()

# Get power distribution

for (i in 1:nrep){

powerPro2 = EBpower\_df(

# posterior distribution of parameters

mcmcChainpro2,

# planned sample sizes

nplan1 = nplan[1],

nplan2 = nplan[2],

# Number of simulations to run

nmcmc = M,

N = length(effectsize)

)

a[i] = sum(powerPro2>=cpower)/length(powerPro2)

e[i] = mean(powerPro2)

}

}

} else if (type=="r"){

n <- nactual

r <- effectsize

z <- 1/2\*log((1+r)/(1-r))

if (random==TRUE){

mcmcChainpro3 = mcmc\_zr(z,nactual,mcstep = mcstep, adaptstep = adaptstep, burninstep = burninstep,nchain = nchain,thinstep = thinstep)

a<-c()

e<-c()

# Get power distribution

for (i in 1:nrep){

powerPro3 = EBpower\_zr(

# posterior distribution of parameters

mcmcChainpro3,

# planned sample size

nplan = nplan,

# Number of simulations to run

nmcmc = M,

N = length(effectsize)

)

a[i] = sum(powerPro3>=cpower)/length(powerPro3)

e[i] = mean(powerPro3)

}

}else{

mcmcChainpro4 = mcmc\_zf(z,nactual, mcstep = mcstep\*length(z), adaptstep = adaptstep, burninstep = burninstep,nchain = nchain,thinstep = thinstep)

a<-c()

e<-c()

# Get power distribution

for (i in 1:nrep){

powerPro4 = EBpower\_zf(

# posterior distribution of parameters

mcmcChainpro4,

# planned sample size

nplan = nplan,

# Number of simulations to run

nmcmc = M,

N = length(effectsize)

)

a[i] = sum(powerPro4>=cpower)/length(powerPro4)

e[i] = mean(powerPro4)

}

}

}

as <- sum(a)/nrep

ep <- sum(e)/nrep

cat( " Assurance is ", as, "\n", "\n", "Expected power is ", ep, "\n","\n")

return(c(as,ep))

}

## A hypothetical example for testing Cohen's ds

# actual sample sizes of group 1

n1 <- c(15,20,30,18,20)

# actual sample sizes of group 2

n2 <- c(15,20,30,18,20)

# combine the group sizes together

n <- cbind(n1,n2)

# Observed effect sizes: hedge'g values

g <- c(0.5,0.72,0.63,0.43,0.24)

# planned sample size

nplan <- c(60,60)

assurance <- pas(effectsize=g,nactual=n,random=T,nplan=nplan,cpower=0.8,type="d",nrep=10,adjust=T)

## A hypothetical example for testing Pearson's correlations

# actual sample sizes of the studies

n <- c(30,40,50,60,70)

# Observed Pearson's correlations

r <- c(0.21,0.25,0.4,0.43,0.45)

# planned sample size

nplan <- 32

assurance <- pas(effectsize=r,nactual=n,random=T,nplan=nplan,cpower=0.8,type="r",nrep=10)