#

# Script to get Risk Margin for CSR model with one line

# by Glenn Meyers

#

#

rm(list = ls()) # clear workspace")

t1=Sys.time()

#

# user inputs

#

insurer.data="~/Dropbox/CAS Loss Reserve Database/comauto\_pos.csv"

grpcode="353"

fixed.rate=0.04

risky.rate=0.10

TVaR.Range=9701:10000

#

# user technical inputs

#

outfile="outCSR.csv"

setwd("~/Dropbox/Risk Margin/Risk Margin - LaTeX") # Mac

#

library(rstan)

rstan\_options(auto\_write = TRUE)

options(mc.cores = parallel::detectCores())

library(parallel)

library(doParallel)

#

# Stan script for univariate model

#

scodeU = "

data{

real logprem[10];

real logloss[55];

int<lower=1,upper=10> w[55];

int<lower=1,upper=10> d[55];

}

parameters{

real r\_alpha[9];

real r\_beta[9];

real<lower=-1.5,upper=0.5> logelr;

real<lower=0,upper=1> a[10];

real gamma;

real delta;

}

transformed parameters{

real alpha[10];

real beta[10];

real speedup[10];

real sig2[10];

real sig[10];

real mu[55];

alpha[1]=0;

for (i in 2:10) alpha[i] = r\_alpha[i-1];

for (i in 1:9) beta[i] = 10\*r\_beta[i]-5;

beta[10] = 0;

speedup[1] = 1;

for (i in 2:10) speedup[i] = speedup[i-1]\*(1-gamma-(i-2)\*delta);

sig2[10] = a[10];

for (i in 1:9) sig2[10-i] = sig2[11-i]+a[i];

for (i in 1:10) sig[i] = sqrt(sig2[i]);

for (i in 1:55){

mu[i] = logprem[w[i]]+logelr+alpha[w[i]]+beta[d[i]]\*speedup[w[i]];

}

}

model {

for (i in 1:9) r\_alpha[i] ~ normal(0,3.162);

for (i in 1:9) r\_beta[i] ~ uniform(0,1);

for (i in 1:10) a[i] ~ uniform(0,1);

logelr ~ uniform(-1.5,0.5);

gamma ~ normal(0,0.05);

delta ~ normal(0,0.01);

for (i in 1:55) logloss[i] ~ normal(mu[i],sig[d[i]]);

}

"

#

# compile stan model

#

dummydata.uni=list(logprem = rep(8,10),

 logloss = rep(8,55),

 w = rep(1,55),

 d = rep(1,55))

fitU = stan(model\_code=scodeU,chains=0,data=dummydata.uni)

#

# get data

#

a=read.csv(insurer.data)

#

# function for 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)

adata=data.frame(grpcode,w,d,premium,cpdloss)

rdata=subset(adata,(adata$w+adata$d)<12)

numw=length(unique(rdata$w))

rdata=subset(adata,(adata$w+adata$d)<12)

rloss=rdata$cpdloss

aloss=adata$cpdloss

Premium=subset(rdata,rdata$d==1)$premium

#

# run the model

#

#

# initialization function for scodeU

#

initU=function(chain\_id){

 set.seed(12345+chain\_id)

 list(r\_alpha=rnorm(9,0,0.2),r\_beta=runif(9),a=runif(10),

 logelr=runif(1,-0.75,-0.5),gamma=rnorm(1,0,0.1),

 delta=rnorm(1,0,0.02))

}

pars.list=c("alpha","beta","gamma","delta","logelr","sig")

#

# univariate line 1

#

#

# data for univariate 1

#

data.u1=list(logprem = log(rdata$premium[1:10]),

 logloss = log(rloss),

 w = rdata$w,

 d = rdata$d)

#

# run the univariate model

#

set.seed(12345)

stan\_thin=1

stan\_iter=5000

Rhat\_target=1.05

max\_Rhat=2

while ((max\_Rhat > Rhat\_target)&(stan\_thin<65)){

 sflist <-

 mclapply(1:4, mc.cores = 4,

 function(i) stan(fit = fitU, data = data.u1,init=initU,

 seed = 12345,iter=stan\_iter,thin=stan\_thin,

 chains = 1, chain\_id = i, cores=4,pars=pars.list))

 fitU1=sflist2stanfit(sflist)

 fitU1\_summary=as.matrix(summary(fitU1)$summary)[,c(1,3,10)]

 mrh=subset(fitU1\_summary,is.na(fitU1\_summary[,3])==F)

 max\_Rhat=round(max(mrh[,3]),4)

 mean\_lp\_\_=round(fitU1\_summary[dim(fitU1\_summary)[1],1],2)

 print(paste("Maximum Rhat =",max\_Rhat," Mean lp\_\_ =",mean\_lp\_\_,

 "Thin =",stan\_thin))

 stan\_thin=2\*stan\_thin

 stan\_iter=2\*stan\_iter

}

#

# extract information from stan output to process in R

#

b=extract(fitU1)

alpha=b$alpha

beta=b$beta

gamma=b$gamma

delta=b$delta

logelr=b$logelr

sigma=b$sig

num.mcmc=length(logelr)

#

# get speedup rates by accident year

#

speedup=matrix(1,num.mcmc,10)

for (i in 2:10){

 speedup[,i]=speedup[,i-1]\*(1-gamma-(i-2)\*delta)

}

#

# simulate outcomes for d=10 using parallel processing

#

cl <- makeCluster(4)

registerDoParallel(cl)

at.wd10=foreach (k=1:length(gamma),.combine=rbind) %dopar%{

 set.seed(12345)

 atv=rep(0,10)

 for (w in 1:10){

 atv[w]=rlnorm(1,log(premium[w])+logelr[k]+alpha[k,w],sigma[k,10])

 }

 at=atv

}

stopCluster(cl)

#

# calculate loss statistics and output to data frame

#

Premium=subset(rdata,rdata$d==1)$premium

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.CSR=rowSums(at.wd10)

ms.td10=mean(Pred.CSR)

ss.td10=sd(Pred.CSR)

CSR.Estimate=round(ms.td10)

CSR.SE=round(ss.td10)

CSR.CV=round(CSR.SE/CSR.Estimate,4)

Outcome=sum(subset(aloss,adata$d==10)[1:10])

pred.CSR=rowSums(at.wd10[,1:10])

CSR.Pct=sum(pred.CSR<=Outcome)/length(pred.CSR)\*100

#

# calculate the best estimate (discounted)

#

pv.paid=matrix(0,10,10)

paid=matrix(0,10,11)

cl <- makeCluster(4)

registerDoParallel(cl)

bestest=foreach (i=1:num.mcmc,.combine=rbind) %dopar%{

 for (w in 1:10){

 for (d in 1:10){

 paid[w,d+1]=

 Premium[w]\*

 exp(logelr[i]+alpha[i,w]+beta[i,d]\*speedup[i,w]+sigma[i,d]^2/2)

 }

 }

 for (w in 2:10){

 for (d in (12-w):10){

 pv.paid[w,d]=(paid[w,d+1]-paid[w,d])/(1+fixed.rate)^(w+d-11.5)

 }

 }

 be=sum(pv.paid)

}

stopCluster(cl)

best.estimate=round(mean(bestest))

# simulate loss for the lower cumulative triangle

# in order of increasing calendar year and

# decreasing accident year within each calendar year

#

loss.lowtri=matrix(0,num.mcmc,45)

mu.lowtri=matrix(0,num.mcmc,45)

wmap=11-1:9

for (i in 1:8){

 wmap=c(wmap,11-1:(9-i))

}

dmap=2:10

for(i in 3:10){

 dmap=c(dmap,i:10)

}

cyfirst=c(1,10,18,25,31,36,40,43,45)

cylast=c(9,17,24,30,35,39,42,44,45)

#

set.seed(12345)

for (i in 1:45){

 mu.lowtri[,i]=log(Premium[wmap[i]])+logelr+alpha[,wmap[i]]+

 beta[,dmap[i]]\*speedup[,wmap[i]]

 loss.lowtri[,i]=rlnorm(num.mcmc,mu.lowtri[,i],sigma[,dmap[i]])

}

#

# unconditional ultimate loss estimates by scenario

#

mean.ult=matrix(0,num.mcmc,10)

for (i in 1:10){

 mean.ult[,i]=exp(log(Premium[i])+logelr+alpha[,i]+sigma[,10]^2/2)

}

ultall=rowSums(mean.ult)

#

# likelihood function of the observed new cy

#

llike=function(x,cy,sz){

 ll=rep(0,sz)

 for (i in (cyfirst[cy]:cylast[cy])){

 ll=ll+dnorm(log(x[i]),mu.lowtri[,i],sigma[,dmap[i]],log=T)

 }

 return(ll)

}

#

# calculate the posterior post\_TVaR -

#

post\_assets=function(post,ult,k){

 set.seed(k)

 x=sample(ult,10000,replace=T,post)

 pa=rep(0,2)

 pa[1]=mean(x)

 pa[2]=mean(sort(x)[TVaR.Range])

 return(pa)

}

#

# get conditional estimates

#

loglike=rep(0,num.mcmc)

p.mean=rep(0,10)

p.assets=rep(0,10)

cl <- makePSOCKcluster(4)

registerDoParallel(cl)

pred.mean.assets=foreach (i=1:num.mcmc,.combine=rbind) %dopar%{

 x=loss.lowtri[i,]

 call.pa=post\_assets(rep(1/num.mcmc,num.mcmc),ultall,i)

 p.mean[1]=call.pa[1]

 p.assets[1]=call.pa[2]

 loglike=rep(0,num.mcmc)

 loglike=loglike+llike(x,1,num.mcmc)

 loglike2=loglike-max(loglike)

 postint=sum(exp(loglike2))

 posterior=exp(loglike2)/postint

 call.pa=post\_assets(posterior,ultall,i)

 p.mean[2]=call.pa[1]

 p.assets[2]=call.pa[2]

 #

 loglike=loglike+llike(x,2,num.mcmc)

 loglike2=loglike-max(loglike)

 postint=sum(exp(loglike2))

 posterior=exp(loglike2)/postint

 call.pa=post\_assets(posterior,ultall,i)

 p.mean[3]=call.pa[1]

 p.assets[3]=call.pa[2]

 #

 loglike=loglike+llike(x,3,num.mcmc)

 loglike2=loglike-max(loglike)

 postint=sum(exp(loglike2))

 posterior=exp(loglike2)/postint

 call.pa=post\_assets(posterior,ultall,i)

 p.mean[4]=call.pa[1]

 p.assets[4]=call.pa[2]

 #

 loglike=loglike+llike(x,4,num.mcmc)

 loglike2=loglike-max(loglike)

 postint=sum(exp(loglike2))

 posterior=exp(loglike2)/postint

 call.pa=post\_assets(posterior,ultall,i)

 p.mean[5]=call.pa[1]

 p.assets[5]=call.pa[2]

 #

 loglike=loglike+llike(x,5,num.mcmc)

 loglike2=loglike-max(loglike)

 postint=sum(exp(loglike2))

 posterior=exp(loglike2)/postint

 call.pa=post\_assets(posterior,ultall,i)

 p.mean[6]=call.pa[1]

 p.assets[6]=call.pa[2]

 #

 loglike=loglike+llike(x,6,num.mcmc)

 loglike2=loglike-max(loglike)

 postint=sum(exp(loglike2))

 posterior=exp(loglike2)/postint

 call.pa=post\_assets(posterior,ultall,i)

 p.mean[7]=call.pa[1]

 p.assets[7]=call.pa[2]

 #

 loglike=loglike+llike(x,7,num.mcmc)

 loglike2=loglike-max(loglike)

 postint=sum(exp(loglike2))

 posterior=exp(loglike2)/postint

 call.pa=post\_assets(posterior,ultall,i)

 p.mean[8]=call.pa[1]

 p.assets[8]=call.pa[2] #

 loglike=loglike+llike(x,8,num.mcmc)

 loglike2=loglike-max(loglike)

 postint=sum(exp(loglike2))

 posterior=exp(loglike2)/postint

 call.pa=post\_assets(posterior,ultall,i)

 p.mean[9]=call.pa[1]

 p.assets[9]=call.pa[2]

 #

 loglike=loglike+llike(x,9,num.mcmc)

 loglike2=loglike-max(loglike)

 postint=sum(exp(loglike2))

 posterior=exp(loglike2)/postint

 call.pa=post\_assets(posterior,ultall,i)

 p.mean[10]=call.pa[1]

 p.assets[10]=call.pa[2]

 result=c(p.mean,p.assets)

}

stopCluster(cl)

#

#

pred.E=as.matrix(pred.mean.assets[,1:10])

pred.A=as.matrix(pred.mean.assets[,11:20])

pred.C=pred.A-pred.E

#

# plot sample ultimate loss estimates for future calendar years

#

cyprobs=seq(0.01,.99,.02)

cyindex=trunc(quantile(1:num.mcmc,probs=cyprobs,names=F))

cynums=order(pred.E[,10])[cyindex]

par(mfrow=c(1,1))

plot(0:9,pred.E[cynums[1],],

 main="Paths of Ultimate Loss Estimates",

 xlab="Future Calendar Year",ylab="Ultimate Loss Estimate",

 ylim=range(pred.E[cynums,]),type="l",

 sub=paste("Best Estimate of Liabilities = ",round(best.estimate)))

for(i in 2:length(cynums)){

 par(new=T)

 plot(0:9,pred.E[cynums[i],],main="",xlab="",ylab="",

 ylim=range(pred.E[cynums,]),type="l",sub="")

}

#

# plot sample capital requirements

#

plot(0:9,pred.C[cynums[1],],

 main="Required Capital by Calendar Year",

 xlab="Future Calendar Year",ylab="Required Capital",

 ylim=range(pred.C[cynums,]),type="l",

 sub=paste("Initial Capital =",round(pred.C[1,1])))

for(i in 2:length(cynums)){

 par(new=T)

 plot(0:9,pred.C[cynums[i],],main="",xlab="",ylab="",

 ylim=range(pred.C[cynums,]),type="l",sub="")

}

#

# plot some capital release paths

#

cyprobs=seq(0.01,.99,.02)

cyindex=trunc(quantile(1:num.mcmc,probs=cyprobs,names=F))

cynums=order(pred.E[,10])[cyindex]

release=pred.C[,1:9]\*(1+fixed.rate)-pred.C[,2:10]

par(mfrow=c(1,1))

plot(1:9,release[cynums[1],],main="Paths of Released Capital",

 xlab="Future Calendar Year",ylab="Capital Released",

 ylim=range(release[cynums,]),type="l")

for(i in 2:length(cynums)){

 par(new=T)

 plot(1:9,release[cynums[i],],main="",xlab="",ylab="",

 ylim=range(release[cynums,]),type="l",sub="")

}

abline(0,0,col="red",lwd=3)

#

# calculate risk margins

#

risk.margin=pred.C[,1]

for (i in 1:9){

 risk.margin=risk.margin-release[,i]/(1+risky.rate)^i

}

par(mfrow=c(1,1))

hist(risk.margin,xlab="Risk Margin",main="Risk Margin",

 sub=paste("Mean Risk Margin =",round(mean(risk.margin))))

#

# plot risk margin as a % of the initial capital

#

risk.margin.pct=100\*risk.margin/pred.C[1,1]

hist(risk.margin.pct,main="Risk Margin as a % of Initial Capital",

 xlab="Risk Margin %",

 sub=paste("Mean =",round(mean(risk.margin.pct),1),"%",

 "Std. Dev. =",round(sd(risk.margin.pct),1),"%"))

t2=Sys.time()

print(t2-t1)