Description of R code and methods

Dr Jon Minton

Thursday, December 04, 2014

# Introduction

This appendix describes the R code used to generate the various models compared in our paper.

# Helper and support functions

This section presents code that performs ancillary (helper) functions for the generation of the methods below.

## Summarise IPD

This code takes a dataframe of IPD and creates summary statistics based on it: the mean and the standard error, based on the standard deviation and the sample size. This is the only information available to all of the method except the bootstrapped IPD-based gold standard.

summarise\_ipd <- function(  
 data,  
 n\_dp=3  
 ){  
 n\_vars <- dim(data)[2]  
 n\_obs <- dim(data)[1]  
 output <- vector("list", length=n\_vars)  
 names(output) <- colnames(data)  
   
 for (i in 1:n\_vars){  
 mu.this <- round(mean(data[,i]), n\_dp)  
 upper.this <- mean(data[,i]) + 1.96\*sd(data[,i])/sqrt(n\_obs)  
 se.this <- round(  
 (upper.this - mean(data[,i]))/1.96,  
 3)  
   
 list.this <- list(  
 mu=mu.this,  
 se=se.this  
 )  
 output[[i]] <- list.this  
 }  
 return(output)   
}

## Bootstrapped IPD

This function takes IPD and estimates the joint means of the data through a bootstrapping procedure. This is considered the 'gold standard' within the evalution, against which the other methods are compared.

bootstrap\_means\_ipd <- function(  
 data,  
 n\_reps  
 ){  
 n\_vars <- dim(data)[2]  
 n\_obs <- dim(data)[1]  
 draws <- 1:n\_reps  
 output <- matrix(NA, nrow=n\_reps, ncol=n\_vars)  
 for (i in 1:n\_reps){  
 boot.this <- data[sample(1:n\_obs, n\_obs, T),]  
 for (j in 1:n\_vars){  
 output[i,j] <- mean(boot.this[,j])  
 }  
 }  
 colnames(output) <- colnames(data)  
 output <- as.data.frame(output)  
 return(output)  
}

## Beta parameters from reported mean and variance

This function returns the beta parameters associated with a given mean and variance

est\_beta <- function(mu, var) {  
 a <- mu \* ((1 - mu) \* (mu / var) - 1)  
 b <- a \* ((1 - mu) / mu)  
 return(list(a=a, b=b))  
}

## Difference parameter beta calculation

This function identifies the beta distribution parameters for the delta distribution used in the difference method

get\_dif\_param <- function(  
 u1\_mu, u1\_sd,   
 u2\_mu, u2\_sd,   
 quietly=T  
 ){  
 mu <- u1\_mu - u2\_mu  
   
 sigma2 <- ifelse(u1\_sd > u2\_sd, u1\_sd^2 - u2\_sd^2, u2\_sd^2 - u1\_sd^2)  
 x <- (1 - mu) / mu  
   
 a <- (x/sigma2-1-2\*x-x^2)/(1+3\*x+3\*x^2+x^3)  
 b<-a\*x  
   
 if(quietly==F){  
 print(a/(a+b)) # check mean of delta  
 print(a\*b/(a+b)^2/(a+b+1)) # check variance of delta   
 }  
 return(list(a=a, b=b))  
}

## Average of the individual variances (AIVM) method

This method takes as its inputs the mean and SDs of two distributions and returns joint PSA estimates drawn from a bivariate normal distribution that has a covariance equal to the average of the variances of the two input distributions

make\_aivm\_cov\_2d <- function(  
 mu\_x, sd\_x,   
 mu\_y, sd\_y,   
 colnames\_,  
 n\_psa\_=n\_psa  
 ){  
   
 var\_x <- sd\_x^2  
 var\_y <- sd\_y^2  
   
 aivm <- min(  
 mean(  
 c(var\_x, var\_y)  
 ),  
 sd\_x \* sd\_y)  
   
 if (THROW\_FACTS) {sponge$aivm <<- aivm}  
   
 sig <- matrix(  
 data=c(  
 var\_x, aivm,   
 aivm, var\_y  
 ),   
 nrow=2, byrow=T  
 )  
   
 aivm\_samples <- mvrnorm(  
 n=n\_psa\_,   
 mu=c(mu\_x, mu\_y),   
 Sigma=sig   
 )  
   
 colnames(aivm\_samples) <- colnames\_  
 aivm\_samples <- as.data.frame(aivm\_samples)  
 return(  
 list(  
 aivm\_samples=aivm\_samples,   
 aivm=aivm)  
 )  
}

## Covariance fitting methods

This function is used for both the methods 'Covariance Fitting (Lower Bounded)' and 'Covariance Fitting (Upper bounded)'. The input argument 'upper' is used to determin which of the two methods will be run. If the option quietly is set to FALSE then it also reports details of the covariances that it tries against the criterion. The criterion is that no pair of estimates should violate the monotonicity assumption. Depending on the value of a global variable THROW\_FACTS further details are written to a list object called sponge. The inc\_by option can also be adjusted to provide a finer or coarser grained searching of the search space.

make\_bcvr\_2d <- function(  
 mu\_x, sd\_x,   
 mu\_y, sd\_y,   
 n\_psa\_,   
 inc\_by=0.00001,   
 colnames\_,  
 upper=T,  
 quietly=T  
 ){   
 if (!quietly){  
 print("make\_bcvr\_2d entered")  
 cat("upper is set to ", upper, "\n")  
 }  
   
 var\_x <- sd\_x^2 # variance of X  
 var\_y <- sd\_y^2 # variance of Y  
   
 if (!quietly){  
 cat("var\_x is ", var\_x, " and var\_y is ", var\_y, "\n")  
 }  
   
 if(upper==T){  
 lowerbound <- 0 # start assuming independent  
 upperbound <- min(sd\_x \* sd\_y,  
 mean(var\_y, var\_y)  
 ) # upper bounds are the minimum of the AIVM or the cov which implies a cor > 1  
   
 if (!quietly){  
 cat("upper bounded. Looking for values between ", lowerbound, " and ", upperbound, "\n")  
 browser()  
 }  
   
 } else {  
 lowerbound <- mean(var\_x, var\_y)  
 upperbound <- sd\_x \* sd\_y # don't select a covariance which implies a correlation > 1  
   
 if (!quietly){  
 cat("lower bounded. Looking for values between ", lowerbound, "and ", upperbound, "\n")  
 browser()  
 }  
 }  
   
   
 cov.this <- lowerbound  
  
 mus <- c(mu\_x, mu\_y)  
 search <- T  
   
 if(cov.this <= upperbound){ # if the maximum value's been reached already  
   
 cat("Upperbound already reached\n")  
 search <- F # if the upper limit's already been reached, go no further  
 if (THROW\_FACTS){sponge$upperbound\_reached <<- T}  
   
 testsig <- matrix(  
 c(  
 var\_x, cov.this,   
 cov.this, var\_y  
 ), nrow=2, byrow=T  
 )  
   
 testsamples <- mvrnorm(n\_psa\_, mu=mus, Sigma=testsig)  
 } else {  
 cat("Upperbound not yet reached\n")  
 this.cov <- lowerbound  
 cat("This covariance: ", cov.this, "\n", sep="")  
 testsig <- matrix(  
 c(  
 var\_x, cov.this,   
 cov.this, var\_y  
 ), nrow=2, byrow=T  
 )  
   
 testsamples <- mvrnorm(n\_psa\_, mu=mus, Sigma=testsig)  
 }  
   
 while(search==T){  
 cat("trying ", cov.this, "\n")  
 testsig <- matrix(  
 c(  
 var\_x, cov.this,   
 cov.this, var\_y  
 ), nrow=2, byrow=T  
 )  
   
 try\_testsamples <- try(mvrnorm(n\_psa\_, mu=mus, Sigma=testsig))  
 if(class(try\_testsamples)=="try-error"){ # if mvrnorm has been passed impossible values  
 search <- F  
 cat("Error picked up\n")  
   
 } else {  
 cat("No error in mvrnorm args\n")  
 testsamples <- try\_testsamples # if the attempted values are correct, use them  
 if (any(testsamples[,1] < testsamples[,2])){  
 cat("Violation with ", cov.this, "\n")  
 cov.this <- cov.this + inc\_by # increment the values by a little bit  
 cat("Trying ", cov.this, "\n")  
 } else {  
 cat("Found ", cov.this, "\n")  
 search <- F  
 }  
 }  
 }  
 if (THROW\_FACTS) {   
 if (exists("sponge$cov\_found")){  
 sponge$cov\_found <<- c(sponge$cov\_found, cov.this)  
 } else {  
 sponge$cov\_found <<- cov.this  
 }  
 }  
 cor.this <- cov.this / (sd\_x \* sd\_y)  
 if (THROW\_FACTS) {  
 if (exists("sponge$cor\_found")){  
 sponge$cor\_found <<- c(sponge$cor\_found, cor.this)  
 } else {  
 sponge$cor\_found <<- cor.this  
 }  
 }  
 colnames(testsamples)=colnames\_  
 return(  
 list(cov=cov.this,   
 samples=testsamples,   
 cor=cor.this)  
 )  
}

# The models

This section shows the code used to generate the models that are being compared in the paper.

## Coordinating function

Each of the methods is run through a coordinating function, create\_draws, which takes as its input a list object, summary\_data, and method, a number referring to the method to be run. It then returns as an output a list of dataframes, each dataframe containing the joint draws generated by the specified method.

The structure of this coordinating function is shown below. The contents of each of the method subsections is presented separately later in the document

create\_draws <- function(  
 summary\_data,  
 method,  
 n\_psa=1000,  
 seed=80,  
 quietly=F  
 ){  
 # summary\_data should be a list  
 # The top level should be the number of variables to estimate  
 n\_vars <- length(summary\_data)  
 output <- matrix(NA, nrow=n\_psa, ncol=n\_vars)  
 colnames(output) <- names(summary\_data)  
   
 if (method==1){## See method 1 code  
 }  
 if (method==2){## see method 2 code  
 }  
 if (method==3){## See method 3 code  
 }  
 if (method==4){## See method 4 code  
 }   
 if (method==5){## See method 5 code  
 }  
 if (method==6){## See method 6 code  
 }  
 if (method==7){## See method 7 code  
 }  
 if (method==8){## See method 8 code   
 }  
 if (method==9){## See method 9 code  
 }  
 if (method==10){## See method 10 code   
 }  
 if (method == 11){## See method 11 code  
 }  
  
 ###  
 return(output)  
}

## Code within the coordinating function

### Method 1: independent samplign

## Method 1 : Independent Sampling (Naive)  
for (i in 1:n\_vars){  
 params.this <- est\_beta(  
 summary\_data[[i]]$mu,  
 summary\_data[[i]]$se^2  
 )  
 draws.this <- rbeta(  
 n\_psa,  
 params.this$a,  
 params.this$b  
 )  
 output[,i] <- draws.this  
}

### Method 2: quantile matching

## Method 2 : Quantile matching/same random number seed   
seeds <- runif(n\_psa)  
for (i in 1:n\_vars){  
 params.this <- est\_beta(  
 summary\_data[[i]]$mu,  
 summary\_data[[i]]$se^2  
 )   
 draws.this <- qbeta(  
 seeds,  
 params.this$a,  
 params.this$b  
 )  
 output[,i] <- draws.this  
}

### Method 3: upward replacement

## Method 3 : Upward Replacement  
for (i in 1:n\_vars){  
 params.this <- est\_beta(  
 summary\_data[[i]]$mu,  
 summary\_data[[i]]$se^2  
 )  
 draws.this <- rbeta(  
 n\_psa,  
 params.this$a,  
 params.this$b  
 )  
   
 if (i > 1){  
 violations <- draws.this > output[,i-1]  
 draws.this[violations] <- output[violations,i-1]  
 }  
 output[,i] <- draws.this  
}

### Method 4, downward replacement

## Method 4 : Downward Replacement  
for (i in n\_vars:1){  
 params.this <- est\_beta(  
 summary\_data[[i]]$mu,  
 summary\_data[[i]]$se^2  
 )  
 draws.this <- rbeta(  
 n\_psa,  
 params.this$a,   
 params.this$b  
 )  
 if (i < n\_vars){  
 violations <- draws.this < output[,i+1]  
 draws.this[violations] <- output[violations, i + 1]  
 }  
 output[,i] <- draws.this  
}

### Method 5, upward resampling

## Method 5 : Upward Resampling  
for (i in 1:n\_vars){  
 params.this <- est\_beta(  
 summary\_data[[i]]$mu,   
 summary\_data[[i]]$se^2  
 )  
 if (i ==1){  
 output[,1] <- rbeta(  
 n\_psa,  
 params.this$a,  
 params.this$b  
 )  
   
 } else {  
 for (j in 1:n\_psa){  
 continue <- F  
 while(continue==F){  
 val.this <- rbeta(1,   
 params.this$a,  
 params.this$b  
 )  
 if (val.this <= output[j,i-1]){  
 output[j, i] <- val.this  
 continue <- T  
 }  
 }  
 }  
 }  
}

### Method 6, upwards resampling

## Method 6 : Upward Resampling  
  
for (i in n\_vars:1){  
 params.this <- est\_beta(  
 summary\_data[[i]]$mu,   
 summary\_data[[i]]$se^2  
 )  
 if (i == n\_vars){  
 output[,n\_vars] <- rbeta(  
 n\_psa,  
 params.this$a,  
 params.this$b  
 )  
   
 } else {  
 for (j in 1:n\_psa){  
 continue <- F  
 while(continue==F){  
 val.this <- rbeta(1,   
 params.this$a,  
 params.this$b  
 )  
 if (val.this >= output[j,i+1]){  
 output[j, i] <- val.this  
 continue <- T  
 }  
 }  
 }  
 }  
}

### Method 7: AIVM Covariance

## Method 7 : AIVM Covariance  
if (n\_vars!=2) stop("Only two parameters allowed with this method")  
  
output <- make\_aivm\_cov\_2d(  
 mu\_x=summary\_data[[1]]$mu,  
 sd\_x=summary\_data[[1]]$se,  
 mu\_y=summary\_data[[2]]$mu,  
 sd\_y=summary\_data[[2]]$se,  
 colnames\_=names(summary\_data),  
 n\_psa\_=n\_psa  
 )$aivm\_samples

### Method 8: Lower bounded covariance fitting

## Method 8 : Lower bounded covariance retrofitting  
if (n\_vars!=2) stop("Only two parameters allowed with this method")  
  
output <- as.data.frame(  
 make\_bcvr\_2d(  
 mu\_x=summary\_data[[1]]$mu,  
 sd\_x=summary\_data[[1]]$se,  
 mu\_y=summary\_data[[2]]$mu,  
 sd\_y=summary\_data[[2]]$se,  
 n\_psa\_=n\_psa,  
 upper=F,  
 colnames\_=names(summary\_data),  
 quietly=quietly  
 )$samples  
)

### Method 9: upper bounded covariance fitting

## Method 9 : Upper Bounded covariance retrofitting  
if (n\_vars!=2) stop("Only two parameters allowed with this method")  
  
output <- as.data.frame(   
 make\_bcvr\_2d(  
 mu\_x=summary\_data[[1]]$mu,  
 sd\_x=summary\_data[[1]]$se,  
 mu\_y=summary\_data[[2]]$mu,  
 sd\_y=summary\_data[[2]]$se,  
 n\_psa\_=n\_psa,  
 upper=T,  
 colnames\_=names(summary\_data),  
 quietly=quietly  
 )$samples  
)

### Method 10: upwards difference method

## Method 10: Beta distribution difference modelling : upwards  
# lowest value is reference  
params.this <- est\_beta(  
 summary\_data[[1]]$mu,   
 summary\_data[[1]]$se^2  
)  
  
draws\_ref <- rbeta(  
 n\_psa,   
 params.this$a,  
 params.this$b  
 )  
output[,1] <- draws\_ref  
  
for (i in 2:n\_vars){  
   
 dif\_params.this <- get\_dif\_param(  
 summary\_data[[i-1]]$mu,   
 summary\_data[[i-1]]$se,  
   
 summary\_data[[i]]$mu,  
 summary\_data[[i]]$se  
 )  
   
 deltas.this <- rbeta(  
 n\_psa,  
 dif\_params.this$a,  
 dif\_params.this$b  
 )  
   
 output[,i] <- output[,i-1] - deltas.this  
}

### Method 11: downwards difference method

# Beta, downwards

params.this <- est\_beta(  
 summary\_data[[n\_vars]]$mu,   
 summary\_data[[n\_vars]]$se^2  
)  
draws\_ref <- rbeta(  
 n\_psa,   
 params.this$a,  
 params.this$b  
)  
output[,n\_vars] <- draws\_ref  
   
for (i in n\_vars:2){  
   
 dif\_params.this <- get\_dif\_param(  
 summary\_data[[i-1]]$mu,   
 summary\_data[[i-1]]$se,  
   
 summary\_data[[i]]$mu,  
 summary\_data[[i]]$se  
 )  
   
 deltas.this <- rbeta(  
 n\_psa,  
 dif\_params.this$a,  
 dif\_params.this$b  
 )  
  
 output[,i-1] <- output[,i] + deltas.this  
}