Article

Analysis of Stochastic Reserving Models By Means of NAIC Claims Data // Supplementary Materials

László Martinek [1,2]

[1] Department of Probability Theory and Statistics, Eötvös Loránd University, Pázmány Péter sétány 1/C, 1117 Budapest, Hungary
[2] NN Group, Prinses Beatrixlaan 35, 2595 AK The Hague, The Netherlands
* Correspondence: martinek@cs.elte.hu

Version May 26, 2019 submitted to Risks

Keywords: stochastic claims reserving, probabilistic forecast, comparison metrics, credibility, Monte Carlo

1. Supplementary materials

```
> # ================================================================================
> # Supplementary Material for paper
> #   'Analysis of Stochastic Reserving Models By Means of NAIC Claims Data'
> # Author: [blind]
> # Date: 6 May 2018
> # ================================================================================
>
> # Packages: ChainLadder, gglot2, reshape, xtable, rjags.
>
> # ================================================================================
> # Part I. Self-developed scripts.
> # ================================================================================
>
> # ================================================================================
> # Constants.
> # ================================================================================
> #Bootstrap repeats.
> BS.R <- 1000
> #Munich Chain Ladder bootstrap repeats.
> bootMCL.R <- 1000
> #Sample size of link ratios in the semi-stochastic method.
> Samplesize <- 5000
> #Use log-scale for CRPS plots.
> logscale.CRPS <- TRUE
> # ================================================================================
> # Read in the CAS csv data files from the CAS folder.
> # ================================================================================
>
> Get.Data <- function(folder){
+
+   #filenames of data per lines of business
+   nam <- list.files(folder)
+   nam <- nam[grep("_pos", nam)]
+   n <- length(nam)
+
+   res <- lapply(nam, function(x) read.csv(file = paste(folder,x,sep="/")))
+   names(res) <- gsub( "_.*$", "", nam )
```

```
+
+   return(res)
+
+ }
> #E.g.:
> #Data <- Get.Data(Folder.CAS)
>
> # ================================================================================
> # Normalize the complete quadrangles with the sum, scaled up by 'Sc'
> # Questions to investigate whether it is good to use this step.
> # Big company - tiny company problem.
> # How does it affect model like over-dispersed Poisson.
> # ================================================================================
>
> Normalize.Data <- function(Data, Sc=1){
+   for(i in 1:length(Data)){
+     Gr <- unique(Data[[i]]$GRCODE)
+     for(j in Gr){
+       aux.ind <- which(Data[[i]]$GRCODE == j)
+       if(sum(Data[[i]]$IncurLoss_C[aux.ind]) != 0){
+         Data[[i]]$IncurLoss_C[aux.ind] <- Sc*Data[[i]]$IncurLoss_C[ aux.ind ]/sum(Data[[i]]$IncurLoss_C[ aux.ind ])
+       }
+       if(sum(Data[[i]]$CumPaidLoss_C[aux.ind]) != 0){
+         Data[[i]]$CumPaidLoss_C[aux.ind] <- Sc*Data[[i]]$CumPaidLoss_C[ aux.ind ]/sum(Data[[i]]$CumPaidLoss_C[ aux.ind ])
+       }
+     }
+   }
+   return(Data)
+ }
> # ================================================================================
> # Returns a TRUE/FALSE vector, which of the companies should not be dropped out.
> # ================================================================================
> SelectSample <- function(Res){
+
+   res <- sapply(1:length(Res$Calculated), function(i){
+     sapply(1:dim(Res$Calculated[[i]])[2], function(j){
+       #Constraints for observations to take into account.
+       c1 <- length(which(is.na(Res$Calculated[[i]][,j])))==0
+       c2 <- Res$Calculated[[i]][grep("width", rownames(Res$Calculated[[i]])),j]!=0
+       c1&&c2
+     })
+   })
+
+   apply(res,1,all)
+
+ }
> # Drop out some of the results. 'munich' isn't taken into account.
> # Input is ResAll list with elements 'comauto', etc.
> SieveResults <- function(ResAll){
+   for(k in 1:length(ResAll)){
+     Res.aux <- ResAll[[k]]
+     Res.aux$Calculated <- Res.aux$Calculated[-which(names(ResAll[[k]]$Calculated)=="munich")]
+     Res.aux$Calculated <- Res.aux$Calculated[-which(names(ResAll[[k]]$Calculated)=="bootstrap.munich")]
+     ind <- which(SelectSample(Res.aux)==T)
+     Res.aux <- ResAll[[k]]
+     for(i in 1:length(Res.aux$Calculated)){ Res.aux$Calculated[[i]] <- Res.aux$Calculated[[i]][,ind] }
+     ResAll[[k]] <- Res.aux
+   }
+   return(ResAll)
+ }
> # ================================================================================
> # Create triangle related to a specific company. Result is the total rectangle.
> # Inputs are:
```

```
> #   (1) Data list from Get.Data
> #   (2) Line of Business from "comauto"  "medmal"   "othliab"  "ppauto"   "prodliab" "wkcomp"
> #   (3) gr code.
> #   (4) 'IncurLoss' or 'CumPaidLoss'
> # ============================================================================
>
> Get.Triangle <- function(Data, lob, grcode, type = 'CumPaidLoss'){
+
+   a <- Data[[lob]]
+   a <- a[a$GRCODE==grcode,c(3,4,5,grep(type, names(a)))]
+   n <- table(a$AccidentYear)
+   m <- length(unique(a$DevelopmentLag))
+   res <- matrix(a[,4], nrow=length(n), ncol=m, byrow = T)
+   rownames(res) = names(n)
+   colnames(res) = 1:m
+   return(res)
+
+ }
> # E.g.:
> # Tr <- Get.Triangle(Data, 'comauto', 32875)
>
> # ============================================================================
> # From the rectangle makes upper triangle for estimations.
> # Also lower triangle can be created with the opposite function.
> # ============================================================================
>
> Upper.Triangle <- function(Tr){
+
+   Tr <- Tr[,dim(Tr)[2]:1]
+   Tr[lower.tri(Tr)] = NA
+   Tr <- Tr[,dim(Tr)[2]:1]
+   return(Tr)
+
+ }
> Lower.Triangle <- function(Tr){
+
+   Tr <- Tr[,dim(Tr)[2]:1]
+   Tr[!lower.tri(Tr)] = NA
+   Tr <- Tr[,dim(Tr)[2]:1]
+   return(Tr)
+
+ }
> # ============================================================================
> # Based on the line of business abbreviation, return the entire name.
> # Used creating the analyses documents.
> # ============================================================================
>
> Get.Name <- function(type){
+
+   if(type == "comauto"){res <- "commercial auto and truck liability and medical insurance"}
+   if(type == "medmal"){res <- "medical malpractice - claims made"}
+   if(type == "othliab"){res <- "other liability - occurrence"}
+   if(type == "ppauto"){res <- "private passenger auto liability and medical"}
+   if(type == "prodliab"){res <- "product liability - occurrence"}
+   if(type == "wkcomp"){res <- "workers' compensation"}
+   return(res)
+
+ }
> # ============================================================================
> # Bootstrap Chain Ladder method, using package ChainLadder.
> # Triangles are cumulative.
> # ============================================================================
>
```

```
> Bootstrap = function(Tr, meth="gamma",  R=BS.R){
+
+   a <- BootChainLadder(Triangle=Tr, R, process.distr=meth)
+   latest <- summary(a)$Totals[1,1]
+   res = list( latest = latest,
+               IBNRs = a$IBNR.Totals)
+
+   return(res)
+
+ }
> # E.g.:
> # Bootstrap(Tr, "gamma", 10)
>
> # ============================================================================
> # Munich Chain Ladder method, using package ChainLadder.
> # Triangles are cumulative, paid and incurred both.
> # Input type: "Paid" or "Incurred" (sum of diagonal in the paid or incurred triangle).
> # Ultimate value should be very close, although worth checking.
> # ============================================================================
>
> Munich = function(Tr.paid, Tr.inc, type="Paid"){
+
+   a <- MunichChainLadder(Tr.paid, Tr.inc)
+   latest <- summary(a)$Totals[[type]][1]
+
+   #Bootstrapping the MCL: implemented by function BootstrapMunich.
+   #Here only the point estimate.
+   res = list( latest = latest,
+               IBNRs = summary(a)$Totals[[type]][2]-latest)
+
+   return(res)
+
+ }
> # E.g.:
> # Munich(Tr.paid.up, Tr.inc.up, "Paid")
>
> # ============================================================================
> # Bootstrap Munich Chain Ladder method.
> # type is either 'Paid' or 'Incurred'.
> # ============================================================================
>
> BootstrapMunich <- function(Tr.p, Tr.i, type="Paid"){
+
+   a <- MCLbootstrap(Tr.p, Tr.i, bootMCL.R)
+   latest <- sum(a$Reserves[[paste("Latest",type,sep="")]])
+   res = list( latest = latest,
+               IBNRs = apply(a[[paste("Ultimate",type,sep="")]],2,sum) - latest)
+
+   return(res)
+
+ }
> #E.g.: similar to 'Munich'.
>
> # ============================================================================
> # Correlated chain ladder model. MCMC model.
> # ============================================================================
>
> # type: line of business, such as "comauto", etc.
> CCL <- function(Data, type, grpcode, Folder.CAS){
+
+   res <- function.CCL(type = type, grpcode = as.character(grpcode),
+                       outfile = "outCCL.csv", losstype = "incloss", Folder.CAS)
+   scen <- res$at.wd10
```

```
+
+   Tr <- Get.Triangle(Data, type, grpcode)
+   latest <- sum(sapply(1:10, function(i) Tr[i,10-i+1]))
+
+   IBNRs <- apply(scen, 1, function(x) sum(x)) - latest
+   if(length(which(is.na(IBNRs)))>0) IBNRs <- IBNRs[-which(is.na(IBNRs))]
+
+   res <- list(latest = latest, IBNRs = IBNRs)
+   return(res)
+
+ }
> #E.g.:
> #CCL(Data, "comauto", 353, Folder.CAS)
>
> # ============================================================================
> # Correlated Incremental Trend Model. MCMC model.
> # ============================================================================
>
> # type: line of business, such as "comauto", etc.
> CIT <- function(Data, type, grpcode, Folder.CAS){
+
+   res <- function.CIT(type = type, grpcode = as.character(grpcode),
+                       outfile = "outCIT.csv", losstype = "paidloss", Folder.CAS)
+   scen <- res$at.wd10
+
+   Tr <- Get.Triangle(Data, type, grpcode)
+   latest <- sum(sapply(1:10, function(i) Tr[i,10-i+1]))
+
+   IBNRs <- apply(scen, 1, function(x) sum(x)) - latest
+   if(length(which(is.na(IBNRs)))>0) IBNRs <- IBNRs[-which(is.na(IBNRs))]
+
+   res <- list(latest = latest, IBNRs = IBNRs)
+   return(res)
+
+ }
> # E.g.:
> # CIT(Data, "comauto", 353, Folder.CAS)
>
> # ============================================================================
> # Semi stochastic model based on all insurance companies' development factors.
> # ============================================================================
> # type <- "comauto"
> # grpcode <- unique(Data[[type]]$GRCODE)
>
> Get.Alphas <- function(grpcode, type, SampleSize){
+
+   linkratios <- sapply(1:length(grpcode), function(i){
+     Tr <- Get.Triangle(Data, type, grpcode[i])
+     Tr.up <- Upper.Triangle(Tr)
+     attr(ata(Tr.up),"vwtd")
+   })
+   #Store non-NA linkratios in the list 'Alpha'. Sample will be drawn from each element of Alpha.
+   Alpha <- sapply(1:dim(linkratios)[1],
+                   function(i){
+                     linkratios[i,intersect( which(is.na(linkratios[i,])==FALSE), which(abs(linkratios[i,])<Inf) )]
+                   })
+   N <- length(Alpha)
+   linksample <- sapply(1:N , function(j) sample(Alpha[[j]], SampleSize, replace = TRUE))
+
+   return(linksample)
+
+ }
> SemiSt <- function(Data, type, grpcode, linksample){
```

```
+
+   Tr <- Get.Triangle(Data, type, grpcode[1])
+   Tr.up <- Upper.Triangle(Tr)
+
+   Dim <- 10
+   res <- sapply(1:dim(linksample)[1],
+                 function(k)sapply(2:Dim, function(j) Tr.up[j,Dim-j+1]*prod(linksample[k,(Dim-1):(Dim-j+1)]))
+   )
+
+   #'Latest' without the first complete row.
+   latest <- sum(sapply(2:Dim, function(j) Tr.up[j,Dim-j+1]))
+   IBNRs <- apply(res,2,sum) - latest
+   real.IBNR <- sum(Tr[2:Dim,Dim]) - latest
+   #'Latest' with the first complete row.
+   latest <- latest + Tr.up[1,Dim]
+   res <- list(latest = latest, IBNRs = IBNRs, real.IBNR = real.IBNR)
+
+   return(res)
+
+ }
> # E.g.
> # grpcode <- unique(Data[["comauto"]]$GRCODE)
> # linksample <- Get.Alphas(grpcode, "comauto", 1000)
> # semi <- SemiSt(Data, "comauto", grpcode[2], linksample)
>
> # ============================================================================
> # Calculation of metrics.
> # CRPS and PIT score based on a sample IBNRs and the real IBNR.
> # Source: from snippets in "FolderPath/allfun.RData"
> # ============================================================================
>
> ScoreFun.low <- function(x,c,n,y){
+   res = (x[1]-c) + sum( y[1:(n-1)]*((n-1):1)^2 )/(n^2)
+   return(c(-res,0))
+ }
> ScoreFun.high <- function(x,c,n,y){
+   res = (c-x[n]) + sum( y[1:(n-1)]*(1:(n-1))^2 )/(n^2)
+   return(c(-res,1))
+ }
> ScoreFun.norm <- function(x,c,n,k,y){
+   res = sum( y[1:k]*(1:k)^2 )/(n^2) + sum( y[(k+1):(n-1)]*((n-k-1):1)^2 )/(n^2) + (x[k+1]-c)*(n-2*k)/n
+   return(c(-res,k/n))
+ }
> ScoreFun <- function(x,c){
+   x=sort(x); n=length(x); k=length(x[x<c]); y=x[-1]-x[-n]
+   if(k==0){ res = ScoreFun.low(x,c,n,y) }
+   if(k==1){
+     res = (c-x[1])^2/(n^2) + (x[2]-c)^2*((n-1)/n)^2 + sum((y[-1]*((n-2):1)/n)^2)
+     res = c(-res,k/n)
+   }
+   if((k>1) && (k<n-1)){res = ScoreFun.norm(x,c,n,k,y) }
+   if(k==(n-1)){
+     res = sum( y[1:(k-1)]*(1:(k-1))^2 )/(n^2) + (c-x[n-1])^2*((n-1)/n)^2 + (x[n]-c)^2/(n^2)
+     res = c(-res,k/n)
+   }
+   if(k==n){ res = ScoreFun.high(x,c,n,y) }
+
+   return(list(CRPS = res[1], PIT = res[2]))
+ }
> # Msep <- function(x,b){}
> # E.g.:
>
> # ============================================================================
```

```
> # Calculations of the Mean Square error of prediction.
> # ============================================================================
>
> # The following function takes E(hat(x)-x)^2
> # Input:
> #   - list with 'latest' and 'IBNRs', see Bootstrap output, for instance.
> #   - total rectangle Tr
> Msep.1 <- function(ibnrs, real.IBNR){
+
+   res <- mean((ibnrs$IBNRs - real.IBNR)^2) / real.IBNR^2#normalized by the real^2!
+   return(res)
+
+ }
> # E.g.:
> # Tr <- Get.Triangle(Data, 'comauto', 715)
> # ibnrs <- Bootstrap(Upper.Triangle(Tr), "gamma", 100)
> # real.IBNR <- sum(Tr[,dim(Tr)[2]])-ibnrs$latest
> # Msep.1(ibnrs,real.IBNR)
>
> # ============================================================================
> # Coverage and average width.
> # ============================================================================
>
> # Coverage values are calculated for each of the triangles.
> # It results in 0-1 values for each, 1 if the real.IBNR falls into the (1+-alpha)/2 quantile,
> # 0 otherwise.
> # In the end the average of them should be taken.
> # alpha is in form c(0.6,0.9) (values between 0 and 1).
> Cover.01 <- function(samp, real.IBNR, alpha){
+
+   int <- sapply(
+     alpha, function(x) c(quantile(samp, (1-x)/2, na.rm = T),quantile(samp, (1+x)/2, na.rm = T))
+   )
+   if(is.na(real.IBNR) || is.na(samp)){
+     res <- rep(NA, length(alpha))
+   }else{
+     res <- as.list(apply(int, 2, function(x) if(x[1] < real.IBNR && x[2] > real.IBNR){1}else{0}))
+   }
+   names(res) <- paste(round(alpha*100),"% cover",sep="")
+   return(res)
+
+ }
> # The width of prediction (based on one prediction).
> # In order to get the average width, take the average of these.
> # Since the magnitude of underlying data is quite different,
> # the width is normalized by the real.IBNR value.
> CoverWidth <- function(samp, real.IBNR, alpha){
+
+   res <- as.numeric(
+     sapply( alpha, function(x) quantile(samp, (1+x)/2, na.rm=T) - quantile(samp, (1-x)/2, na.rm=T))
+   )
+   if(is.na(real.IBNR)){res <- rep(NA, length(alpha))}else{res <- res/real.IBNR}
+   res <- as.list(res)
+   names(res) <- paste(round(alpha*100),"% width",sep="")
+   return(res)
+
+ }
> # ============================================================================
> # As a function of 'Data' and a homogeneous risk group, such as 'comauto',
> # metrics are calculated in case of bootstrap / Munich Chain Ladder based models.
> # Inputs field 'Methods' can be:
> #   - "boot.od.pois"
> #   - "boot.gamma"
```

```
> #   - "CCL", see Meyers 2015
> #   - "CIT", see Meyers 2015
> #   - "SemiSt" as the semi-stochastic method
> #   - "cred.bootstrap.od.pois"
> #   - "standard.bootstrap.od.pois" from own functions, identical to boot.od.pois
> #
> # linksample contrains the random draw of link ratios for the SemiSt method
> # FCred contains the credibility and standard development factors
> # ============================================================================
>
> # Bootstrap and other simple paid (OR incurred) based component.
> # Only 1 triangle used for a provisions calculation.
> Metrics.Basic <- function(gr.code, Data, type, Methods, cover.alpha, Folder.CAS,
+                           linksample = NULL, FCred = NULL){
+
+   Tr <- Get.Triangle(Data, type, gr.code)
+   Tr.up <- Upper.Triangle(Tr)
+
+   #Localize reserving model, and generate IBNR estimations.
+   if(substr(Methods,1,4)=="boot"){
+     meth <- substr(Methods,6,20)
+     #Spare (or eventually 0) data have to be taken out.
+     #Now simply taken out if error occurs with the try function.
+     Bs <- try(Bootstrap(Tr.up, meth), silent = T)}
+
+   #Methods described in Meyers CAS monography.
+   if(Methods == "CCL"){
+     Bs <- try(CCL(Data, type, gr.code, Folder.CAS), silent = T)
+   }
+
+   if(Methods == "CIT"){
+     Bs <- try(CIT(Data, type, gr.code, Folder.CAS), silent = T)
+   }
+
+   #Semi-stochastic method.
+   if(Methods == "SemiSt"){
+     Bs <- try(SemiSt(Data, type, gr.code, linksample), silent = T)
+   }
+
+   #Over-dispersed Poisson model from the own codes.
+   if(Methods == "standard.bootstrap.od.pois"){
+     Bs <- try(OwnBootstrap(Data, type, gr.code, FCred, "standard.bootstrap.od.pois"))
+   }
+
+   #Credibility-type over-dispersed Poisson bootstrap model.
+   if(Methods == "cred.bootstrap.od.pois"){
+     Bs <- try(OwnBootstrap(Data, type, gr.code, FCred, "cred.bootstrap.od.pois"))
+   }
+
+   #New methods to be added here.
+
+   if(typeof(Bs)=="list"){
+     real.IBNR <- sum(Tr[,dim(Tr)[2]])-Bs$latest
+     res <- c(ScoreFun(Bs$IBNRs, real.IBNR),
+              real.IBNR = real.IBNR, estim.IBNR = mean(Bs$IBNRs),
+              Msep = Msep.1(Bs,real.IBNR))
+     #Include coverage and average width.
+     res <- c(res, Cover.01(Bs$IBNRs, real.IBNR, cover.alpha))
+     #Include average width
+     res <- c(res, CoverWidth(Bs$IBNRs, real.IBNR, cover.alpha))
+   }else{
+     res <- list(CRPS = NA, PIT = NA, real.IBNR = NA, estim.IBNR = NA, Msep = NA)
+     #res <- c(res, rep(NA, 2*length(cover.alpha)))
```

```
+     #Include coverage and average width.
+     res <- c(res, Cover.01(res$real.IBNR, res$real.IBNR, cover.alpha))
+     #Include average width
+     res <- c(res, CoverWidth(res$real.IBNR, res$real.IBNR, cover.alpha))
+   }
+
+   return(res)
+
+ }
> # Munich Chain Ladder based component, or any which uses Paid-Incurred triangles.
> # Methods: 'munich' or 'bootstrap.munich'
> Metrics.PaidInc <- function(gr.code, Data, lob, Methods, cover.alpha){
+
+   Tr.paid <- Get.Triangle(Data, lob, gr.code, "CumPaidLoss")
+   Tr.inc <- Get.Triangle(Data, lob, gr.code, "IncurLoss")
+   Tr.paid.up <- Upper.Triangle(Tr.paid)
+   Tr.inc.up <- Upper.Triangle(Tr.inc)
+
+   #"Paid" can be changed.
+   if(Methods=="munich"){
+     Bs <- try(Munich(Tr.paid.up, Tr.inc.up, "Paid"), silent=T) }
+
+   if(Methods=="bootstrap.munich"){
+     Bs <- try(BootstrapMunich(Tr.paid, Tr.inc, "Paid"), silent=T) }
+
+   #Further methods to be added here.
+
+   if(typeof(Bs)=="list"){
+     real.IBNR <- sum(Tr.paid[,dim(Tr.paid)[2]])-Bs$latest
+     aux <- try(ScoreFun(Bs$IBNRs, real.IBNR), silent=T)
+     if(length(aux)==1){aux = list(CRPS = NA, PIT = NA)}else{}
+     res <- c( aux, real.IBNR = real.IBNR, estim.IBNR = mean(Bs$IBNRs),
+               Msep = Msep.1(Bs,real.IBNR))
+     #Include coverage and average width.
+     res <- c(res, Cover.01(Bs$IBNRs, real.IBNR, cover.alpha))
+     #Include average width
+     res <- c(res, CoverWidth(Bs$IBNRs, real.IBNR, cover.alpha))
+   }else{
+     res <- list(CRPS = NA, PIT = NA, real.IBNR = NA, estim.IBNR = NA, Msep = NA)
+     #res <- c(res, rep(NA, 2*length(cover.alpha)))
+     #Include coverage and average width.
+     res <- c(res, Cover.01(res$real.IBNR, res$real.IBNR, cover.alpha))
+     #Include average width
+     res <- c(res, CoverWidth(res$real.IBNR, res$real.IBNR, cover.alpha))
+   }
+
+   return(res)
+
+ }
> # General function composed from the previous.
> Metrics <- function(Data, type, Methods, Companies="All", cover.alpha, Folder.CAS){
+
+   if(Companies == "All"){
+     a <- Data[[type]]
+   }else{
+     Companies <- intersect(Companies, 1:length(unique(Data[[type]]$GRCODE)))
+     unit.length <- dim(Data[[type]])[1] / length(unique(Data[[type]]$GRCODE))
+     No.companies <- sort(sapply(1:unit.length, function(n) n+(Companies-1)*unit.length))
+     a <- Data[[type]][No.companies,]}
+
+   if(Methods == "boot.gamma" || Methods == "boot.od.pois" || Methods == "CCL" || Methods == "CIT"){
+     res <- sapply( unique(a$GRCODE), function(x) Metrics.Basic(x, Data, type, Methods, cover.alpha, Folder.CAS) )
+   }
```

```
+   if(Methods == "SemiSt"){
+     grpcode <- unique(a$GRCODE)
+     linksample <- Get.Alphas(grpcode, type, Samplesize)
+     res <- sapply( unique(a$GRCODE), function(x) Metrics.Basic(x, Data, type, Methods, cover.alpha, Folder.CAS, linksampl
+   }
+   if(Methods == "munich" || Methods == "bootstrap.munich"){
+     res <- sapply( unique(a$GRCODE), function(x) Metrics.PaidInc(x, Data, type, Methods, cover.alpha) )
+   }
+   if(Methods == "cred.bootstrap.od.pois" || Methods == "standard.bootstrap.od.pois"){
+     FCred <- f.cred(Data,type)
+     res <- sapply( unique(a$GRCODE), function(x) Metrics.Basic(x, Data, type, Methods,
+                                                                cover.alpha, Folder.CAS, FCred = FCred) )
+   }
+
+   colnames(res) <- unique(a$GRCODE)[1:dim(res)[2]]
+
+   return(res)
+
+ }
> # E.g.:
> # Metrics(Data, "comauto", "boot.gamma", 5, c(0.66,0.9), Folder.CAS)
>
> # ============================================================================
> # Total Results.
> # ============================================================================
>
> ResultSummary <- function(Data, type, companies, Methods, alphas, Folder.CAS){
+
+   M <- lapply(Methods, function(x) Metrics(Data, type, x, companies, alphas, Folder.CAS))
+   names(M) = Methods
+
+   #Plotting the PIT results.
+   pit <- sapply(Methods, function(x) unlist(M[[x]][grep("PIT",rownames(M[[x]])),]))
+   pit <- melt(pit, id = c('cond'), variable_name = 'variab')
+   pit.plot <- ggplot(na.omit(pit), aes(x=value)) +
+     geom_histogram(origin=0, binwidth=0.1, colour="black", fill="white") +
+     facet_grid(X2 ~ ., scales = "free") + facet_wrap(facet = ~ X2, ncol=3) +
+     ylab("frequency") + labs(title = "Histograms of PIT values") +
+     scale_x_continuous(limits = c(0,1))
+
+   #Plotting CRPS boxplots.
+   crps <- sapply(Methods, function(x) unlist(M[[x]][grep("CRPS",rownames(M[[x]])),]))
+   #crps <- cbind(cond=1:dim(crps)[1], crps)
+   Df <- melt(crps, variable_name = 'methpairs')
+   #Df$cond=rep(1:dim(crps)[2],each=dim(crps)[1])
+   crps.plot <- ggplot(Df, aes(factor(X2),value)) + geom_boxplot() +
+     ylab("score values") + xlab("method") + labs(title = "Boxplots of CRPS values")
+
+   #Average CRPS values.
+   crps <- sapply(Methods, function(x) unlist(M[[x]][grep("CRPS",rownames(M[[x]])),]))
+   mean.crps <- apply(crps,2, function(x) mean(x,na.rm=T))
+   mean.crps <- rbind(mean.crps,
+                      SampleSize = apply(crps, 2, function(x) length(which(is.na(x)==FALSE))))
+
+   #Mean square error of prediction.
+   msep <- sapply(Methods, function(x) unlist(M[[x]][grep("Msep",rownames(M[[x]])),]))
+   mean.msep <- apply(msep,2, function(x) mean(x[which(x<1e+12)],na.rm=T))
+   mean.msep <- rbind(mean.msep,
+                      SampleSize = apply(msep, 2, function(x){
+                        length( intersect(which(is.na(x)==FALSE), which(x < 1e+12))) })
+   )
+
+   #Coverage and average width.
```

```
+   cover <- sapply(Methods, function(x){
+     apply(M[[x]][grep("cover",rownames(M[[x]])),], 1, function(y) mean(as.numeric(y), na.rm=T))
+   })
+   averagewidth <- sapply(Methods, function(x){
+     apply(M[[x]][grep("width",rownames(M[[x]])),], 1, function(y){
+       y <- as.numeric(y)
+       mean( y[which(y!=Inf)], na.rm=T )})
+   })
+
+   res <- list(Calculated = M, PitPlot = pit.plot, CrpsPlot = crps.plot,
+               MeanCrps = mean.crps,
+               Msep = mean.msep,
+               Coverage = cover, AverageWidth = averagewidth)
+
+   return(res)
+
+ }
> # ============================================================================
> # Total results combined. If all the expensive calculations are done already.
> # ============================================================================
>
> ResultSummaryCombined <- function(Result, types, Drop = NULL){
+
+   Res <- list()
+   types <- c(types, "Empty")
+   Methods <- unique(unlist(sapply(types, function(x) names(Result[[x]]$Calculated))))
+   if(!is.null(Drop)){ Methods <- setdiff(Methods,Drop) } #if some of the results are not needed afterwards (dropped)
+   for(y in Methods){
+     Res$Calculated[[y]] <- data.frame(sapply(types,
+                                              function(x){
+                                                if(y %in% names(Result[[x]]$Calculated)){
+                                                  Result[[x]]$Calculated[[y]]
+                                                }else{
+                                                  NA
+                                                }
+                                              } ))
+   }
+
+   M <- Res$Calculated
+   if(length(which(colnames(M[[1]]) == "Empty"))>0){
+     aux.empty <- which(colnames(M[[1]]) == "Empty")
+     for(i in 1:length(M)) M[[i]] <- M[[i]][,-aux.empty]
+   }
+
+   #Plotting the PIT results.
+   pit <- sapply(Methods, function(x) unlist(M[[x]][grep("PIT",rownames(M[[x]])),]))
+
+   pit <- melt(pit, id = c('cond'), variable_name = 'variab')
+   pit <- data.frame("X2" = pit$X2, "value" = pit$value)
+   #names(pit) <- c("value","X2")
+   pit.plot <- ggplot(na.omit(pit), aes(x=value)) +
+     geom_histogram(origin = 0, binwidth=0.1, colour="black", fill="white") +
+     facet_grid(X2 ~ ., scales = "free") + facet_wrap(facet = ~ X2, ncol=3) +
+     ylab("frequency") + labs(title = "Histograms of PIT values") +
+     scale_x_continuous(limits = c(0,1))
+
+   #Plotting CRPS boxplots.
+   crps <- sapply(Methods, function(x) unlist(M[[x]][grep("CRPS",rownames(M[[x]])),]))
+   #crps <- cbind(cond=1:dim(crps)[1], crps)
+
+   Df <- melt(crps, variable_name = 'methpairs')
+   Df <- data.frame("X2" = Df$X2, "value" = Df$value)
+   #names(Df) <- c("value","X2")
```

```
+   #Df$cond=rep(1:dim(crps)[2],each=dim(crps)[1])
+   if(logscale.CRPS==TRUE){
+     crps.plot <- ggplot(Df, aes(factor(X2),sign(value)*log(abs(value)))) + geom_boxplot() +
+       ylab("score values (on logarithmic scale)") + xlab("method") + labs(title = "Boxplots of CRPS values") +
+       theme(axis.text.x=element_text(angle=45,hjust=1,vjust=1)) +
+       theme(axis.title=element_text(size=14))
+   }else{
+     crps.plot <- ggplot(Df, aes(factor(X2),value)) + geom_boxplot() +
+       ylab("score values") + xlab("method") + labs(title = "Boxplots of CRPS values") +
+       theme(axis.text.x=element_text(angle=45,hjust=1,vjust=1))
+   }
+
+
+   #Average CRPS values.
+   mean.crps <- Get.Metrics(Res,"CRPS")
+   #Mean square error of prediction.
+   mean.msep <- Get.Metrics(Res,"Msep")
+
+   #Coverage and average width.
+   cover <- data.frame()
+   coverNames <- rownames(M[[1]])[grep("cover", rownames(M[[1]]))]
+   for(z in coverNames){ cover <- rbind(cover,round(Get.Metrics(Res,z),3)) }
+
+   averagewidth <- data.frame()
+   widthNames <- rownames(M[[1]])[grep("width", rownames(M[[1]]))]
+   for(z in widthNames){ averagewidth <- rbind(averagewidth,round(Get.Metrics(Res,z),3)) }
+
+   res <- list(Calculated = M, PitPlot = pit.plot, CrpsPlot = crps.plot,
+               MeanCrps = mean.crps,
+               Msep = mean.msep,
+               Coverage = cover, AverageWidth = averagewidth)
+
+   return(res)
+
+ }
> # The following function combines 2 results of function ResultSummary.
> # methods1, methods2 inputs are characters describing the methods to be taken out of the Res1, Res2
> # E.g. methods1 = c("boot.od.pois"), methods2 = c("SemiSt","CCL")
> # Output: list Res with element $Calculated combined.
> # Use if 2 separate sets of methods were run separately.
>
> ResultSummaryCombined2 <- function(Res1, Res2, methods1, methods2){
+
+   methods1 <- intersect( names(Res1$Calculated), methods1 )
+   methods2 <- intersect( names(Res2$Calculated), methods2 )
+   Res <- list()
+   for(x in methods1) Res$Calculated[[x]] <- Res1$Calculated[[x]]
+   for(x in methods2) Res$Calculated[[x]] <- Res2$Calculated[[x]]
+
+   return(Res)
+
+ }
> # ============================================================================
> # Get metrics from the calculated data 'Res'.
> # MetricName stands for "CRPS" or "67% cover" etc.
> # SummaryType stands for "Mean" by default, or "Median" or "1st Qu." etc.
> # ============================================================================
> Get.Metrics <- function(Res, MetricName, SummaryType="Mean"){
+
+   Methods <- names(Res$Calculated)
+   res <- sapply(Methods, function(x){
+     ind <- grep(MetricName, rownames(Res$Calculated[[x]]))
+     vec <- as.numeric(unlist(Res$Calculated[[x]][ind,]))
```

```
+
+     #Values above absolute value 1e+12 and NA values are dropped
+     ind1 <- intersect(which(is.na(vec)==FALSE) , which(abs(vec)<1e+12))
+     #In addition, drop companies where the estimation doesn't produce reasonable solution.
+     ind2 <- grep("estim.IBNR", rownames(Res$Calculated[[x]]))
+     vec2 <- as.numeric(unlist(Res$Calculated[[x]][ind2,]))
+     ind2 <- which(abs(vec2)<1e+24)
+
+     ind <- intersect(ind1, ind2)
+     vec <- vec[ind]
+     c( summary(vec)[[SummaryType]], length(vec) )
+   })
+   if(MetricName == "Msep" || MetricName == "CRPS"){
+     RowNames <- c(paste(SummaryType, MetricName, sep="."), "SampleSize")
+   }else{
+     RowNames <- c(MetricName, "SampleSize")
+   }
+   rownames(res) = RowNames
+
+   return(res)
+
+ }
> #E.g.
> #Get.Metrics(Res,"CRPS")
> #Get.Metrics(Res,"90% width")
>
> # ============================================================================
> # Get metrics from the calculated data 'Res'.
> # ============================================================================
>
> # ========================
> # Estimate sigma_j values.
> # ========================
> # Calculate sigma_j values for each company, fixed j.
> # Trs : list containing triangles, j : F_j index.
> sigma.j <- function(Trs,j){
+   J <- dim(Trs[[1]])[1]
+   K <- length(Trs)
+   sapply(1:K, function(k){
+     tr <- Trs[[k]]
+     tr1 <- tr[1:(J-j),j]; tr2 <- tr[1:(J-j),j+1]
+     sum( tr1 * (tr2/tr1 - sum(tr2)/sum(tr1))^2 , na.rm = TRUE) / (J-j-1)
+   })
+ }
> sigma.hat <- function(Data, type){
+   grCodes <- unique(Data[[type]]$GRCODE)
+   Trs <- lapply(grCodes, function(i) Get.Triangle(Data, type, i))
+   sapply(1:(dim(Trs[[1]])[1]-1), function(j) mean(sigma.j(Trs,j), na.rm = TRUE))
+ }
> # E.g.:
> # sigma.hat(Data, "othliab")
>
> # ========================
> # Calculate Sum(C_ij) values, i = 1..I-j
> # ========================
> # Sum(C_i,j+1) / Sum(C_i,j+1) values per company, fixed j.
> f.hat <- function(tr){
+   J = dim(tr)[2]
+   sapply(1:(J-1), function(j){
+     c(  sum(tr[1:(J-j),j+1]) / sum(tr[1:(J-j),j]) )
+   })
+ }
> # ========================
```

```
> # Estimate tau_j values.
> # ======================
> tau.j <- function(Trs,j){
+
+   K=length(Trs)
+   J=dim(Trs[[1]])[1]
+   #Calculation of c_j coefficients.
+   #Sum(C_ij) values per company, fixed j.
+   w <- sapply(1:K, function(k) Trs[[k]][1:(J-j),j])
+   if(!is.null(dim(w))) w <- apply(w , 2, sum)
+   c.j <- 1 / sum( w/sum(w) * (1-w/sum(w)) ) * (K-1)/K
+   #Tau_j calculation.
+   w2 <- sapply(1:K, function(k) Trs[[k]][1:(J-j),j+1])
+   if(!is.null(dim(w2))) w2 <- apply(w2 , 2, sum)
+   a <- sum( w/sum(w) * (w2/w - sum(w2)/sum(w))^2, na.rm = TRUE ) / (K-1)*K
+   b <- mean(sigma.j(Trs,j), na.rm = TRUE) * K / sum(w)
+   (a-b) * c.j
+
+ }
> tau.hat <- function(Data, type){
+
+   grCodes <- unique(Data[[type]]$GRCODE)
+   Trs <- lapply(grCodes, function(i) Get.Triangle(Data, type, i))
+   sapply(1:(dim(Trs[[1]])[1]-1), function(j) tau.j(Trs,j))
+
+ }
> # ======================
> # Estimate the kappa = sigma^2 / tau^2 parameters.
> # ======================
> kappa.hat <- function(Data, type){
+
+   Sigma2 <- sigma.hat(Data,type)
+   Tau2 <- sapply( tau.hat(Data,type), function(x) max(0,x) )
+   Kap <- Sigma2/Tau2
+   Kap[union(which(is.na(Kap)), which(Kap == Inf))] <- Inf
+   return(Kap)
+
+ }
> # ======================
> # Calculate alpha_j values for each company.
> # ======================
> alpha.hat <- function(Data,type){
+
+   kappa.j <- kappa.hat(Data,type)
+   grCodes <- unique(Data[[type]]$GRCODE)
+   Trs <- lapply(grCodes, function(i) Get.Triangle(Data, type, i))
+   K=length(Trs)
+   J=dim(Trs[[1]])[1]
+   w <- sapply(1:(J-1), function(j) sapply(1:K, function(k) sum(Trs[[k]][1:(J-j),j])))
+   alpha <- sapply(1:(J-1), function(j) w[,j]/(w[,j]+kappa.j[j]))
+   return(alpha)
+
+ }
> # E.g.
> # alpha.hat(Data,"comauto")
>
> # ======================
> # Credibility estimation of F_j values.
> # ======================
> f.cred <- function(Data,type){
+
+   alpha <- alpha.hat(Data,type)
+
```

```
+   grCodes <- unique(Data[[type]]$GRCODE)
+   Trs <- lapply(grCodes, function(i) Get.Triangle(Data, type, i))
+   K <- length(Trs)
+   f.hats <- lapply(Trs, f.hat)
+   J <- dim(Trs[[1]])[1]
+   f.j <- sapply(1:(J-1), function(j){
+     sum(sapply(1:K, function(k) f.hats[[k]][j]*alpha[k,j]), na.rm = TRUE) / sum(alpha[,j])
+   })
+
+   f.j[which(is.na(f.j))] <- 0
+   #Since the triangles are not trapezoids, last sigma2 amd tau2 cannot be estimated.
+   #There we use the original factor of the individual.
+   #It can also be possible that neither hat(F), nor f_j exists.
+   F.cred <- sapply(1:K, function(k){
+     aux <- f.hats[[k]]*alpha[k,]; aux[is.na(aux)] <- 0
+     aux <- aux + f.j*(1-alpha[k,])
+     aux[which(aux==0)] <- f.hats[[k]][which(aux==0)]
+     aux
+   })
+
+   F.hat <- sapply(1:K, function(k){ f.hats[[k]] })
+   #If no other solution (no variance etc.), the simple average of F.hat used in the last credibilities.
+   #Only for trianlges where there is neither own development factor nor alpha.
+   F.cred[J-1,which(is.na(F.cred[J-1,]))] <- mean(F.cred[J-1,which(!is.na(F.cred[J-1,]))])
+   F.cred[J-2,which(is.na(F.cred[J-2,]))] <- mean(F.cred[J-2,which(!is.na(F.cred[J-2,]))])
+
+   return(list(F.cred=F.cred, F.hat=F.hat))
+
+ }
> # ======================
> # Plot development patterns, as a function of development years.
> # ======================
> plot.pattern <- function(fc,grpNames){
+
+   aux <- rbind(matrix(1, ncol=dim(fc)[2]), fc[1,])
+   for(i in 2:dim(fc)[1]) aux <- rbind(aux, apply(fc[1:i,],2,prod))
+   rownames(aux) <- 1:(dim(fc)[1]+1)
+   colnames(aux) <- grpNames
+   fc <- melt(aux)
+   names(fc) <- c("dev","comp","value")
+   res <- ggplot(data=fc, aes(x=dev, y=value, group=comp, shape=factor(comp), colour=factor(comp))) +
+     geom_line(size=0.5, aes(linetype=factor(comp)), size=1) +     # Set linetype by comp
+     geom_point(size=2.5, fill="white") +         # Use larger points, fill with white
+     #expand_limits(y=0) +                        # Set y range to include 0
+     scale_colour_hue(name="company code",      # Set legend title
+                      l=30)  +                  # Use darker colors (lightness=30)
+     scale_shape_manual(name="company code",
+                        values=c(22:(22-dim(aux)[2]+1))) +      # Use points with a fill color
+     scale_linetype_discrete(name="company code") +
+     xlab("year") + ylab("product of development factors") + # Set axis labels
+     ggtitle(paste("Multiplicative accumulation of development \n factors in business line ",type,".",sep="")) +     # Set
+     theme_bw() +
+     theme(legend.position=c(.9, .25), legend.text=element_text(size=7), legend.title=element_text(size=7)) +           #
+     scale_x_discrete(limit = c(1:dim(aux)[1]), labels = paste(1:dim(aux)[1]))
+
+   return(res)
+
+ }
> # E.g.:
> # type <- "comauto"
> # FCred <- f.cred(Data,type)
> # toPlot <- c(2,5,8,9)
> # grpNames <- unique(Data$comauto$GRCODE)[toPlot]
```

```
> # fc <- FCred$F.hat[,toPlot]
> # p1 <- plot.pattern(fc,grpNames)
> # fc <- FCred$F.cred[,toPlot]
> # p2 <- plot.pattern(fc,grpNames)
>
> # Bootstrap with the credibility development factors.
> # k = 1
> # type = "comauto"
> # grCodes <- unique(Data[[type]]$GRCODE)
> # Trs <- lapply(grCodes, function(i) Get.Triangle(Data, type, i))
> # tr <- Trs[[k]]
>
> # ======================
> # Calculating backwards the cumulative elements.
> # ======================
> BackwardDev <- function(tr, FCred, k){
+
+   tr <- Upper.Triangle(tr)
+   J <- dim(tr)
+
+   #With the credility development factors.
+   dev <- FCred$F.cred[,k]
+   tr2 <- tr
+   for(i in 1:(J[1]-1)){ jj <- J[1]-i; for(j in jj:1){ tr2[i,j] <- tr2[i,j+1]/dev[j] } }
+   aux <- tr2
+
+   #With the standard CL development factors.
+   dev <- FCred$F.hat[,k]; if(length(which(is.na(dev)))>0){dev[which(is.na(dev))] <- 1}
+   tr2 <- tr
+   for(i in 1:(J[1]-1)){ jj <- J[1]-i; for(j in jj:1){ tr2[i,j] <- tr2[i,j+1]/dev[j] } }
+   aux2 <- tr2
+
+   list(orig = tr, cred = aux, standard = aux2) #The original and the credibility transformed triangle.
+
+ }
> # E.g.
> # BackwardDev(Upper.Triangle(Trs[[k]]), FCred, k)
>
> # ======================
> # Calculate Pearson residuals and scale parameter.
> # Input trianlges: original triangle and backwards adjusted triangle. (cumulative input)
> # ======================
> PearsonRes <- function(tr.orig, tr.back){
+
+   residual <- ( cum2incr(tr.orig) - cum2incr(tr.back) ) / sqrt( cum2incr(tr.back) ) #Pearson residuals.
+   J <- dim(residual)[1]
+   scaleparam <- sum(residual^2, na.rm = TRUE) / (choose(J+1,2)-(2*J-1))
+   residual.adj <- residual * sqrt(choose(J+1,2)/(choose(J+1,2)-(2*J-1)))
+
+   return(list(residual = residual, residual.adj = residual.adj, scaleparam = scaleparam))
+
+ }
> # E.g.
> # k <- 1
> # Tr <- BackwardDev(Upper.Triangle(Trs[[k]]), FCred)
> # Pear <- PearsonRes(Tr$orig, Tr$cred)
> # PearsonRes(Tr$orig, Tr$standard)
>
> # ======================
> # Generate one sample of ultimate claims (for each contract year).
> # Over-dispersed Poisson model.
> # Inputs:
> #   - tr.orig the original upper triangle, tr.back the backwards adjustet triangle. (cumulative input)
```

```
> # ======================
> One.odpois.boot <- function(tr.orig, tr.back){
+
+   Pear <- PearsonRes(tr.orig, tr.back)
+   #PearsonRes uses cumulative input. After that we turn the triangles into increments.
+   tr.orig <- cum2incr(tr.orig)
+   tr.back <- cum2incr(tr.back)
+
+   J <- dim(tr.orig)[1]
+   samp <- Pear$residual.adj[!is.na(Pear$residual.adj)]
+   res.star <- matrix(NA, J, J)
+   aux <- sapply( J:1, function(i){ sample( samp, size = i, replace = TRUE ) })
+   for(j in 1:J) res.star[1:(J-j+1),j] <- aux[[j]]
+   tr.new <- res.star*sqrt(tr.back) + tr.back
+   tr.new <- incr2cum(tr.new)
+
+   f <- sapply(1:(J-1), function(j) sum(tr.new[1:(J-j),j+1])/sum(tr.new[1:(J-j),j]))
+   #Set NA values to 1.
+   f[which(is.na(f))] <- 1
+   fulltr.new <- tr.new
+   for(j in 1:(J-1)) fulltr.new[(J-j+1):J, j+1] <- fulltr.new[(J-j+1):J, j] * f[j]
+   fulltr.new <- cum2incr(fulltr.new)
+
+   #Generate over-dispersed Poisson sample.
+   incr.samp <- sapply(2:J, function(i){
+     sapply(fulltr.new[i,(J-i+2):J]/Pear$scaleparam, function(l) rpois(1,l))*Pear$scaleparam
+   })
+   res <- as.numeric(lapply(incr.samp, function(x) sum(x, na.rm = TRUE)))
+
+   return(res)
+
+ }
> # ======================
> # As a function of a cumularive triangle, calculate IBNRs on bootstrap basis.
> # Design to apply for credibility model.
> # Although applicable for standard 'od.pois' bootstrap.
> # Triangles should have the same dimensions (accident-development).
> # Input:
> #   - Data, type, grpcode (integer)
> #   - result of f.cred
> #   - model can be "standard.bootstrap.od.pois", "cred.bootstrap.od.pois"
> # ======================
>
> OwnBootstrap <- function(Data, type, grpcode, FCred, model="standard.bootstrap.od.pois"){
+
+   tr <- Get.Triangle(Data, type, grpcode)
+   k <- which(unique(Data[[type]]$GRCODE) == grpcode)
+   Tr <- BackwardDev(Upper.Triangle(tr), FCred, k)
+   J <- dim(Tr$orig)[1]
+
+   if(model == "standard.bootstrap.od.pois"){
+     res <- try( replicate(BS.R,One.odpois.boot(Tr$orig,Tr$standard)), silent = T )
+   }
+   if(model == "cred.bootstrap.od.pois"){
+     res <- try( replicate(BS.R,One.odpois.boot(Tr$orig,Tr$cred)), silent = T )
+   }
+
+   latest <- sum(sapply(1:J, function(i) tr[i,J-i+1]))
+   IBNRs <- apply(res,2, function(x) sum(x, na.rm=T))
+   real.IBNR <- sum(tr[,J]) - latest
+   res <- list(latest = latest, IBNRs = IBNRs, real.IBNR = real.IBNR)
+
+   return(res)
```

```
+
+ }
> # E.g.
> # FCred <- f.cred(Data,type)
> # grpcode = unique(Data[[type]]$GRCODE)[67]
> # res1 = OwnBootstrap(Data, type, grpcode, FCred, "standard.bootstrap.od.pois")
> # res2 = OwnBootstrap(Data, type, grpcode, FCred, "cred.bootstrap.od.pois")
>
>
> # ============================================================================
> # Bootstrapping the Munich Chain Ladder algorithm.
> # Goal: produce distributions of ultimate claims instead of pure point estimations.
> # ============================================================================
> # Example.
> # library(ChainLadder)
> # grpcode <- 353
> # Data <- Get.Data(Folder.CAS)
> # #Triangles are cumulative!
> # #Paid-incurred triangles. Apply the Munich Chain Ladder function.
> # Tr.p <- Get.Triangle(Data, 'comauto', grpcode,"CumPaidLoss")
> # Tr.i <- Get.Triangle(Data, 'comauto', grpcode,"IncurLoss")
> # bootstrap.result <- MCLbootstrap(Tr.p,Tr.i,10)
>
> # ============================================================================
> # Bootstrap Munich Chain Ladder.
> # Inputs:
> #   - paid-incurred cumulative triangles (total if available): Tr.p, Tr.i
> #   - bootMCL.R number of bootstrap steps
> # ============================================================================
> MCLbootstrap <- function(Tr.p, Tr.i, bootMCL.R){
+
+   Tr.i.complete <- Tr.i
+   Tr.p.complete <- Tr.p
+   Tr.i <- Upper.Triangle(Tr.i)
+   Tr.p <- Upper.Triangle(Tr.p)
+   mcl <- MunichChainLadder(Tr.p,Tr.i)
+   #Dimension of the quadrangles.
+   n <- dim(Tr.p)[1]
+   m <- dim(Tr.p)[2]
+   resi <- data.frame(mcl$PaidResiduals, mcl$IncurredResiduals, mcl$QResiduals, mcl$QinverseResiduals)
+   resi <- data.frame(resi)
+
+   #Parameter sets which remain unchanged across the iterations.
+   sig.P <- c(rep(mcl$MackPaid$sigma, each=n), rep(NA,each=n))
+   tau.P <- c(rep(mcl$rhoP.sigma, each=n))
+   sig.I <- c(rep(mcl$MackIncurred$sigma, each=n), rep(NA,each=n))
+   tau.I <- c(rep(mcl$rhoI.sigma, each=n))
+   C.P <- as.numeric(Tr.p)
+   C.I <- as.numeric(Tr.i)
+   f.P <- rep(mcl$MackPaid$f, each=n)
+   f.I <- rep(mcl$MackIncurred$f, each=n)
+   q.inv.f <- rep(mcl$qinverse.f, each=n)
+   q.f <- rep(mcl$q.f, each=n)
+
+   #Residuals are of NA values on the diagonals. Where possible, exchange NA with direct calculation.
+   #See the formulae for instance on page 12 in LV2010.
+   aux.ind <- which(is.na(resi$mcl.QinverseResiduals))
+   resi$mcl.QinverseResiduals[aux.ind] <- ((C.I/C.P - q.inv.f)/tau.P*sqrt(C.P))[aux.ind]
+   resi$mcl.QResiduals[aux.ind] <- ((C.P/C.I - q.f)/tau.I*sqrt(C.I))[aux.ind]
+
+   #Correct the bootstrap bias by sqrt(n-j / n-j-1). Correction only for j=1..n-2.
+   bias.correct <- sapply((n-rep(1:n,each=n)) / (n-rep(1:n,each=n)-1), function(x) if(x==Inf || x==0){1}else{x})
+   resi <- data.frame(apply(resi, 2, function(x) x*bias.correct))
```

```
+
+   #The iterative loop starts here.
+   #Random sample from residuals. Result is a pseudo sample.
+   samp.ind <- which(!is.na(resi[,1]))
+   Samp <- lapply(1:bootMCL.R, function(x) sample(samp.ind,length(samp.ind),replace = TRUE))
+
+   boot.res <- lapply(Samp, function(samp){ MCLbootstrap.iteration(resi, samp.ind, samp, Tr.p, Tr.i, mcl) })
+   UltimateInc <- sapply(1:bootMCL.R, function(k){ boot.res[[k]]$Incurred[((m-1)*n+1):(m*n)] })
+   UltimatePaid <- sapply(1:bootMCL.R, function(k){ boot.res[[k]]$Paid[((m-1)*n+1):(m*n)] })
+
+   reserve <- data.frame(ReservePaid.Boot = apply(UltimatePaid,1, mean) - Tr.p[n+(n-1)*((n-1):0)],
+                         ReservePaid.MCL = summary(mcl)$ByOrigin$'Ult. Paid' - summary(mcl)$ByOrigin$'Latest Paid',
+                         ReservePaid.real = Tr.p.complete[(n*(m-1)+1):(n*m)] - Tr.p[n+(n-1)*((n-1):0)],
+                         ReserveIncurred.Boot = apply(UltimateInc, 1, mean) - Tr.i[n+(n-1)*((n-1):0)],
+                         ReserveIncurred.MCL = summary(mcl)$ByOrigin$'Ult. Incurred' - summary(mcl)$ByOrigin$'Latest Incur
+                         ReserveIncurred.real = Tr.i.complete[(n*(m-1)+1):(n*m)] - Tr.i[n+(n-1)*((n-1):0)],
+                         LatestPaid = Tr.p[n+(n-1)*((n-1):0)],
+                         LatestIncurred = Tr.i[n+(n-1)*((n-1):0)]
+   )
+
+   out <- list(UltimateIncurred = UltimateInc, UltimatePaid = UltimatePaid,
+               MCL.Ult.Incurred = summary(mcl)$ByOrigin$'Ult. Incurred',
+               MCL.Ult.Paid = summary(mcl)$ByOrigin$'Ult. Paid',
+               Reserves = reserve)
+
+   return(out)
+
+ }
> # ============================================================================
> # One iterative step in bootstrapping the munich chain ladder residuals,
> #  then calculating the missing elements in paid-incurred triangles.
> # Inputs:
> #   - dataframe of residuals (outcomes of MunichChainLadder)
> #   - row indices in the dataframe which are substituted with another randomly chosen row (samp.ind)
> #   - indices in order is samp.ind, bootstraps residuals
> #   - original paid-incurred triangles
> #   - Truncate: MCL development factors (lambda.I and lambda.P here) if they exceed 1e+3.
> #       The reason is the very low sigma and tau parameters, unreliablem.
> #       It is basically equivalent changing to simple chain ladder in the latest development years.
> # ============================================================================
> MCLbootstrap.iteration <- function(resi, samp.ind, samp, Tr.p, Tr.i, mcl, Truncate=TRUE){
+
+   #Dimension of the quadrangles.
+   n <- dim(Tr.p)[1]
+   m <- dim(Tr.p)[2]
+
+   #Parameter sets which remain unchanged across the iterations.
+   sig.P <- c(rep(mcl$MackPaid$sigma, each=n), rep(NA,each=n))
+   tau.P <- c(rep(mcl$rhoP.sigma, each=n))
+   sig.I <- c(rep(mcl$MackIncurred$sigma, each=n), rep(NA,each=n))
+   tau.I <- c(rep(mcl$rhoI.sigma, each=n))
+   C.P <- as.numeric(Tr.p)
+   C.I <- as.numeric(Tr.i)
+   f.P <- rep(mcl$MackPaid$f, each=n)
+   f.I <- rep(mcl$MackIncurred$f, each=n)
+   q.inv.f <- rep(mcl$qinverse.f, each=n)
+   q.f <- rep(mcl$q.f, each=n)
+
+   #Step 1. One iteration loop starts here.
+   resi.B <- data.frame(resi)
+   resi.B[samp.ind,] <- resi[samp,]
+   #Alternative Pearson residuals.
+   r.P.B <- resi.B$mcl.PaidResiduals
```

```
+   r.Qinv.B <- resi.B$mcl.QinverseResiduals
+   r.Q.B <- resi.B$mcl.QResiduals
+   r.I.B <- resi.B$mcl.IncurredResiduals
+
+   #Step 2. Calculate pseudo samples from the triangles.
+   F.P.B <- r.P.B*sig.P/sqrt(C.P) + f.P
+   F.P.B[n*(m-1)+1] <- f.P[n*(m-1)+1]
+   #The 2nd last factor changed to the original development factor instead of NA. As well as for the other one below.
+   F.I.B <- r.I.B*sig.I/sqrt(C.I) + f.I
+   F.I.B[n*(m-1)+1] <- f.I[n*(m-1)+1]
+   Q.B <- r.Q.B*tau.I/sqrt(C.I) + q.f
+   Qinv.B <- r.Qinv.B*tau.P/sqrt(C.P) + q.inv.f
+
+   #Step 3. CP and CI weighted average of bootstrap paid and incurred development factors.
+   f.P.B <- sapply(1:(m-1), function(j) sum((F.P.B*C.P)[((j-1)*n+1):((n-1)*j)])/sum(C.P[((j-1)*n+1):((n-1)*j)]) )
+   f.P.B <- c(f.P.B,1)
+   if( length(which(is.na(f.P.B)))>0 ){f.P.B[which(is.na(f.P.B))]<-1}
+   f.I.B <- sapply(1:(m-1), function(j) sum((F.I.B*C.I)[((j-1)*n+1):((n-1)*j)])/sum(C.I[((j-1)*n+1):((n-1)*j)]) )
+   f.I.B <- c(f.I.B,1)
+   if( length(which(is.na(f.I.B)))>0 ){f.I.B[which(is.na(f.I.B))]<-1}
+   #Indices!
+   q.B <- sapply(1:m, function(j) sum((Q.B*C.I)[((j-1)*n+1):((n-1)*j+1)])/sum(C.I[((j-1)*n+1):((n-1)*j+1)]) )
+   q.inv.B <- sapply(1:m, function(j) sum((Qinv.B*C.P)[((j-1)*n+1):((n-1)*j+1)])/sum(C.P[((j-1)*n+1):((n-1)*j+1)]) )
+
+   #Step 4. Calculate correlation coefficients.
+   rho.P.B <- sum(r.Qinv.B*r.P.B, na.rm = TRUE) / sum(r.Qinv.B^2, na.rm = TRUE)
+   rho.I.B <- sum(r.Q.B*r.I.B, na.rm = TRUE) / sum(r.Q.B^2, na.rm = TRUE)
+
+   #Step 5. Calculate sigma and tau variances.
+   sig.P.B <- sapply(1:(m-1), function(j){
+     aux.ind <- ((j-1)*n+1):((n-1)*j)
+     sum(C.P[aux.ind]*(F.P.B - rep(f.P.B,each=n))[aux.ind]^2)/(n-j-1)
+   })
+   if(length(which(is.na(sig.P.B)))>0){sig.P.B[which(is.na(sig.P.B))] <- min(sig.P.B, na.rm = T)}
+   sig.P.B <- c(sig.P.B, min(sig.P.B))
+
+   sig.I.B <- sapply(1:(m-1), function(j){
+     aux.ind <- ((j-1)*n+1):((n-1)*j)
+     sum(C.I[aux.ind]*(F.I.B - rep(f.I.B,each=n))[aux.ind]^2)/(n-j-1)
+   })
+   if(length(which(is.na(sig.I.B)))>0){sig.I.B[which(is.na(sig.I.B))] <- min(sig.I.B, na.rm = T)}
+   sig.I.B <- c(sig.I.B, min(sig.I.B))
+
+   tau.P.B <- sapply(1:(m-1), function(j){
+     aux.ind <- ((j-1)*n+1):((n-1)*j)
+     sum(C.P[aux.ind]*(Qinv.B - rep(q.inv.B,each=n))[aux.ind]^2)/(n-j-1)
+   })
+   if(length(which(is.na(tau.P.B)))>0){tau.P.B[which(is.na(tau.P.B))] <- min(tau.P.B, na.rm = T)}
+   tau.P.B <- c(tau.P.B, min(tau.P.B))
+
+   tau.I.B <- sapply(1:(m-1), function(j){
+     aux.ind <- ((j-1)*n+1):((n-1)*j)
+     sum(C.I[aux.ind]*(Q.B - rep(q.B,each=n))[aux.ind]^2)/(n-j-1)
+   })
+   if(length(which(is.na(tau.I.B)))>0){tau.I.B[which(is.na(tau.I.B))] <- min(tau.I.B, na.rm = T)}
+   tau.I.B <- c(tau.I.B, min(tau.I.B))
+
+   #Step 6. Calculate the bootstrap development factors.
+   lambda.P.B <- rep(f.P.B, each=n) + rho.P.B * rep(sig.P.B/tau.P.B, each=n) * (Qinv.B - rep(q.inv.B,each=n))
+   lambda.I.B <- rep(f.I.B, each=n) + rho.I.B * rep(sig.I.B/tau.I.B, each=n) * (Q.B - rep(q.B,each=n))
+   if(Truncate==TRUE){
+     if(length(which(abs(lambda.I.B)>1e+3)) > 0){
+       lambda.I.B[which(abs(lambda.I.B)>1e+3)] <- rep(f.I.B, each=n)[which(abs(lambda.I.B)>1e+3)]
```

```
+     }
+     if(length(which(abs(lambda.P.B)>1e+3)) > 0){
+       lambda.P.B[which(abs(lambda.P.B)>1e+3)] <- rep(f.P.B, each=n)[which(abs(lambda.P.B)>1e+3)]
+     }
+   }
+
+   if(length(is.na(lambda.I.B))>0){
+     lambda.I.B[which(is.na(lambda.I.B))] <- rep(f.I.B, each=n)[which(is.na(lambda.I.B))]
+   }
+   if(length(is.na(lambda.P.B))>0){
+     lambda.P.B[which(is.na(lambda.P.B))] <- rep(f.P.B, each=n)[which(is.na(lambda.P.B))]
+   }
+
+   #Step 7. Simulate future payments and incurred claims.
+   X.P <- as.numeric(Tr.p)
+   X.I <- as.numeric(Tr.i)
+
+   for(jj in 1:(m-1)){
+
+     ###jj = 1 #j goes from 1 to m-1
+     ind <- sapply(1:n, function(i) ((n-1)*i+1 + (jj-1)*n))[1:(n-jj)]
+
+     X.P[ind+n] <- sapply(1:length(ind), function(i){
+       rnorm(1, mean = lambda.P.B[ind[i]]*X.P[ind[i]], sd = sig.P.B[(n-length(ind)):(n-1)][i]*sqrt(abs(X.P[ind[i]])))
+     })
+
+     X.I[ind+n] <- sapply(1:length(ind), function(i){
+       rnorm(1, mean = lambda.I.B[ind[i]]*X.I[ind[i]], sd = sig.I.B[(n-length(ind)):(n-1)][i]*sqrt(abs(X.I[ind[i]])))
+     })
+
+     Q.B[ind+n] <- X.P[ind+n] / X.I[ind+n]
+     Qinv.B[ind+n] <- X.I[ind+n] / X.P[ind+n]
+     lambda.P.B <- rep(f.P.B, each=n) + rho.P.B * rep(sig.P.B/tau.P.B, each=n) * (Qinv.B - rep(q.inv.B,each=n))
+     lambda.I.B <- rep(f.I.B, each=n) + rho.I.B * rep(sig.I.B/tau.I.B, each=n) * (Q.B - rep(q.B,each=n))
+     if(Truncate==TRUE){
+       if(length(which(abs(lambda.I.B)>1e+3)) > 0){
+         lambda.I.B[which(abs(lambda.I.B)>1e+3)] <- rep(f.I.B, each=n)[which(abs(lambda.I.B)>1e+3)]
+       }
+       if(length(which(abs(lambda.P.B)>1e+3)) > 0){
+         lambda.P.B[which(abs(lambda.P.B)>1e+3)] <- rep(f.P.B, each=n)[which(abs(lambda.P.B)>1e+3)]
+       }
+     }
+     if(length(is.na(lambda.I.B))>0){
+       lambda.I.B[which(is.na(lambda.I.B))] <- rep(f.I.B, each=n)[which(is.na(lambda.I.B))]
+     }
+     if(length(is.na(lambda.P.B))>0){
+       lambda.P.B[which(is.na(lambda.P.B))] <- rep(f.P.B, each=n)[which(is.na(lambda.P.B))]
+     }
+
+   }
+
+   return(list(Incurred = X.I, Paid = X.P))
+
+ }
> # ================================================================================
> # Example Quarg and Mack.
> # ================================================================================
> # Tr.p <- c(576, 866, 1412, 2286, 1868, 1442, 2044,
> #           1804, 1948, 3758, 5292, 3778, 4010, NA,
> #           1970, 2162, 4252, 5724, 4648, NA, NA,
> #           2024, 2232, 4416, 5850, NA, NA, NA,
> #           2074, 2284, 4494, NA, NA, NA, NA,
> #           2102, 2348, NA, NA, NA, NA, NA,
```

```
> #           2131, NA, NA, NA, NA, NA, NA)
> # Tr.p <- matrix(Tr.p,7,7)
> #
> # Tr.i <- c(978, 1844, 2904, 3502, 2812, 2642, 5022,
> #           2104, 2552, 4354, 5958, 4882, 4406, NA,
> #           2134, 2466, 4698, 6070, 4852, NA, NA,
> #           2144, 2480, 4600, 6142, NA, NA, NA,
> #           2174, 2508, 4644, NA, NA, NA, NA,
> #           2182, 2454, NA, NA, NA, NA, NA,
> #           2174, NA, NA, NA, NA, NA, NA)
> # Tr.i <- matrix(Tr.i,7,7)
> # ============================================================================
>
>
> # ============================================================================
> # Part II. Codes from Glenn Meyers (with minor modifications).
> # Source: www.casact.org/pubs/monographs/meyers/Monograph_Tables_and_Scripts.xlsx
> # Accessed: 6 May 2018.
> # ============================================================================
>
> # ============================================================================
> # The Correlated Chain Ladder model.
> # Bayesian model of incurred loss data.
> # ============================================================================
>
> # Script to run the CCL Model on data from the CAS Loss Reserve Database
> # To run the LCL model, set prior for rho ~ dunif(-.00001,00001) in JAGS script
> # by Glenn Meyers
> #
> # setwd(Folder.CAS)
> #
> # user inputs
> #
> # insurer.data="comauto_pos.csv"
> # insurer.data="ppauto_pos.csv"
> # insurer.data="wkcomp_pos.csv"
> # insurer.data="othliab_pos.csv"
> # insurer.data="prodliab_pos.csv"
> # insurer.data="medmal_pos.csv"
>
> # losstype="incloss"  #"incloss" if incurred loss or "cpdloss" if paid loss
>
> # ============================================================================
> # ============================================================================
> # Main CCL function.
> function.CCL <- function(type = "comauto", grpcode = "353", outfile = "outCCL.csv",
+                          losstype = "incloss", Folder.CAS){
+
+   insurer.data <- paste(type,"pos.csv",sep="_")
+   #losstype can be "incloss" for incurred or "cpdloss" for paid
+   nburn <- 10000
+
+   # ============================================================================
+   # ============================================================================
+   # JAGS script
+   #
+   modelString = "model {
+   mu[1]<-alpha[w[1]]+beta[d[1]]
+   logloss[1]~dnorm(mu[1],1/sig2[1])
+   for (i in 2:length(w)){
+   mu[i]<-alpha[w[i]]+beta[d[i]]+rho*(logloss[i-1]-mu[i-1])*wne1[i]
+   logloss[i]~dnorm(mu[i],1/sig2[i])
+   }
```

```
+   #
+   # set up sig2
+   #
+   for (i in 1:length(w)){
+   sig2[i]<-sigd2[d[i]]
+   }
+   for (j in 1:10){
+   sigd2[j]<-sum(a[j:10])
+   }
+   for (k in 1:10){
+   a[k]~dunif(0.000001,1)
+   }
+   #
+   # specify priors
+   #
+   for (i in 1:numlev){
+   alpha[i]~dnorm(log(premium[i])+logelr,.1)
+   }
+   logelr~dunif(-1.5,0.5)
+   #
+   for (i in 1:9){
+   beta[i]~dunif(-5,5)
+   }
+   beta[10]<-0
+   rho~dunif(-1,1)
+   # rho~dunif(-.00001,.00001) # Use for LCL model
+ }"
+
+   # ============================================================================
+   # ============================================================================
+   # get data
+   #
+   a <- read.csv(paste(Folder.CAS,insurer.data,sep="/"))
+
+   # ============================================================================
+   # ============================================================================
+   # function to get Schedule P triangle data given ins group and line of business
+   #
+   ins.line.data <- function(g.code){
+     b=subset(a,a$GRCODE==g.code)
+     name=b$GRNAME
+     grpcode=b$GRCODE
+     w=b$AccidentYear
+     d=b$DevelopmentLag
+     cum_incloss=b[,6]
+     cum_pdloss=b[,7]
+     bulk_loss=b[,8]
+     dir_premium=b[,9]
+     ced_premium=b[,10]
+     net_premium=b[,11]
+     single=b[,12]
+     posted_reserve97=b[,13]
+     # get incremental paid losses - assume data is sorted by ay and lag
+     inc_pdloss=numeric(0)
+     for (i in unique(w)){
+       s=(w==i)
+       pl=c(0,cum_pdloss[s])
+       ndev=length(pl)-1
+       il=rep(0,ndev)
+       for (j in 1:ndev){
+         il[j]=pl[j+1]-pl[j]
+       }
+       inc_pdloss=c(inc_pdloss,il)
```

```
+     }
+     data.out=data.frame(grpcode,w,d,net_premium,dir_premium,ced_premium,
+                         cum_pdloss,cum_incloss,bulk_loss,inc_pdloss,single,posted_reserve97)
+     return(data.out)
+   }
+
+   # ============================================================================
+   # ============================================================================
+   # read and aggregate the insurer data and
+   # set up training and test data frames
+   #
+   cdata=ins.line.data(grpcode)
+   set.seed(12345)
+   w=cdata$w-1987
+   d=cdata$d
+
+   # ============================================================================
+   # ============================================================================
+   # sort the data in order of d, then w within d
+   #
+   o1=100*d+w
+   o=order(o1)
+   w=w[o]
+   d=d[o]
+   premium=cdata$net_premium[o]
+   cpdloss=cdata$cum_pdloss[o]
+   cpdloss=pmax(cpdloss,1)
+   incloss=cdata$cum_incloss[o]-cdata$bulk_loss[o]
+   incloss=pmax(incloss,1)
+   wne1=ifelse(w==1,0,1)
+   adata=data.frame(grpcode,w,d,premium,cpdloss,incloss,wne1)
+   rdata=subset(adata,(adata$w+adata$d)<12)
+   numw=length(unique(rdata$w))
+   rdata=subset(adata,(adata$w+adata$d)<12)
+   if(losstype=="incloss") rloss=rdata$incloss else rloss=rdata$cpdloss
+   if(losstype=="incloss") aloss=adata$incloss else aloss=adata$cpdloss
+
+   # ============================================================================
+   # ============================================================================
+   # Initialize JAGS model
+   #
+   inits1=list(.RNG.name= "base::Wichmann-Hill",
+               .RNG.seed= 12341)
+   inits2=list(.RNG.name= "base::Marsaglia-Multicarry",
+               .RNG.seed= 12342)
+   inits3=list(.RNG.name= "base::Super-Duper",
+               .RNG.seed= 12343)
+   inits4=list(.RNG.name= "base::Mersenne-Twister",
+               .RNG.seed= 12344)
+   data.for.jags=list(premium= premium[1:10],
+                      logloss = log(rloss),
+                      numlev  = numw,
+                      w       = rdata$w,
+                      wne1    = rdata$wne1,
+                      d       = rdata$d)
+
+   # ============================================================================
+   # ============================================================================
+   # run the model
+   #
+   # call to run.jags #
+   nthin=2
+   maxpsrf=2
```

```
+   while (maxpsrf>1.05){
+     nthin=nthin*2
+     print(paste("nthin =",nthin))
+     jagout=run.jags(model=modelString,monitor=c("alpha","beta[1:9]","sigd2","rho"),
+                     data=data.for.jags,n.chains=4,method="parallel",
+                     inits=list(inits1,inits2,inits3,inits4),thin=nthin,silent.jags=F,
+                     plots=TRUE,burnin=nburn,sample=2500,psrf.target=1.05)
+     gelman=gelman.diag(jagout)
+     maxpsrf=max(gelman$psrf[,1])
+     print(paste("maxpsrf =",maxpsrf))
+   }
+   print(jagout$timetaken)
+
+   # ============================================================================
+   # ============================================================================
+   # extract information from jags output to process in R
+   #
+   b=as.matrix(jagout$mcmc)
+   alpha=b[,1:10]
+   beta=cbind(b[,11:19],rep(0,dim(b)[1]))
+   rho=b[,20]
+   sigd2=b[,21:30]
+
+   # ============================================================================
+   # ============================================================================
+   # simulate loss statistics by accident year reflecting total risk for the JAGS model
+   #
+   set.seed(12345)
+   Premium=subset(rdata,rdata$d==1)$premium
+   at.wd10=matrix(0,dim(b)[1],10)
+   ss.wd10=rep(0,10)
+   ms.wd10=rep(0,10)
+   mu.wd11mw=matrix(0,dim(b)[1],10)
+   mu.wd10=matrix(0,dim(b)[1],10)
+   at.wd10[,1]=rep(rloss[55],dim(b)[1])
+   mu.wd10[,1]=alpha[,1]+beta[,10]
+   for (w in 2:10){
+     mu.wd10[,w]=alpha[,w]+beta[,10]+rho*(log(at.wd10[,w-1])-mu.wd10[,w-1])
+     for (i in 1:dim(b)[1]){
+       at.wd10[i,w]=rlnorm(1,mu.wd10[i,w],sqrt(sigd2[i,10]))
+     }
+   }
+   ms.wd10[1]=mean(at.wd10[,1])
+   for (w in 2:10){
+     ms.wd10[w]=mean(at.wd10[,w])
+     ss.wd10[w]=sd(at.wd10[,w])
+   }
+   Pred.CCL=rowSums(at.wd10)
+   ms.td10=mean(Pred.CCL)
+   ss.td10=sd(Pred.CCL)
+   CCL.Estimate=round(ms.wd10)
+   CCL.S.E.=round(ss.wd10)
+   CCL.CV=round(CCL.S.E./CCL.Estimate,4)
+   act=sum(subset(aloss,adata$d==10)[1:10])
+   pct.CCL=sum(Pred.CCL<=act)/length(Pred.CCL)*100
+
+   #
+   # put CCL accident year statistics into a data frame
+   #
+   W=c(1:10,"Total")
+   CCL.Estimate=c(CCL.Estimate,round(ms.td10))
+   CCL.S.E.=c(CCL.S.E.,round(ss.td10))
+   CCL.CV=c(CCL.CV,round(ss.td10/ms.td10,4))
```

```
+   Premium=c(Premium,sum(Premium))
+   Outcome=subset(aloss,adata$d==10)
+   Outcome=c(Outcome,sum(Outcome))
+   Group=rep(grpcode,11)
+   CCL.Pct=c(rep(NA,10),pct.CCL)
+   risk=data.frame(W,Premium,CCL.Estimate,CCL.S.E.,CCL.CV,Outcome,CCL.Pct)
+   #print(risk)
+   #write.csv(risk,file=outfile,row.names=F)
+   #par(mfrow=c(1,1))
+   #hist(rho,main="",xlim=c(-1,1),xlab=expression(rho))
+   #hist(Pred.CCL,main="Predictive Distribution of Outcomes",xlab="")
+
+   return(list(risk = risk, jagout = jagout, at.wd10 = at.wd10))
+
+   }
> #
> # optional diagnostics
> #
> # crosscorr.plot(jagout$mcmc)
> # traceplot(jagout$mcmc)
> # print(gelman.diag(jagout))
> # gelman.plot(jagout)
> # print(summary(jagout)[[1]][,1:2])
>
>
>
> # ============================================================================
> # Self-developed scripts based on the functions above.
> # ============================================================================
>
> #res = function.CCL(type = "comauto", grpcode = "353", outfile = "outCCL.csv", losstype = "incloss", Folder.CAS)
> #scen = res$at.wd10
> #apply(scen, 2, function(x) mean(x,na.rm=T))
>
> # ============================================================================
> # Codes from Glenn Meyers.
> # The Correlated Incremental Trend model.
> # Bayesian model of incurred loss data.
> # ============================================================================
> #
> # Script to run the CIT Model on data from the CAS Loss Reserve Database
> # To run the LIT model, set prior for rho ~ dunif(-.00001,00001) in JAGS script
> # by Glenn Meyers
> #
> #rm(list = ls())      # clear workspace"
> #
> # user inputs
> #
> #insurer.data="comauto_pos.csv"
> #insurer.data="ppauto_pos.csv"
> #insurer.data="wkcomp_pos.csv"
> #insurer.data="othliab_pos.csv"
> #insurer.data="prodliab_pos.csv"
> #insurer.data="medmal_pos.csv"
>
> # ============================================================================
> # ============================================================================
> # Main CIT function.
>
> function.CIT <- function(type = "comauto", grpcode = "353", outfile = "outCIT.csv",
+                          losstype = "paidloss", Folder.CAS){
+
+   insurer.data <- paste(type,"pos.csv",sep="_")
```

```
+   #grpcode="353"
+   nburn=50000
+   #
+
+   # ============================================================================
+   # ============================================================================
+   # JAGS script
+   #
+   modelString = "model {
+   mu[1]<-alpha[w[1]]+beta[d[1]]+tau*(w[1]+d[1]-1)
+   logz[1]~dnorm(mu[1],1/sig2[1])
+   z[1]<-exp(logz[1])
+   loss[1]~dnorm(z[1],deltam2)
+   for(i in 2:length(w)){
+   mu[i]<-alpha[w[i]]+beta[d[i]]+tau*(w[i]+d[i]-1)
+   logz[i]~dnorm(mu[i],1/sig2[i])
+   z[i]<-exp(logz[i])+rho*(loss[i-1]-z[i-1])*exp(tau)*wne1[i]
+   loss[i]~dnorm(z[i],deltam2)
+   }
+   #
+   # set up sig2
+   #
+   for (i in 1:length(w)){
+   sig2[i]<-sigd2[d[i]]
+   }
+   sigd2[1]~dunif(.000001,0.5)
+   for (j in 2:10){
+   sigd2[j]~dunif(sigd2[j-1],sigd2[j-1]+.1)         # control growth of sigma
+   }
+   #
+   # specify priors
+   #
+   for (i in 1:numlev){
+   alpha[i]~dnorm(log(premium[i])+logelr,0.1)       # std dev of alpha = 1/sqrt(.1) = 3.16
+   }
+   for (i in 1:4){
+   beta[i]~dunif(0,10)
+   }
+   for (i in 5:9){
+   beta[i]~dunif(0,beta[i-1])                       # force beta to decrease for d > 4
+   }
+   beta[10]<-0
+   rho~dunif(-1,1)
+   # rho~dunif(-.00001,.00001) # Use for LIT model
+   logelr~dunif(-5,1)
+   tau~dnorm(0,1000)                                    # std dev of tau = 1/sqrt(1000) = 0.0316
+   delta~dunif(0,sum(premium)/10)
+   deltam2<-1/delta^2
+ }"
+
+   # ============================================================================
+   # ============================================================================
+   # get data
+   #
+   a <- read.csv(paste(Folder.CAS,insurer.data,sep="/"))
+
+   # ============================================================================
+   # ============================================================================
+   #
+   # function to get Schedule P triangle data given ins group and line of business
+   #
+   ins.line.data=function(g.code){
+     b=subset(a,a$GRCODE==g.code)
```

```
+     name=b$GRNAME
+     grpcode=b$GRCODE
+     w=b$AccidentYear
+     d=b$DevelopmentLag
+     cum_incloss=b[,6]
+     cum_pdloss=b[,7]
+     bulk_loss=b[,8]
+     dir_premium=b[,9]
+     ced_premium=b[,10]
+     net_premium=b[,11]
+     single=b$Single
+     posted_reserve97=b[,13]
+     # get incremental paid losses - assume data is sorted by ay and lag
+     inc_pdloss=numeric(0)
+     for (i in unique(w)){
+       s=(w==i)
+       pl=c(0,cum_pdloss[s])
+       ndev=length(pl)-1
+       il=rep(0,ndev)
+       for (j in 1:ndev){
+         il[j]=pl[j+1]-pl[j]
+       }
+       inc_pdloss=c(inc_pdloss,il)
+     }
+     data.out=data.frame(grpcode,w,d,net_premium,dir_premium,ced_premium,
+                         cum_pdloss,cum_incloss,bulk_loss,inc_pdloss,single,posted_reserve97)
+     return(data.out)
+   }
+
+   # ============================================================================
+   # ============================================================================
+   #
+   # read and aggregate the insurer data and
+   # set up training and test data frames
+   cdata=ins.line.data(grpcode)
+   set.seed(12345)
+   w=cdata$w-1987
+   d=cdata$d
+
+   # ============================================================================
+   # ============================================================================
+   #
+   # sort the data in order of d, then w within d
+   #
+   o1=100*d+w
+   o=order(o1)
+   w=w[o]
+   d=d[o]
+   premium=cdata$net_premium[o]
+   ipdloss=cdata$inc_pdloss[o]
+   cpdloss=cdata$cum_pdloss[o]
+   wne1=ifelse(w==1,0,1)
+   adata=data.frame(grpcode,w,d,premium,ipdloss,cpdloss,wne1)
+   rdata=subset(adata,(adata$w+adata$d)<12)
+   numw=length(unique(rdata$w))
+   rdata=subset(adata,(adata$w+adata$d)<12)
+   rloss=rdata$ipdloss
+   aloss=adata$ipdloss
+   Premium=rdata$premium[1:10]
+
+   # ============================================================================
+   # ============================================================================
+   #
```

```
+   # Initialize JAGS model
+   #
+   inits1=list(.RNG.name= "base::Wichmann-Hill",
+               .RNG.seed= 12341)
+   inits2=list(.RNG.name= "base::Marsaglia-Multicarry",
+               .RNG.seed= 12342)
+   inits3=list(.RNG.name= "base::Super-Duper",
+               .RNG.seed= 12343)
+   inits4=list(.RNG.name= "base::Mersenne-Twister",
+               .RNG.seed= 12344)
+   data.for.jags=list('premium'= Premium,
+                      'loss'   = rloss,
+                      'numlev' = numw,
+                      'w'      = rdata$w,
+                      'wne1'   = rdata$wne1,
+                      'd'      = rdata$d)
+
+   # ============================================================================
+   # ============================================================================
+   #
+   # run the model
+   #
+   nthin=2
+   maxpsrf=2
+   while (maxpsrf>1.05){
+     nthin=nthin*2
+     print(paste("nthin =",nthin))
+     jagout=run.jags(model=modelString,
+                     monitor=c("alpha","beta[1:9]","sigd2","rho","delta","tau","logelr"),
+                     data=data.for.jags,n.chains=4,method="parallel",
+                     inits=list(inits1,inits2,inits3,inits4),thin=nthin,silent.jags=F,
+                     plots=TRUE,burnin=nburn,sample=2500,psrf.target=1.05)
+     gelman=gelman.diag(jagout)
+     maxpsrf=max(gelman$psrf[,1])
+     print(paste("maxpsrf =",maxpsrf))
+   }
+
+   # ============================================================================
+   # ============================================================================
+   #
+   # optional diagnostics
+   #
+   #library(coda)
+   #crosscorr.plot(jagout$mcmc)
+   # traceplot(jagout$mcmc)
+   # print(gelman.diag(jagout))
+   # gelman.plot(jagout)
+   #print(summary(jagout)[[1]][,1:2])
+
+   # ============================================================================
+   # ============================================================================
+   #
+   # extract information from jags output to process in R
+   #
+   b=as.matrix(jagout$mcmc)
+   alpha=b[,1:10]
+   beta=cbind(b[,11:19],rep(0,dim(b)[1]))
+   delta=b[,20]
+   logelr=b[,21]
+   rho=b[,22]
+   sigd2=b[,23:32]
+   tau=b[,33]
+
```

```
+   # ============================================================================
+   # ============================================================================
+   #
+   # simulate loss statistics by accident year reflecting total risk for the JAGS model
+   #
+   set.seed(12345)
+   at.wd10=matrix(0,dim(b)[1],10)
+   for (w in 1:10){
+     latest=sum(subset(rdata$ipdloss,rdata$w==w))
+     at.wd10[,w]=rep(latest,dim(b)[1])
+   }
+   library(ChainLadder)
+   itriangle=as.triangle(rdata,origin="w",dev="d",value="ipdloss")
+   z=matrix(0,10,10)
+   logz=matrix(0,10,10)
+   for (i in 1:dim(b)[1]){
+     m=matrix(0,10,10)
+     for (d in 2:9){
+       w=1
+       m[w,d]=alpha[i,w]+beta[i,d]+tau[i]*(w+d-1)
+       for (w in 2:(11-d)){
+         m[w,d]=alpha[i,w]+beta[i,d]+tau[i]*(w+d-1)
+         z[w,d]=exp(m[w,d])+rho[i]*(itriangle[w-1,d]-z[w-1,d])*exp(tau[i])
+       }
+     }
+     d=10
+     w=1
+     m[w,d]=alpha[i,w]+beta[i,d]+tau[i]*(w+d-1)
+     #
+     for (d in 2:10){
+       for (w in (12-d):10){
+         m[w,d]=alpha[i,w]+beta[i,d]+tau[i]*(w+d-1)
+         logz[w,d]=rnorm(1,m[w,d],sigd2[i,d])
+         z[w,d]=exp(m[w,d])+
+           rho[i]*(itriangle[w-1,d]-z[w-1,d])*exp(tau[i])
+         itriangle[w,d]=rnorm(1,z[w,d],delta[i])
+         at.wd10[i,w]=at.wd10[i,w]+itriangle[w,d]
+       }
+     }
+   }
+   ss.wd10=rep(0,10)
+   ms.wd10=rep(0,10)
+   ms.wd10[1]=mean(at.wd10[,1])
+   for (w in 2:10){
+     ms.wd10[w]=mean(at.wd10[,w])
+     ss.wd10[w]=sd(at.wd10[,w])
+   }
+   Pred.CIT=rowSums(at.wd10)
+   ms.td10=mean(Pred.CIT)
+   ss.td10=sd(Pred.CIT)
+   CIT.Estimate=round(ms.wd10)
+   CIT.SE=round(ss.wd10)
+   CIT.CV=round(CIT.SE/CIT.Estimate,4)
+   Outcome=subset(adata$cpdloss,adata$d==10)[1:10]
+   CIT.Pct=sum(Pred.CIT<=sum(Outcome))/length(Pred.CIT)*100
+
+   # ============================================================================
+   # ============================================================================
+   #
+   # put accident year statistics into a data frame
+   #
+   W=c(1:10,"Total")
+   CIT.Estimate=c(CIT.Estimate,round(ms.td10))
```

```
+   CIT.SE=c(CIT.SE,round(ss.td10))
+   CIT.CV=c(CIT.CV,round(ss.td10/ms.td10,4))
+   Premium=c(Premium,sum(Premium))
+   Outcome=c(Outcome,sum(Outcome))
+   Group=rep(grpcode,11)
+   CIT.Pct=c(rep(NA,10),CIT.Pct)
+   risk=data.frame(W,Premium,CIT.Estimate,CIT.SE,CIT.CV,Outcome,CIT.Pct)
+   #print(risk)
+
+   return(list(risk = risk, jagout = jagout, at.wd10 = at.wd10))
+
+   }
>
>
> # ============================================================================
> # ============================================================================
>
> # ============================================================================
> # Self-developed scripts based on the functions above.
> # ============================================================================
>
> # res = function.CIT(type = "comauto", grpcode = "353", outfile = "outCCL.csv", losstype = "incloss", Folder.CAS)
> # scen = res$at.wd10
> # apply(scen, 2, function(x) mean(x,na.rm=T))
>
>
> # ============================================================================
> # Part III. Perform calculations and save .RData.
> # ============================================================================
>
> # # Manual entry. Example calculation.
> # type <- "comauto" #select portfolio
> # normalize <- NULL #NULL if unscaled quadrangles / number N if total quadrangle sum is N
> # Folder.Main <- "~/path_to_main_folder/"
> #
> # # Folders and read in source code.
> # Folder.CAS <- paste(Folder.Main, "CAS_Data/", sep="") # NAIC data.
> # Folder.Codes <- paste(Folder.Main, "Codes/",sep="")
> # Folder.RDataSource <- paste(Folder.Main, "RDataSource/",sep="")
> # source(paste(Folder.Codes,"/supplementary_code.R",sep=""))
> #
> # # Calculation. Read in database.
> # Data <- Get.Data(Folder.CAS)
> # # Normalizing to 1 doesn't work with over-dispersed Poisson. 2nd constant is the total reserve (paid/inc).
> # # Put it in comment if not needed.
> # if(is.numeric(normalize)==TRUE){Data <- Normalize.Data(Data,normalize)}
> #
> # time0 <- Sys.time()
> # Res <- ResultSummary(Data = Data, type = type, companies = "All",
> #                      #Methods = c("boot.gamma","boot.od.pois","munich","bootstrap.munich","SemiSt"),
> #                      #Methods = c("boot.od.pois","standard.bootstrap.od.pois","cred.bootstrap.od.pois"),
> #                      Methods = c("munich"),
> #                      alphas = c(2/3, 0.9), Folder.CAS)
> # time1 <- Sys.time()
> #
> # # Save time used for calculation.
> # Res$TimeElapsed <- list(Begin = time0, End = time1, Used = time1-time0)
> # # save(Res, file=paste(Folder.RDataSource,"/",type,".RData",sep=""))
>
> # ============================================================================
> # THE END.
> # ============================================================================
```

>
>

Conflicts of Interest: The authors declare no conflict of interest.