HW8

ASG

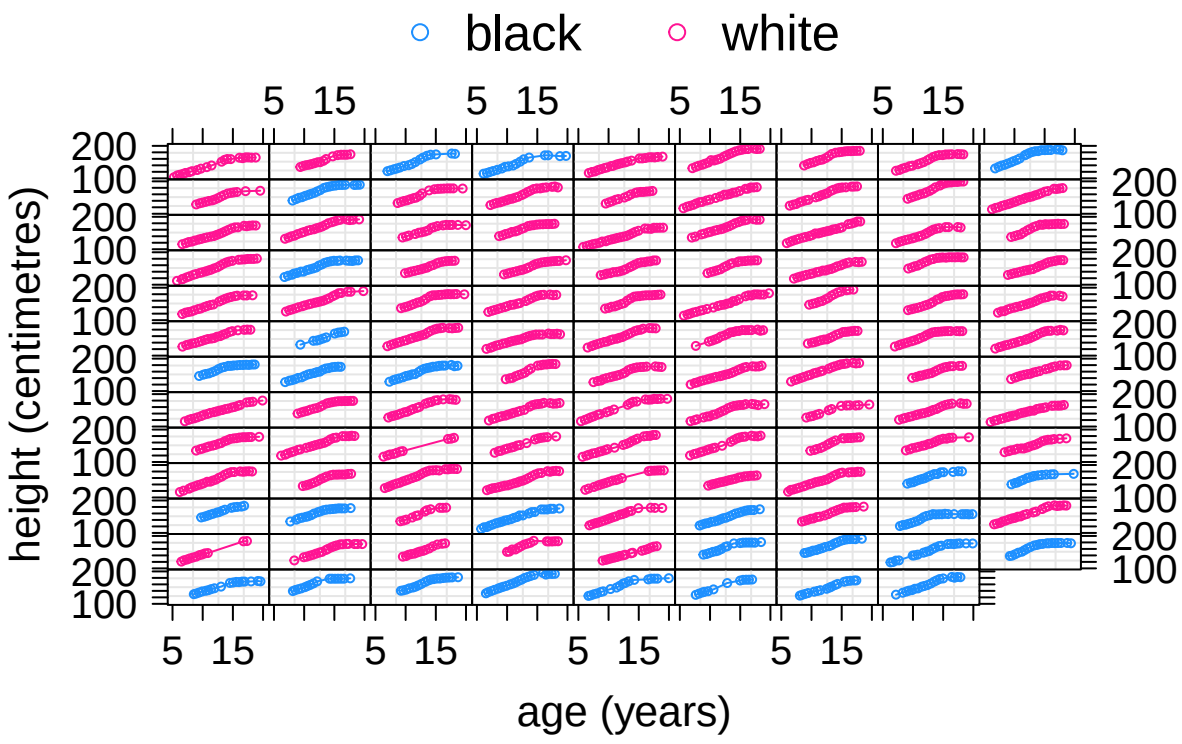
2023-04-19

2

a

```
library(HRW)  ; library(lattice)  ;  library(nlme)

# Load in the data:

data(growthIndiana)

# Extract data on males:

growthIndianaMales <- growthIndiana[growthIndiana$male==1,]

# Create relevant variables:

yOrig <- growthIndianaMales$height
xOrig <- growthIndianaMales$age
idnumOrig <- growthIndianaMales$idnum
typeIsB <- growthIndianaMales$black

# Create new (ordered) ID numbers:

idnum <- rep(NA,length(idnumOrig))
uqID <- unique(idnumOrig)
for (i in 1:length(uqID))
      idnum[idnumOrig==uqID[i]] <- i

# Save mean and standard deviation information:

mean.x <- mean(xOrig)   ;   sd.x <- sd(xOrig)
mean.y <- mean(yOrig)   ;   sd.y <- sd(yOrig)

# Store the standardised versions of the predictor
# and response variables in x and y:

x <- (xOrig - mean.x)/sd.x
y <- (yOrig - mean.y)/sd.y

# Do plot of raw data:
```

```
allDataDF <- data.frame(xOrig,yOrig,idnum,typeIsB)
cex.labVal <- 1.35
colourVec <- c("dodgerblue","deeppink","blue","darkmagenta")

figRaw <- xyplot(yOrig~xOrig|idnum,groups=idnum,data=allDataDF,
                 layout=c(9,13),xlab=list(label="age (years)",cex=cex.labVal),
                 ylab=list(label="height (centimetres)",cex=cex.labVal),
                 scales=list(cex=1.25),strip=FALSE,as.table=TRUE,
                 key = list(title="",
                            columns = 2,
                            points = list(pch = rep(1,2),col = colourVec[1:2]),
                            text = list(c("black","white"),cex=1.55)),
                 panel=function(x,y,subscripts,groups)
                 {
                    panel.grid()
                    colourInd <- 2 - typeIsB[subscripts[1]]
                    panel.superpose(x,y,subscripts,groups,
                             type="b",col=colourVec[colourInd],pch=1,cex=0.5)

                 })
print(figRaw)
```




```
# Set dimension variables:

numObs <- length(x)
```

```
numGrp <- length(unique(idnum))

# Set up X and Z matrix for mean curve:

Xbase <- cbind(rep(1,numObs),x)
XBvA <- typeIsB*Xbase
X <- cbind(Xbase,XBvA)

numIntKnots <- 20
intKnots <- quantile(unique(x),
                     seq(0,1,length=numIntKnots+2))[-c(1,numIntKnots+2)]

range.x <- c(1.01*min(x) - 0.01*max(x),1.01*max(x)-0.01*min(x))

Zbase <- ZOSull(x,intKnots=intKnots,range.x=range.x)
ZBvA <- typeIsB*Zbase

# Set up Z matrix for group specific curves:

numIntKnotsGrp <- 10
intKnotsGrp <- quantile(unique(x),
        seq(0,1,length=numIntKnotsGrp+2))[-c(1,numIntKnotsGrp+2)]

Zgrp <- ZOSull(x,intKnots=intKnotsGrp,range.x=range.x)

# Let up dimension variables:

ncZ <- ncol(Zbase) ; ncZgrp <- ncol(Zgrp)

# Fit using linear mixed model software:

dummyId <- factor(rep(1,numObs))
Zblock <- list(dummyId=pdIdent(~-1+Zbase),
               dummyId=pdIdent(~-1+ZBvA),
               idnum=pdSymm(~x),
               idnum=pdIdent(~-1+Zgrp))
growthINmalGD <- groupedData(y~X[,-1]|rep(1,length=numObs),
                         data=data.frame(y,X,idnum))
cat("Starting fitting (takes several minutes)...\n")
```

```
## Starting fitting (takes several minutes)...
```

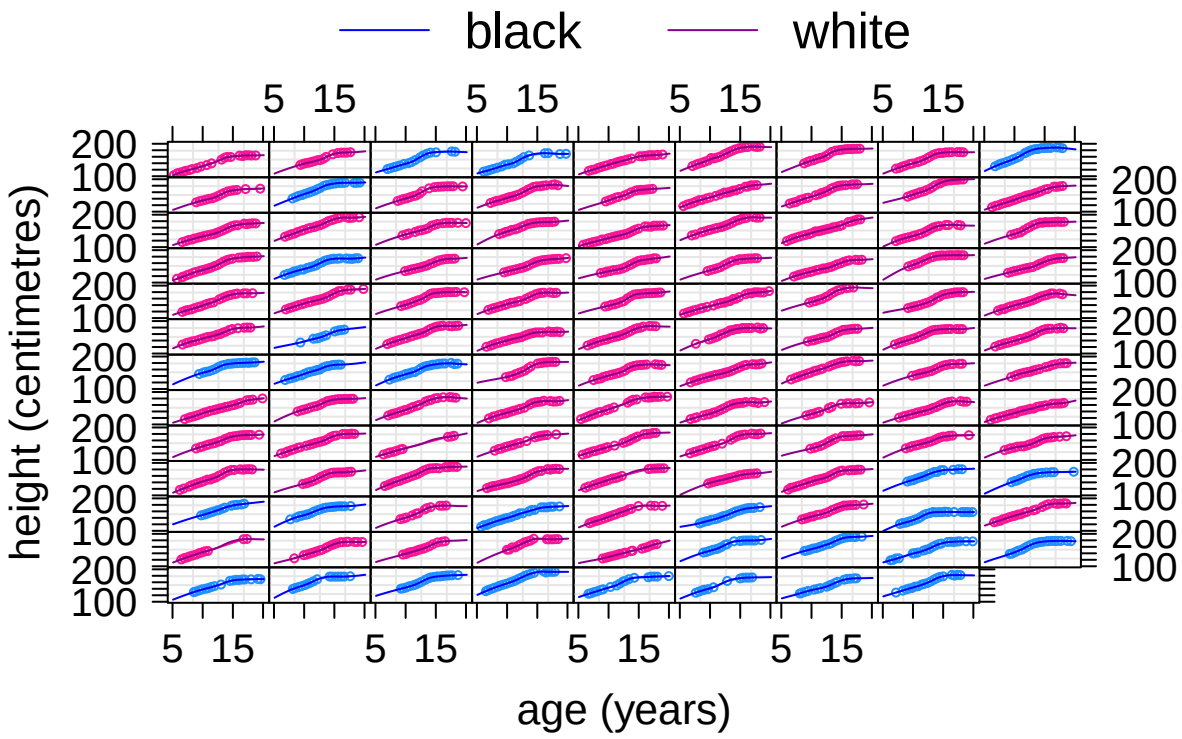
```
fit <- lme(y~-1+X,data=growthINmalGD,random=Zblock)
cat("Finished fitting.\n")
```

```
## Finished fitting.
```

```
sig.epsHat <- fit$sigma
sig.uHat <- sig.epsHat*exp(unlist(fit$modelStruct))

lamVal <- (sig.epsHat/sig.uHat)^2
```

```
# Set up plotting grids:

ng <- 101
xg <- seq(range.x[1],range.x[2],length=ng)
Xg <- cbind(rep(1,ng),xg)
Zg <- ZOSull(xg,intKnots=intKnots,range.x=range.x)
betaHat <- as.vector(fit$coef$fixed)
uHatBase <- as.vector(fit$coef$random[[1]])
uHatCont <- as.vector(fit$coef$random[[2]])

fhatBaseg <- Xg%*%betaHat[1:2] + Zg%*%uHatBase
Contg <-  Xg%*%betaHat[3:4] + Zg%*%uHatCont
fhatBlackg <- fhatBaseg + Contg

# Plot fitted curves:

Xsubjg <- cbind(rep(1,ng),xg)
Zgrpg <- ZOSull(xg,intKnots=intKnotsGrp,range.x=range.x)
meanCrvs <- vector("list",numGrp)
for (i in 1:numGrp)
{
   indsi <- (1:numObs)[idnum==i]
   uLinHati <- as.vector(fit$coef$random[[3]][i,])
   uSplHati <- as.vector(fit$coef$random[[4]][i,])
   ghati <- Xsubjg%*%uLinHati + Zgrpg%*%uSplHati
   meanCrvs[[i]] <- fhatBaseg + typeIsB[indsi[1]]*Contg + ghati
}

# Convert to original units:

xgOrig <- xgOrig <- mean.x + sd.x*xg
meanCrvsOrig <- vector("list",numGrp)
for (i in 1:numGrp)
   meanCrvsOrig[[i]] <- mean.y + sd.y*meanCrvs[[i]]

# Do lattice type plot showing group-specific fits:

figFit <- xyplot(yOrig~xOrig|idnum,groups=idnum,data=allDataDF,
                 layout=c(9,13),xlab=list(label="age (years)",cex=cex.labVal),
                 ylab=list(label="height (centimetres)",cex=cex.labVal),
                 scales=list(cex=1.25),strip=FALSE,as.table=TRUE,
                 key = list(title="",
                            columns = 2,
                            lines = list(lty = rep(1,2),col = colourVec[3:4]),
                            text = list(c("black","white"),cex=1.55)),
                 panel=function(x,y,subscripts,groups)
                 {
                    panel.grid()
                    iGrp <- idnum[subscripts][1]
                    colourInd <- 2 - typeIsB[subscripts[1]]
                    panel.superpose(x,y,subscripts,groups,
                             type="b",col=colourVec[colourInd],pch=1,cex=0.5)
                    panel.xyplot(xgOrig,meanCrvsOrig[[iGrp]],
```

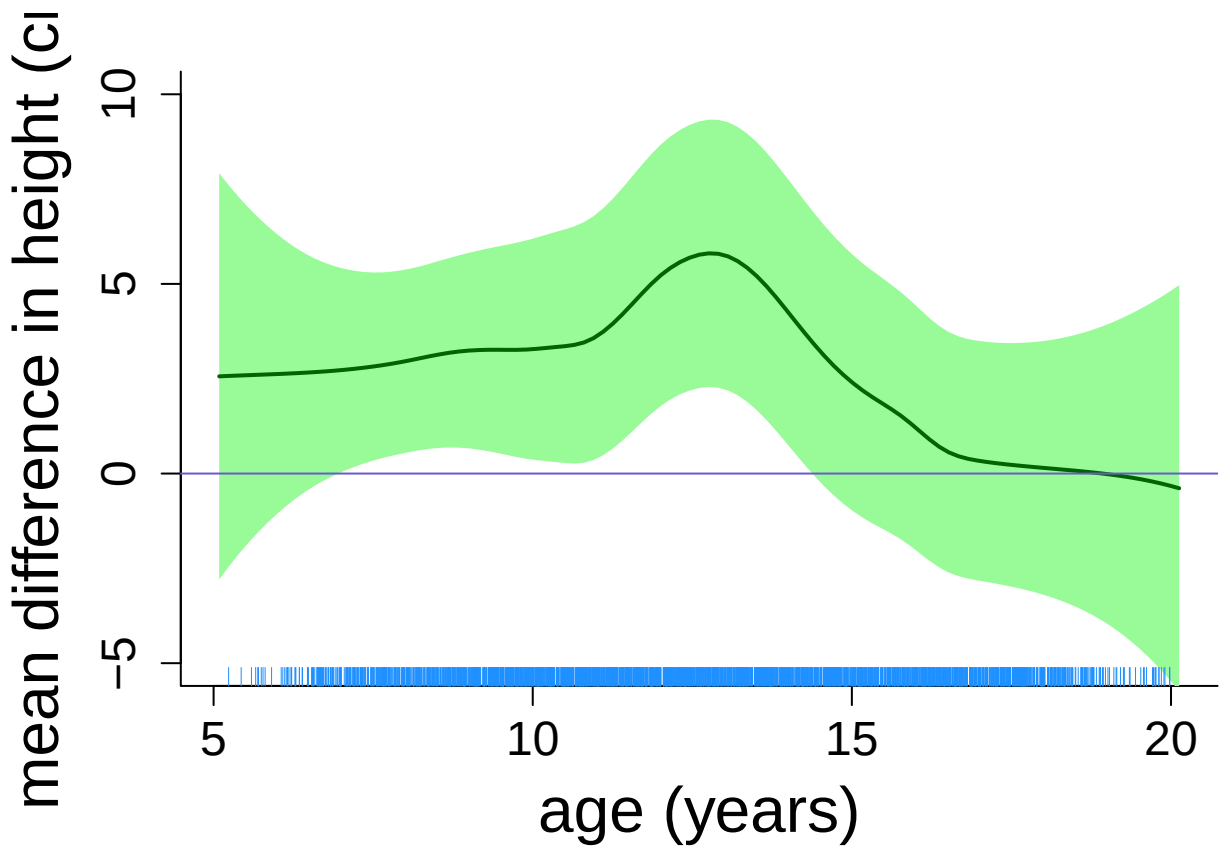
```
                                type="l",col=colourVec[colourInd+2])
                })
print(figFit)
```




```
# Obtain penalty matrix:

sigsq.epsHat <- fit$sigma^2
sig.uHat <- as.numeric(sqrt(sigsq.epsHat)*exp(unlist(fit$modelStruct)))

sigsq.Gbl <- sig.uHat[6]^2
sigsq.Gblcont <- sig.uHat[5]^2
cholSigma <- rbind(c(sig.uHat[2],sig.uHat[3]),c(0,sig.uHat[4]))
SigmaHat <- crossprod(cholSigma)
sigsq.Sbj <- sig.uHat[1]^2

DmatGbl <- (sigsq.epsHat/sigsq.Gbl)*diag(ncZ)
DmatGblcont <- (sigsq.epsHat/sigsq.Gblcont)*diag(ncZ)
DmatLinSbj <- sigsq.epsHat*kronecker(diag(numGrp),solve(SigmaHat))
DmatSplSbj <- (sigsq.epsHat/sigsq.Sbj)*diag(ncol(Zgrp)*numGrp)

dimVec <- c(4,nrow(DmatGbl),nrow(DmatGblcont),nrow(DmatLinSbj),
            nrow(DmatSplSbj))

lamMat <- matrix(0,sum(dimVec),sum(dimVec))
csdV <- cumsum(dimVec)
```

```
lamMat[(csdV[1]+1):csdV[2],(csdV[1]+1):csdV[2]] <- DmatGbl
lamMat[(csdV[2]+1):csdV[3],
       (csdV[2]+1):csdV[3]] <- DmatGblcont
lamMat[(csdV[3]+1):csdV[4],
       (csdV[3]+1):csdV[4]] <- DmatLinSbj
lamMat[(csdV[4]+1):csdV[5],
       (csdV[4]+1):csdV[5]] <- DmatSplSbj

# Obtain C matrix:

uqID <- unique(idnum)
Cmat <- cbind(X,Zbase,ZBvA)
for (iSbj in 1:numGrp)
{
   newCols <- matrix(0,numObs,2)
   indsCurr <- (1:numObs)[idnum==uqID[iSbj]]
   newCols[indsCurr,] <- X[indsCurr,1:2]
   Cmat <- cbind(Cmat,newCols)
}
for (iSbj in 1:numGrp)
{
   newCols <- matrix(0,numObs,ncol(Zgrp))
   indsCurr <- (1:numObs)[idnum==uqID[iSbj]]
   newCols[indsCurr,] <- Zgrp[indsCurr,]
   Cmat <- cbind(Cmat,newCols)
}

# Obtain full covariance matrix:

CTC <- crossprod(Cmat)
fullCovMat <- solve(CTC + lamMat)

# Find subset of covariance corresponding to the contrast curve:

contInds <- c(3,4,(ncZ+5):(2*ncZ+4))
contCovMat <- fullCovMat[contInds,contInds]

# Obtain approximate pointwise 95% confidence limits:

Cg <- cbind(Xg,Zg)
sdg <- sqrt(sigsq.epsHat)*sqrt(diag(Cg%*%contCovMat%*%t(Cg)))
lowerg <- Contg - qnorm(0.975)*sdg ; upperg <- Contg + qnorm(0.975)*sdg

ContgOrig <- sd.y*Contg
lowergOrig <- sd.y*lowerg
uppergOrig <- sd.y*upperg

# Plot contrast with variability band:

par(mai=c(1,0.9,0.3,0.2))
cex.labVal <- 2   ; cex.axisVal <- 1.5
plot(xgOrig,ContgOrig,bty="l",
     type="n",xlab="age (years)",ylab="mean difference in height (cm)",
```

```
     ylim=c(-5,10),cex.lab=cex.labVal,cex.axis=cex.axisVal)

polygon(c(xgOrig,rev(xgOrig)),c(lowergOrig,rev(uppergOrig)),col="PaleGreen",
     border=FALSE)

lines(xgOrig,ContgOrig,col="DarkGreen",lwd=2)
abline(0,0,col="slateblue")
rug(xOrig,col="dodgerblue")
```



the two grid graphs shows the raw grouped data and the other one shows the fitted curve for the same data.

b

```
compileCode <- TRUE

# Set MCMC sample size paramaters:

nWarm <- 1000
nKept <- 1000
nThin <- 1

# Load required packages:

library(HRW)   ;   library(rstan)   ;  library(lattice)
```

```
## Loading required package: StanHeaders

## Loading required package: ggplot2

## rstan (Version 2.21.8, GitRev: 2e1f913d3ca3)

## For execution on a local, multicore CPU with excess RAM we recommend calling
## options(mc.cores = parallel::detectCores()).
## To avoid recompilation of unchanged Stan programs, we recommend calling
## rstan_options(auto_write = TRUE)
```

```
# Load in the data:

data(growthIndiana)

# Extract data on males:

growthIndianaMales <- growthIndiana[growthIndiana$male == 1,]

# Create relevant variables:

height <- growthIndianaMales$height
age <- growthIndianaMales$age
idnumOrig <- growthIndianaMales$idnum
typeIsB <- growthIndianaMales$black

# Create new (ordered) ID numbers:

idnum <- rep(NA,length(idnumOrig))
uqID <- unique(idnumOrig)
for (i in 1:length(uqID))
      idnum[idnumOrig == uqID[i]] <- i

# Use xOrig and yOrig to denote the original continuous
# predictor and response variables:

xOrig <- age ;  yOrig <- height

# Save mean and standard deviation information:

mean.x <- mean(xOrig)   ;   sd.x <- sd(xOrig)
mean.y <- mean(yOrig)   ;   sd.y <- sd(yOrig)

# Store the standardised versions of the predictor
# and response variables in x and y:

x <- (xOrig - mean.x)/sd.x
y <- (yOrig - mean.y)/sd.y

# Set up X and Z matrix for mean curve:

numObs <- length(x)
numGrp <- length(unique(idnum))
```

```
Xbase <- cbind(rep(1,numObs),x)
XB <- typeIsB*Xbase

numIntKnots <- 20
intKnots <- quantile(unique(x),
                     seq(0,1,length=numIntKnots+2))[-c(1,numIntKnots+2)]

range.x <- c(1.01*min(x) - 0.01*max(x),1.01*max(x)-0.01*min(x))

Z <- ZOSull(x,intKnots=intKnots,range.x=range.x)
ZA <- (1-typeIsB)*Z
ZB <- typeIsB*Z

# Set up Z matrix for group specific curves:

numIntKnotsGrp <- 10
intKnotsGrp <- quantile(unique(x),
        seq(0,1,length=numIntKnotsGrp+2))[-c(1,numIntKnotsGrp+2)]

Zgrp <- ZOSull(x,intKnots=intKnotsGrp,range.x=range.x)

# Specify hyperparameters:

sigmaBeta <- 1e5  ;  AU <- 1e5
AuGbl <- 1e5      ;  AuGrp <- 1e5
Aeps <- 1e5

# Let up dimension variables:

ncZ <- ncol(Z) ; ncZgrp <- ncol(Zgrp)

# Set up model in Stan:

normGrpSpecContrModel <-
'data
{
   int<lower=1> numObs;              int<lower=1> numGrp;
   int<lower=1> ncZ;                 int<lower=1> ncZgrp;
   real<lower=0> sigmaBeta;          real<lower=0> AU;
   real<lower=0> AuGbl;              real<lower=0> AuGrp;
   real<lower=0> Aeps;               matrix[numObs,2] Xbase;
   matrix[numObs,2] XB;              real y[numObs];
   int<lower=1> idnum[numObs];       matrix[numObs,ncZ] ZA;
   matrix[numObs,ncZ] ZB;            matrix[numObs,ncZgrp] Zgrp;
}
transformed data
{
   vector[2] zeroVec;           zeroVec = rep_vector(0,2);
}
parameters
{
   vector[2] beta;                vector[2] betaBvsA;
   vector[2] U[numGrp];           vector[ncZ] uGblA;
```

```
   vector[ncZ] uGblB;            matrix[numGrp,ncZgrp] uGrp;
   cov_matrix[2] SigmaGrp;       real<lower=0> siguGblA;
   real<lower=0> siguGblB;       real<lower=0> siguGrp;
   real<lower=0> sigEps;         vector[2] a;
}
transformed parameters
{
   vector[numObs] meanVec;
   meanVec = Xbase*beta + XB*betaBvsA + ZA*uGblA + ZB*uGblB
             + to_vector(U[idnum,1]) + to_vector(U[idnum,2]).*col(Xbase,2)
             + rows_dot_product(uGrp[idnum],Zgrp);
}
model
{
   vector[2] scaleSigmaGrp;

   y ~ normal(meanVec,sigEps);

   U ~ multi_normal(zeroVec,SigmaGrp);
   uGblA ~ normal(0,siguGblA);  uGblB ~ normal(0,siguGblB);
   to_vector(uGrp) ~ normal(0,siguGrp);

   a ~ inv_gamma(0.5,pow(AU,-2));
   for (k in 1:2) scaleSigmaGrp[k] = 4/a[k];
   SigmaGrp ~ inv_wishart(3,diag_matrix(scaleSigmaGrp));

   beta ~ normal(0,sigmaBeta);   betaBvsA ~ normal(0,sigmaBeta);
   siguGblA ~ cauchy(0,AuGbl);   siguGblB ~ cauchy(0,AuGbl);
   siguGrp ~ cauchy(0,AuGrp);    sigEps ~ cauchy(0,Aeps);
}'


# Set up data list:

allData <- list(numObs=numObs,numGrp=numGrp,ncZ=ncZ,
                ncZgrp=ncZgrp,AU=AU,AuGbl=AuGbl,
                AuGrp=AuGrp,Aeps=Aeps,sigmaBeta=sigmaBeta,
                y=y,idnum=idnum,ZA=ZA,ZB=ZB,
                Zgrp=Zgrp,Xbase=Xbase,XB=XB)

# Compile code for model if required:

if (compileCode)
   stanCompilObj <- stan(model_code=normGrpSpecContrModel,data=allData,
                         iter=1,chains=1)
```

```
## Trying to compile a simple C file

## Running /Library/Frameworks/R.framework/Resources/bin/R CMD SHLIB foo.c
## clang -mmacosx-version-min=10.13 -I"/Library/Frameworks/R.framework/Resources/include" -DNDEBUG   -I
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/4.2/Resources/library/StanHeaders/incl
## In file included from /Library/Frameworks/R.framework/Versions/4.2/Resources/library/RcppEigen/inclu
```

```
## In file included from /Library/Frameworks/R.framework/Versions/4.2/Resources/library/RcppEigen/inclu
## /Library/Frameworks/R.framework/Versions/4.2/Resources/library/RcppEigen/include/Eigen/src/Core/util
## namespace Eigen {
## ^
## /Library/Frameworks/R.framework/Versions/4.2/Resources/library/RcppEigen/include/Eigen/src/Core/util
## namespace Eigen {
##                ^
##                ;
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/4.2/Resources/library/StanHeaders/inc
## In file included from /Library/Frameworks/R.framework/Versions/4.2/Resources/library/RcppEigen/inclu
## /Library/Frameworks/R.framework/Versions/4.2/Resources/library/RcppEigen/include/Eigen/Core:96:10: f
## #include <complex>
##          ^~~~~~~~~
## 3 errors generated.
## make: *** [foo.o] Error 1
##
## SAMPLING FOR MODEL '180a82ef220311c3d0e03a748d5a12dd' NOW (CHAIN 1).
## Chain 1: Rejecting initial value:
## Chain 1:   Error evaluating the log probability at the initial value.
## Chain 1: Exception: inv_wishart_lpdf: LDLT_Factor of scale parameter is not positive definite.  last
##
## Chain 1: Rejecting initial value:
## Chain 1:   Error evaluating the log probability at the initial value.
## Chain 1: Exception: inv_wishart_lpdf: LDLT_Factor of scale parameter is not positive definite.  last
##
## Chain 1: Rejecting initial value:
## Chain 1:   Error evaluating the log probability at the initial value.
## Chain 1: Exception: inv_wishart_lpdf: LDLT_Factor of scale parameter is not positive definite.  last
##
## Chain 1: Rejecting initial value:
## Chain 1:   Error evaluating the log probability at the initial value.
## Chain 1: Exception: inv_wishart_lpdf: LDLT_Factor of scale parameter is not positive definite.  last
##
## Chain 1: Rejecting initial value:
## Chain 1:   Error evaluating the log probability at the initial value.
## Chain 1: Exception: inv_wishart_lpdf: LDLT_Factor of scale parameter is not positive definite.  last
##
## Chain 1: Rejecting initial value:
## Chain 1:   Error evaluating the log probability at the initial value.
## Chain 1: Exception: inv_wishart_lpdf: LDLT_Factor of scale parameter is not positive definite.  last
##
## Chain 1:
## Chain 1: Gradient evaluation took 0.002929 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 29.29 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: WARNING: No variance estimation is
## Chain 1:          performed for num_warmup < 20
## Chain 1:
## Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 3e-06 seconds (Warm-up)
```

```
## Chain 1:                0.004972 seconds (Sampling)
## Chain 1:                0.004975 seconds (Total)
## Chain 1:
```

```
## Warning: There were 1 divergent transitions after warmup. See
## https://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
## to find out why this is a problem and how to eliminate them.
```

```
## Warning: Examine the pairs() plot to diagnose sampling problems
```

```
# Perform MCMC:

stanObj <-  stan(model_code=normGrpSpecContrModel,data=allData,warmup=nWarm,
                 iter=(nWarm + nKept),chains=1,thin=nThin,refresh=10,
                 fit=stanCompilObj)
```

```
##
## SAMPLING FOR MODEL '180a82ef220311c3d0e03a748d5a12dd' NOW (CHAIN 1).
## Chain 1: Rejecting initial value:
## Chain 1:   Error evaluating the log probability at the initial value.
## Chain 1: Exception: inv_wishart_lpdf: LDLT_Factor of scale parameter is not positive definite.  last
##
## Chain 1: Rejecting initial value:
## Chain 1:   Error evaluating the log probability at the initial value.
## Chain 1: Exception: inv_wishart_lpdf: LDLT_Factor of scale parameter is not positive definite.  last
##
## Chain 1: Rejecting initial value:
## Chain 1:   Error evaluating the log probability at the initial value.
## Chain 1: Exception: inv_wishart_lpdf: LDLT_Factor of scale parameter is not positive definite.  last
##
## Chain 1: Rejecting initial value:
## Chain 1:   Error evaluating the log probability at the initial value.
## Chain 1: Exception: inv_wishart_lpdf: LDLT_Factor of scale parameter is not positive definite.  last
##
## Chain 1:
## Chain 1: Gradient evaluation took 0.001905 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 19.05 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:   10 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:   20 / 2000 [  1%]  (Warmup)
## Chain 1: Iteration:   30 / 2000 [  1%]  (Warmup)
## Chain 1: Iteration:   40 / 2000 [  2%]  (Warmup)
## Chain 1: Iteration:   50 / 2000 [  2%]  (Warmup)
## Chain 1: Iteration:   60 / 2000 [  3%]  (Warmup)
## Chain 1: Iteration:   70 / 2000 [  3%]  (Warmup)
## Chain 1: Iteration:   80 / 2000 [  4%]  (Warmup)
## Chain 1: Iteration:   90 / 2000 [  4%]  (Warmup)
## Chain 1: Iteration:  100 / 2000 [  5%]  (Warmup)
## Chain 1: Iteration:  110 / 2000 [  5%]  (Warmup)
## Chain 1: Iteration:  120 / 2000 [  6%]  (Warmup)
```

```
## Chain 1: Iteration:  130 / 2000 [  6%]  (Warmup)
## Chain 1: Iteration:  140 / 2000 [  7%]  (Warmup)
## Chain 1: Iteration:  150 / 2000 [  7%]  (Warmup)
## Chain 1: Iteration:  160 / 2000 [  8%]  (Warmup)
## Chain 1: Iteration:  170 / 2000 [  8%]  (Warmup)
## Chain 1: Iteration:  180 / 2000 [  9%]  (Warmup)
## Chain 1: Iteration:  190 / 2000 [  9%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  210 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  220 / 2000 [ 11%]  (Warmup)
## Chain 1: Iteration:  230 / 2000 [ 11%]  (Warmup)
## Chain 1: Iteration:  240 / 2000 [ 12%]  (Warmup)
## Chain 1: Iteration:  250 / 2000 [ 12%]  (Warmup)
## Chain 1: Iteration:  260 / 2000 [ 13%]  (Warmup)
## Chain 1: Iteration:  270 / 2000 [ 13%]  (Warmup)
## Chain 1: Iteration:  280 / 2000 [ 14%]  (Warmup)
## Chain 1: Iteration:  290 / 2000 [ 14%]  (Warmup)
## Chain 1: Iteration:  300 / 2000 [ 15%]  (Warmup)
## Chain 1: Iteration:  310 / 2000 [ 15%]  (Warmup)
## Chain 1: Iteration:  320 / 2000 [ 16%]  (Warmup)
## Chain 1: Iteration:  330 / 2000 [ 16%]  (Warmup)
## Chain 1: Iteration:  340 / 2000 [ 17%]  (Warmup)
## Chain 1: Iteration:  350 / 2000 [ 17%]  (Warmup)
## Chain 1: Iteration:  360 / 2000 [ 18%]  (Warmup)
## Chain 1: Iteration:  370 / 2000 [ 18%]  (Warmup)
## Chain 1: Iteration:  380 / 2000 [ 19%]  (Warmup)
## Chain 1: Iteration:  390 / 2000 [ 19%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  410 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  420 / 2000 [ 21%]  (Warmup)
## Chain 1: Iteration:  430 / 2000 [ 21%]  (Warmup)
## Chain 1: Iteration:  440 / 2000 [ 22%]  (Warmup)
## Chain 1: Iteration:  450 / 2000 [ 22%]  (Warmup)
## Chain 1: Iteration:  460 / 2000 [ 23%]  (Warmup)
## Chain 1: Iteration:  470 / 2000 [ 23%]  (Warmup)
## Chain 1: Iteration:  480 / 2000 [ 24%]  (Warmup)
## Chain 1: Iteration:  490 / 2000 [ 24%]  (Warmup)
## Chain 1: Iteration:  500 / 2000 [ 25%]  (Warmup)
## Chain 1: Iteration:  510 / 2000 [ 25%]  (Warmup)
## Chain 1: Iteration:  520 / 2000 [ 26%]  (Warmup)
## Chain 1: Iteration:  530 / 2000 [ 26%]  (Warmup)
## Chain 1: Iteration:  540 / 2000 [ 27%]  (Warmup)
## Chain 1: Iteration:  550 / 2000 [ 27%]  (Warmup)
## Chain 1: Iteration:  560 / 2000 [ 28%]  (Warmup)
## Chain 1: Iteration:  570 / 2000 [ 28%]  (Warmup)
## Chain 1: Iteration:  580 / 2000 [ 29%]  (Warmup)
## Chain 1: Iteration:  590 / 2000 [ 29%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  610 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  620 / 2000 [ 31%]  (Warmup)
## Chain 1: Iteration:  630 / 2000 [ 31%]  (Warmup)
## Chain 1: Iteration:  640 / 2000 [ 32%]  (Warmup)
## Chain 1: Iteration:  650 / 2000 [ 32%]  (Warmup)
## Chain 1: Iteration:  660 / 2000 [ 33%]  (Warmup)
```

```
## Chain 1: Iteration:  670 / 2000 [ 33%]  (Warmup)
## Chain 1: Iteration:  680 / 2000 [ 34%]  (Warmup)
## Chain 1: Iteration:  690 / 2000 [ 34%]  (Warmup)
## Chain 1: Iteration:  700 / 2000 [ 35%]  (Warmup)
## Chain 1: Iteration:  710 / 2000 [ 35%]  (Warmup)
## Chain 1: Iteration:  720 / 2000 [ 36%]  (Warmup)
## Chain 1: Iteration:  730 / 2000 [ 36%]  (Warmup)
## Chain 1: Iteration:  740 / 2000 [ 37%]  (Warmup)
## Chain 1: Iteration:  750 / 2000 [ 37%]  (Warmup)
## Chain 1: Iteration:  760 / 2000 [ 38%]  (Warmup)
## Chain 1: Iteration:  770 / 2000 [ 38%]  (Warmup)
## Chain 1: Iteration:  780 / 2000 [ 39%]  (Warmup)
## Chain 1: Iteration:  790 / 2000 [ 39%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration:  810 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration:  820 / 2000 [ 41%]  (Warmup)
## Chain 1: Iteration:  830 / 2000 [ 41%]  (Warmup)
## Chain 1: Iteration:  840 / 2000 [ 42%]  (Warmup)
## Chain 1: Iteration:  850 / 2000 [ 42%]  (Warmup)
## Chain 1: Iteration:  860 / 2000 [ 43%]  (Warmup)
## Chain 1: Iteration:  870 / 2000 [ 43%]  (Warmup)
## Chain 1: Iteration:  880 / 2000 [ 44%]  (Warmup)
## Chain 1: Iteration:  890 / 2000 [ 44%]  (Warmup)
## Chain 1: Iteration:  900 / 2000 [ 45%]  (Warmup)
## Chain 1: Iteration:  910 / 2000 [ 45%]  (Warmup)
## Chain 1: Iteration:  920 / 2000 [ 46%]  (Warmup)
## Chain 1: Iteration:  930 / 2000 [ 46%]  (Warmup)
## Chain 1: Iteration:  940 / 2000 [ 47%]  (Warmup)
## Chain 1: Iteration:  950 / 2000 [ 47%]  (Warmup)
## Chain 1: Iteration:  960 / 2000 [ 48%]  (Warmup)
## Chain 1: Iteration:  970 / 2000 [ 48%]  (Warmup)
## Chain 1: Iteration:  980 / 2000 [ 49%]  (Warmup)
## Chain 1: Iteration:  990 / 2000 [ 49%]  (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1010 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1020 / 2000 [ 51%]  (Sampling)
## Chain 1: Iteration: 1030 / 2000 [ 51%]  (Sampling)
## Chain 1: Iteration: 1040 / 2000 [ 52%]  (Sampling)
## Chain 1: Iteration: 1050 / 2000 [ 52%]  (Sampling)
## Chain 1: Iteration: 1060 / 2000 [ 53%]  (Sampling)
## Chain 1: Iteration: 1070 / 2000 [ 53%]  (Sampling)
## Chain 1: Iteration: 1080 / 2000 [ 54%]  (Sampling)
## Chain 1: Iteration: 1090 / 2000 [ 54%]  (Sampling)
## Chain 1: Iteration: 1100 / 2000 [ 55%]  (Sampling)
## Chain 1: Iteration: 1110 / 2000 [ 55%]  (Sampling)
## Chain 1: Iteration: 1120 / 2000 [ 56%]  (Sampling)
## Chain 1: Iteration: 1130 / 2000 [ 56%]  (Sampling)
## Chain 1: Iteration: 1140 / 2000 [ 57%]  (Sampling)
## Chain 1: Iteration: 1150 / 2000 [ 57%]  (Sampling)
## Chain 1: Iteration: 1160 / 2000 [ 58%]  (Sampling)
## Chain 1: Iteration: 1170 / 2000 [ 58%]  (Sampling)
## Chain 1: Iteration: 1180 / 2000 [ 59%]  (Sampling)
## Chain 1: Iteration: 1190 / 2000 [ 59%]  (Sampling)
```

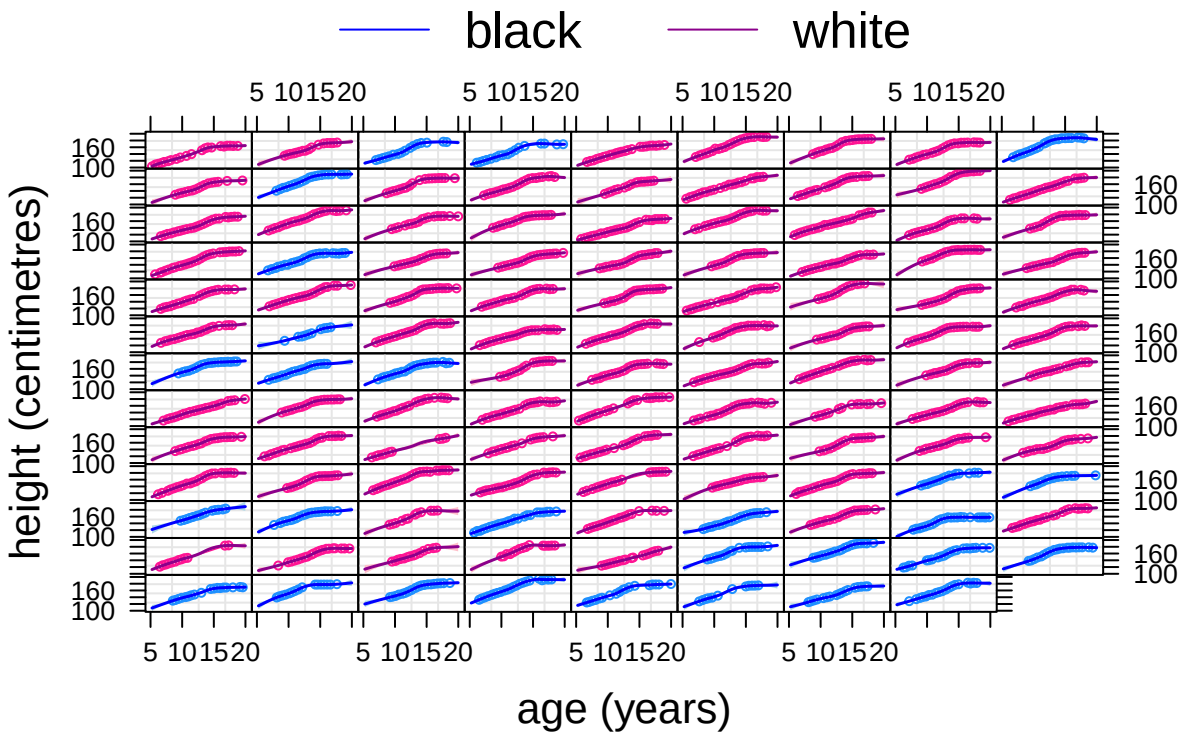
```
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1210 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1220 / 2000 [ 61%]  (Sampling)
## Chain 1: Iteration: 1230 / 2000 [ 61%]  (Sampling)
## Chain 1: Iteration: 1240 / 2000 [ 62%]  (Sampling)
## Chain 1: Iteration: 1250 / 2000 [ 62%]  (Sampling)
## Chain 1: Iteration: 1260 / 2000 [ 63%]  (Sampling)
## Chain 1: Iteration: 1270 / 2000 [ 63%]  (Sampling)
## Chain 1: Iteration: 1280 / 2000 [ 64%]  (Sampling)
## Chain 1: Iteration: 1290 / 2000 [ 64%]  (Sampling)
## Chain 1: Iteration: 1300 / 2000 [ 65%]  (Sampling)
## Chain 1: Iteration: 1310 / 2000 [ 65%]  (Sampling)
## Chain 1: Iteration: 1320 / 2000 [ 66%]  (Sampling)
## Chain 1: Iteration: 1330 / 2000 [ 66%]  (Sampling)
## Chain 1: Iteration: 1340 / 2000 [ 67%]  (Sampling)
## Chain 1: Iteration: 1350 / 2000 [ 67%]  (Sampling)
## Chain 1: Iteration: 1360 / 2000 [ 68%]  (Sampling)
## Chain 1: Iteration: 1370 / 2000 [ 68%]  (Sampling)
## Chain 1: Iteration: 1380 / 2000 [ 69%]  (Sampling)
## Chain 1: Iteration: 1390 / 2000 [ 69%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1410 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1420 / 2000 [ 71%]  (Sampling)
## Chain 1: Iteration: 1430 / 2000 [ 71%]  (Sampling)
## Chain 1: Iteration: 1440 / 2000 [ 72%]  (Sampling)
## Chain 1: Iteration: 1450 / 2000 [ 72%]  (Sampling)
## Chain 1: Iteration: 1460 / 2000 [ 73%]  (Sampling)
## Chain 1: Iteration: 1470 / 2000 [ 73%]  (Sampling)
## Chain 1: Iteration: 1480 / 2000 [ 74%]  (Sampling)
## Chain 1: Iteration: 1490 / 2000 [ 74%]  (Sampling)
## Chain 1: Iteration: 1500 / 2000 [ 75%]  (Sampling)
## Chain 1: Iteration: 1510 / 2000 [ 75%]  (Sampling)
## Chain 1: Iteration: 1520 / 2000 [ 76%]  (Sampling)
## Chain 1: Iteration: 1530 / 2000 [ 76%]  (Sampling)
## Chain 1: Iteration: 1540 / 2000 [ 77%]  (Sampling)
## Chain 1: Iteration: 1550 / 2000 [ 77%]  (Sampling)
## Chain 1: Iteration: 1560 / 2000 [ 78%]  (Sampling)
## Chain 1: Iteration: 1570 / 2000 [ 78%]  (Sampling)
## Chain 1: Iteration: 1580 / 2000 [ 79%]  (Sampling)
## Chain 1: Iteration: 1590 / 2000 [ 79%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1610 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1620 / 2000 [ 81%]  (Sampling)
## Chain 1: Iteration: 1630 / 2000 [ 81%]  (Sampling)
## Chain 1: Iteration: 1640 / 2000 [ 82%]  (Sampling)
## Chain 1: Iteration: 1650 / 2000 [ 82%]  (Sampling)
## Chain 1: Iteration: 1660 / 2000 [ 83%]  (Sampling)
## Chain 1: Iteration: 1670 / 2000 [ 83%]  (Sampling)
## Chain 1: Iteration: 1680 / 2000 [ 84%]  (Sampling)
## Chain 1: Iteration: 1690 / 2000 [ 84%]  (Sampling)
## Chain 1: Iteration: 1700 / 2000 [ 85%]  (Sampling)
## Chain 1: Iteration: 1710 / 2000 [ 85%]  (Sampling)
## Chain 1: Iteration: 1720 / 2000 [ 86%]  (Sampling)
## Chain 1: Iteration: 1730 / 2000 [ 86%]  (Sampling)
```

```
## Chain 1: Iteration: 1740 / 2000 [ 87%]  (Sampling)
## Chain 1: Iteration: 1750 / 2000 [ 87%]  (Sampling)
## Chain 1: Iteration: 1760 / 2000 [ 88%]  (Sampling)
## Chain 1: Iteration: 1770 / 2000 [ 88%]  (Sampling)
## Chain 1: Iteration: 1780 / 2000 [ 89%]  (Sampling)
## Chain 1: Iteration: 1790 / 2000 [ 89%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 1810 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 1820 / 2000 [ 91%]  (Sampling)
## Chain 1: Iteration: 1830 / 2000 [ 91%]  (Sampling)
## Chain 1: Iteration: 1840 / 2000 [ 92%]  (Sampling)
## Chain 1: Iteration: 1850 / 2000 [ 92%]  (Sampling)
## Chain 1: Iteration: 1860 / 2000 [ 93%]  (Sampling)
## Chain 1: Iteration: 1870 / 2000 [ 93%]  (Sampling)
## Chain 1: Iteration: 1880 / 2000 [ 94%]  (Sampling)
## Chain 1: Iteration: 1890 / 2000 [ 94%]  (Sampling)
## Chain 1: Iteration: 1900 / 2000 [ 95%]  (Sampling)
## Chain 1: Iteration: 1910 / 2000 [ 95%]  (Sampling)
## Chain 1: Iteration: 1920 / 2000 [ 96%]  (Sampling)
## Chain 1: Iteration: 1930 / 2000 [ 96%]  (Sampling)
## Chain 1: Iteration: 1940 / 2000 [ 97%]  (Sampling)
## Chain 1: Iteration: 1950 / 2000 [ 97%]  (Sampling)
## Chain 1: Iteration: 1960 / 2000 [ 98%]  (Sampling)
## Chain 1: Iteration: 1970 / 2000 [ 98%]  (Sampling)
## Chain 1: Iteration: 1980 / 2000 [ 99%]  (Sampling)
## Chain 1: Iteration: 1990 / 2000 [ 99%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 663.109 seconds (Warm-up)
## Chain 1:                473.875 seconds (Sampling)
## Chain 1:                1136.98 seconds (Total)
## Chain 1:

## Warning: There were 61 divergent transitions after warmup. See
## https://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
## to find out why this is a problem and how to eliminate them.

## Warning: Examine the pairs() plot to diagnose sampling problems

## Warning: The largest R-hat is 1.07, indicating chains have not mixed.
## Running the chains for more iterations may help. See
## https://mc-stan.org/misc/warnings.html#r-hat

## Warning: Bulk Effective Samples Size (ESS) is too low, indicating posterior means and medians may be
## Running the chains for more iterations may help. See
## https://mc-stan.org/misc/warnings.html#bulk-ess

## Warning: Tail Effective Samples Size (ESS) is too low, indicating posterior variances and tail quanti
## Running the chains for more iterations may help. See
## https://mc-stan.org/misc/warnings.html#tail-ess
```

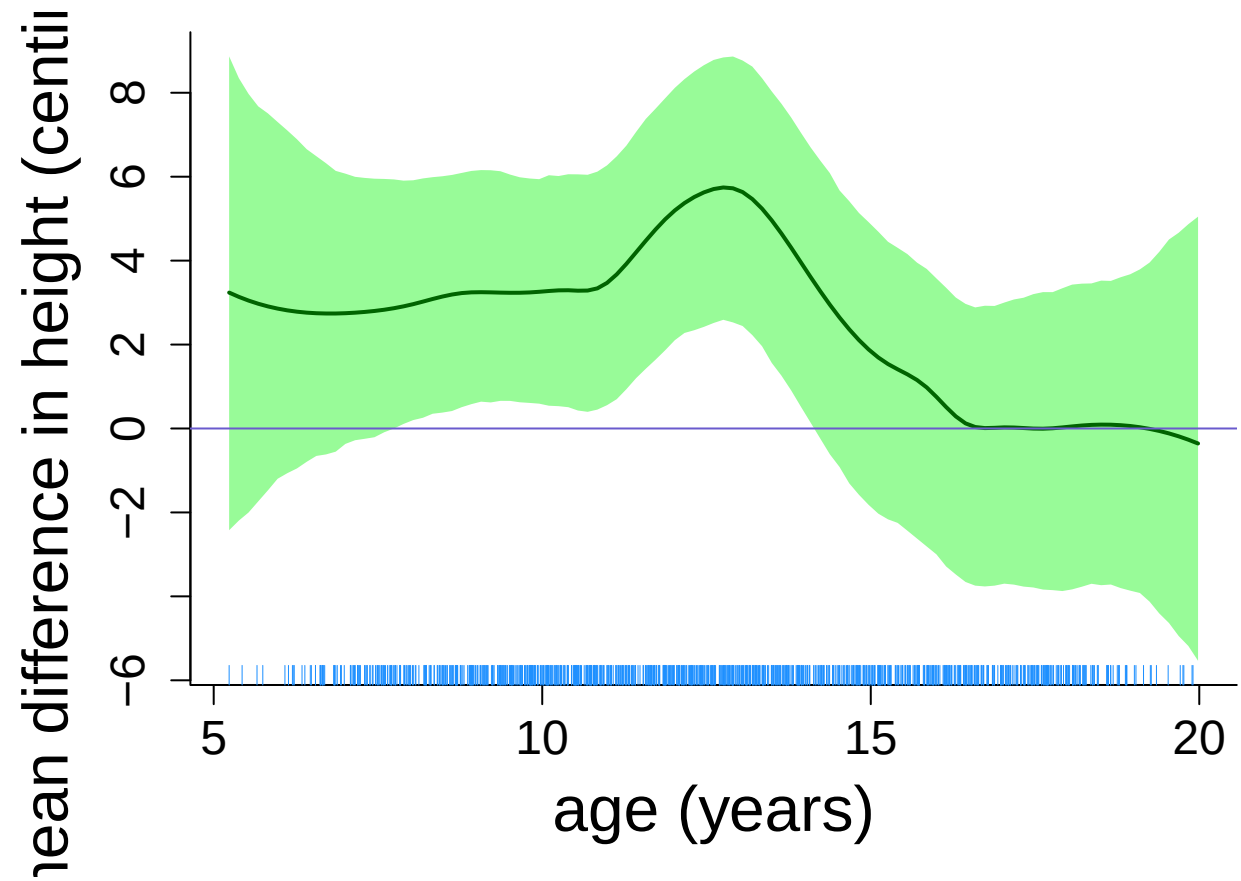
```
# Save and extract relevant MCMC samples:

betaMCMC <- NULL  ;  betaBvsAMCMC <- NULL
for (i in 1:2)
{
   charVar1 <- paste("beta[",as.character(i),"]",sep="")
   betaMCMC <- rbind(betaMCMC,extract(stanObj,charVar1,permuted=FALSE))
   charVar2 <- paste("betaBvsA[",as.character(i),"]",sep="")
   betaBvsAMCMC <- rbind(betaBvsAMCMC,extract(stanObj,charVar2,permuted=FALSE))
}

uGblAMCMC <- NULL
for (k in 1:ncZ)
{
   charVar <- paste("uGblA[",as.character(k),"]",sep="")
   uGblAMCMC <- rbind(uGblAMCMC,extract(stanObj,charVar,permuted=FALSE))
}
uGblBMCMC <- NULL
for (k in 1:ncZ)
{
   charVar <- paste("uGblB[",as.character(k),"]",sep="")
   uGblBMCMC <- rbind(uGblBMCMC,extract(stanObj,charVar,permuted=FALSE))
}
siguGblAMCMC <- as.vector(extract(stanObj,"siguGblA",permuted=FALSE))
siguGblBMCMC <- as.vector(extract(stanObj,"siguGblB",permuted=FALSE))
siguGrpMCMC <- as.vector(extract(stanObj,"siguGrp",permuted=FALSE))
sigEpsMCMC <- as.vector(extract(stanObj,"sigEps",permuted=FALSE))

UMCMC <- NULL
for (i in 1:numGrp)
   for (j in 1:2)
   {
      charVar <- paste("U[",as.character(i),",",as.character(j),"]",sep="")
      UMCMC <- rbind(UMCMC,as.vector(extract(stanObj,charVar,permuted=FALSE)))
   }

uGrpMCMC <- NULL
for (i in 1:numGrp)
   for (j in 1:ncZgrp)
   {
      charVar <- paste("uGrp[",as.character(i),",",as.character(j),"]",sep="")
      uGrpMCMC <- rbind(uGrpMCMC,as.vector(extract(stanObj,charVar,permuted=FALSE)))
   }

# Do lattice-type graphic showing fitted group-specific curves:

AptCol <- "dodgerblue"   ;   AlnCol <- "blue"
BptCol <- "deeppink"     ;   BlnCol <- "maroon"
lwdVal <- 1.5

ng <- 101
xg <- seq(min(x),max(x),length=ng)
Xg <- cbind(rep(1,ng),xg)
```

```
Zg <- ZOSull(xg,intKnots=intKnots,range.x=range.x)
Zggrp <- ZOSull(xg,intKnots=intKnotsGrp,range.x=range.x)

fAgMCMC <- Xg%*%betaMCMC + Zg%*%uGblAMCMC
fBgMCMC <- Xg%*%(betaMCMC + betaBvsAMCMC) + Zg%*%uGblBMCMC

meanCrvs <- vector("list",numGrp)
credLows <- vector("list",numGrp)
credUpps <- vector("list",numGrp)
ggMCMC <- vector("list",numGrp)
for (i in 1:numGrp)
{
   # Determine MCMC sample of group-specific deviation
   # grids for group i:

   ggMCMC[[i]] <- (Xg%*%rbind(UMCMC[(2*i-1),],UMCMC[(2*i),])
                   + Zggrp%*%uGrpMCMC[(i-1)*ncZgrp+1:ncZgrp,])

   # Determine if the current group is Type A or Type B:

   indsi <- (1:numObs)[idnum == i]
   currIsB <- typeIsB[indsi[1]]

   if (!currIsB)
      currMCMC <-  fAgMCMC + ggMCMC[[i]]

   if (currIsB)
      currMCMC <-  fBgMCMC + ggMCMC[[i]]

   meanCrvs[[i]] <- apply(currMCMC,1,mean)
   credLows[[i]] <- apply(currMCMC,1,quantile,0.025)
   credUpps[[i]] <- apply(currMCMC,1,quantile,0.975)
}

# Convert sigEpsMCMC to the original units:

sigEpsMCMCorig <-  sd.y*sigEpsMCMC

# Convert fAgMCMC, fBgMCMC to the original units:

fAgMCMCorig <-  mean.y + sd.y*fAgMCMC
fBgMCMCorig <-  mean.y + sd.y*fBgMCMC

# Convert the group-specific curves to the orginal units:

meanCrvsOrig <- vector("list",numGrp)
credLowsOrig <- vector("list",numGrp)
credUppsOrig <- vector("list",numGrp)

for (i in 1:numGrp)
{
   meanCrvsOrig[[i]] <- mean.y + sd.y*meanCrvs[[i]]
   credLowsOrig[[i]] <- mean.y + sd.y*credLows[[i]]
```

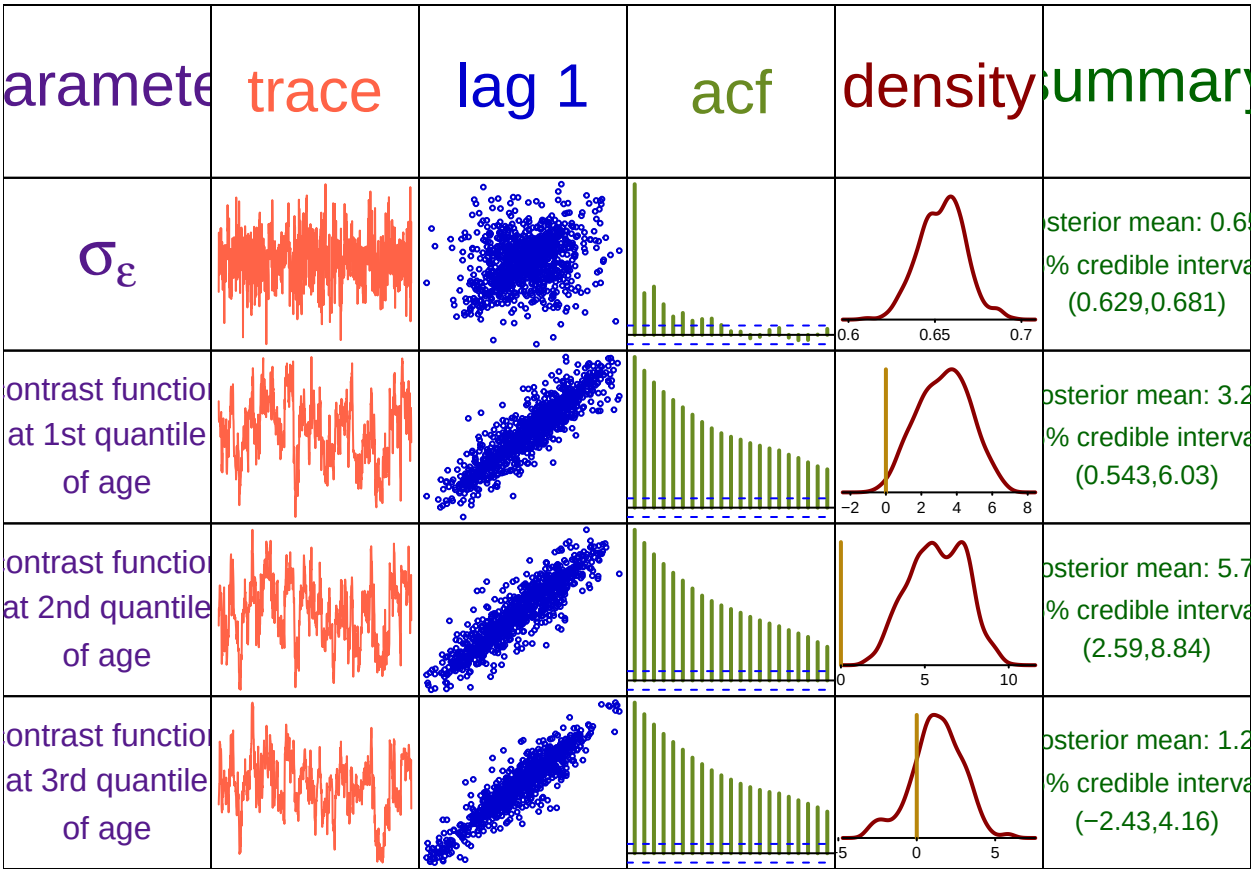
```
   credUppsOrig[[i]] <- mean.y + sd.y*credUpps[[i]]
}

# Convert the horizontal plotting grid to the original units:

xgOrig <- mean.x + sd.x*xg

allDataDF <- data.frame(xOrig,yOrig,idnum,typeIsB)

cex.labVal <- 1.35
colourVec <- c("dodgerblue","deeppink","blue","darkmagenta","lightblue","pink")

figFit <- xyplot(yOrig~xOrig|idnum,groups = idnum,data = allDataDF,
                 layout = c(9,13),xlab = list(label = "age (years)",cex = cex.labVal),
                 ylab = list(label = "height (centimetres)",cex = cex.labVal),
                 cex.lab = 4,strip = FALSE,as.table = TRUE,
                 key = list(title = "",
                      columns = 2,
                      lines = list(lty = rep(1,2),col = colourVec[3:4]),
                      text = list(c("black","white"),cex = 1.55)),
                 panel = function(x,y,subscripts,groups)
                 {
                    panel.grid()
                    iGrp <- panel.number()
                    colourInd <- 2 - typeIsB[subscripts[1]]
                    panel.superpose(x,y,subscripts,groups,
                                    type = "b",col = colourVec[colourInd],
                                    pch = 1,cex = 0.5)
                    panel.polygon(c(xgOrig,rev(xgOrig)),
                                  c(credLowsOrig[[iGrp]],
                                  rev(credUppsOrig[[iGrp]])),
                                  col = colourVec[colourInd+4],border = FALSE)
                    panel.xyplot(xgOrig,meanCrvsOrig[[iGrp]],
                                 col = colourVec[colourInd+2],
                                 type = "l",lwd = lwdVal)

                 })
print(figFit)
```




```
# Do contrast function plot:

ContgMCMCorig <- fBgMCMCorig - fAgMCMCorig
ContgMeanOrig <- apply(ContgMCMCorig,1,mean)
ContgLowerOrig <- apply(ContgMCMCorig,1,quantile,0.025)
ContgUpperOrig <- apply(ContgMCMCorig,1,quantile,0.975)

cex.labVal <- 2 ; cex.axisVal <- 1.5
ylimVal <- range(c(ContgLowerOrig,ContgUpperOrig))
par(mai = c(1,0.95,0.1,0.1))
plot(xgOrig,ContgMeanOrig,bty = "l",type = "n",xlab = "age (years)",
     ylab = "mean difference in height (centimetres)",
     ylim = ylimVal,cex.lab = cex.labVal,cex.axis = cex.axisVal)

polygon(c(xgOrig,rev(xgOrig)),c(ContgLowerOrig,rev(ContgUpperOrig)),
         col = "palegreen",border = FALSE)
lines(xgOrig,ContgMeanOrig,col = "darkgreen",lwd = 2)
abline(0,0,col = "slateblue")
nSub <- 1000
subInds <- sample(1:length(x),nSub,replace = FALSE)
rug(xOrig[subInds],col = "dodgerblue")
```




```
# Do MCMC summary plot to check convergence:

indQ1 <- length(xgOrig[xgOrig<quantile(xOrig,0.25)])
indQ2 <- length(xgOrig[xgOrig<quantile(xOrig,0.50)])
indQ3 <- length(xgOrig[xgOrig<quantile(xOrig,0.75)])
ContQ1MCMCorig <- ContgMCMCorig[indQ1,]
ContQ2MCMCorig <- ContgMCMCorig[indQ2,]
ContQ3MCMCorig <- ContgMCMCorig[indQ3,]

MCMClist <- list(cbind(sigEpsMCMCorig,ContQ1MCMCorig,ContQ2MCMCorig,ContQ3MCMCorig))
parNamesVal <- list(expression(sigma[epsilon]),
                    c("contrast function","at 1st quantile","of age"),
                    c("contrast function","at 2nd quantile","of age"),
                    c("contrast function","at 3rd quantile","of age"))
summMCMC(MCMClist,parNames = parNamesVal)
```




```
############ End of maleGrowthIndianaBayes ############
```

In the diagnostic plots , we can see that the trace behaves normally and is as usual. We can also see in the lag scatterplot , we have a linear relationship which might be a bit concerning for all the distributions. Transforming the variables might fix it ? I am not really sure

c

4.1

a

```
########## R script: femSBMDbayes ##########

# For conducting a Bayesian additive mixed
# model analysis on the spinal bone mineral
# density data using Markov chain Monte Carlo
# and Stan.

# Last changed: 29 AUG 2018

# Set flag for code compilation (needed if
# running script first time in current session):
```

```
compileCode <- TRUE

# Set MCMC sample size paramaters:

nWarm <- 1000
nKept <- 1000
nThin <- 1

# Load required packages:

library(HRW)   ;   library(rstan)   ;   library(lattice)

# Load in the female spinal bone data and extract the
# component variables:

data(femSBMD)
femSBMD$asian<-ifelse(femSBMD$ethnicity=='Asian',1,0)
idnum <- femSBMD$idnum
x1 <- femSBMD$black
x2 <- femSBMD$hispanic
x3 <- femSBMD$white
x0 <- femSBMD$asian
x4Orig <- femSBMD$age
yOrig <- femSBMD$spnbmd

# Standardise data for Bayesian analysis:

mean.x4 <- mean(x4Orig) ; sd.x4 <- sd(x4Orig)
mean.y <- mean(yOrig)   ; sd.y <- sd(yOrig)
x4 <- (x4Orig - mean.x4)/sd.x4
y <- (yOrig - mean.y)/sd.y

# Set up matrices for additive model:

numObs <- length(y)
X <- cbind(rep(1,numObs),x0,x1,x2,x3,x4)
ncX <- ncol(X)

numIntKnots <- 15
intKnots <- quantile(unique(x4),seq(0,1,length =
                (numIntKnots+2))[-c(1,(numIntKnots+2))])
range.x4 <- c(1.01*min(x4)-0.01*max(x4),1.01*max(x4)-0.01*min(x4))
Zspl <- ZOSull(x4,intKnots = intKnots,range.x = range.x4)
ncZ <- ncol(Zspl)
Zspl_asian<-X[,2]*Zspl;Zspl_black<-X[,3]*Zspl;
Zspl_hispanic<-X[,4]*Zspl;Zspl_white<-X[,5]*Zspl;

numGrp <- length(unique(idnum))
numObs <- length(y)

# Set hyperparameters:

sigmaBeta <- 1e5 ; AU <- 1e5 ; Au_asian <- 1e5 ; Au_black <- 1e5 ; Au_hispanic <- 1e5 ; Au_white <- 1e5
```

```
# Specify model in Stan:

addMixModModel <-
'data
{
   int<lower=1> numObs;          int<lower=1> numGrp;
   int<lower=1> ncX;             int<lower=1> ncZ;
   real<lower=0> sigmaBeta;      real<lower=0> AU;
   real<lower=0> Aeps;           real<lower=0> Au_asian;
   vector[numObs] y;             int<lower=1> idnum[numObs];
   matrix[numObs,ncX] X;         matrix[numObs,ncZ] Zspl_asian;
   real<lower=0> Au_black;            real<lower=0> Au_hispanic;
   real<lower=0> Au_white;       matrix[numObs,ncZ] Zspl_black;
   matrix[numObs,ncZ] Zspl_hispanic;  matrix[numObs,ncZ] Zspl_white;
}
parameters
{
   vector[ncX] beta;             vector[numGrp] U;
   vector[ncZ] u_asian;          real<lower=0> sigmaU;
   real<lower=0> sigmau_asian;        real<lower=0> sigmaEps;
   vector[ncZ] u_black;          vector[ncZ] u_hispanic;
   vector[ncZ] u_white;          real<lower=0> sigmau_black;
   real<lower=0> sigmau_hispanic;        real<lower=0> sigmau_white;
}
model
{
   y ~ normal(X*beta + U[idnum] + Zspl_asian*u_asian+Zspl_black*u_black+Zspl_hispanic*u_hispanic+Zspl_wh
   U ~ normal(0,sigmaU);            u_asian ~ normal(0,sigmau_asian);
   beta  ~ normal(0,sigmaBeta) ;    sigmaEps ~ cauchy(0,Aeps);
   sigmaU ~ cauchy(0,AU) ;    sigmau_asian ~ cauchy(0,Au_asian);
   u_black ~ normal(0,sigmau_black);  sigmau_black ~ cauchy(0,Au_black);
   u_hispanic ~ normal(0,sigmau_hispanic);  sigmau_hispanic ~ cauchy(0,Au_hispanic);
   u_white ~ normal(0,sigmau_white);  sigmau_white ~ cauchy(0,Au_white);
}'
# Zspl*u_black*X[,3]+Zspl*u_hispanic*X[,3]+Zspl*u_white*X[,4]
# Fit model using MCMC via Stan:

allData <- list(numObs = numObs,numGrp = numGrp,ncX = ncX,ncZ = ncZ,
                idnum = idnum,X = X,y = y,Zspl_asian = Zspl_asian,sigmaBeta = sigmaBeta,
                AU = AU,Au_asian = Au_asian,Au_black = Au_black,Au_hispanic = Au_hispanic,
                Au_white = Au_white,Aeps = Aeps,Zspl_black = Zspl_black,
                Zspl_hispanic = Zspl_hispanic,Zspl_white = Zspl_white)

# Compile code for model if required:

if (compileCode)
   stanCompilObj <- stan(model_code = addMixModModel,data = allData,
                         iter = 1,chains = 1)
```

```
## Trying to compile a simple C file

## Running /Library/Frameworks/R.framework/Resources/bin/R CMD SHLIB foo.c
## clang -mmacosx-version-min=10.13 -I"/Library/Frameworks/R.framework/Resources/include" -DNDEBUG   -I
```

```
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/4.2/Resources/library/StanHeaders/incl
## In file included from /Library/Frameworks/R.framework/Versions/4.2/Resources/library/RcppEigen/inclu
## In file included from /Library/Frameworks/R.framework/Versions/4.2/Resources/library/RcppEigen/inclu
## /Library/Frameworks/R.framework/Versions/4.2/Resources/library/RcppEigen/include/Eigen/src/Core/util/
## namespace Eigen {
## ^
## /Library/Frameworks/R.framework/Versions/4.2/Resources/library/RcppEigen/include/Eigen/src/Core/util/
## namespace Eigen {
##                 ^
##                 ;
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/4.2/Resources/library/StanHeaders/incl
## In file included from /Library/Frameworks/R.framework/Versions/4.2/Resources/library/RcppEigen/inclu
## /Library/Frameworks/R.framework/Versions/4.2/Resources/library/RcppEigen/include/Eigen/Core:96:10: fa
## #include <complex>
##          ^~~~~~~~
## 3 errors generated.
## make: *** [foo.o] Error 1
##
## SAMPLING FOR MODEL '7a08aa170de6b569efa14bf9f6fa78f2' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0.000709 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 7.09 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: WARNING: No variance estimation is
## Chain 1:          performed for num_warmup < 20
## Chain 1:
## Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 1e-06 seconds (Warm-up)
## Chain 1:                0.000688 seconds (Sampling)
## Chain 1:                0.000689 seconds (Total)
## Chain 1:
```

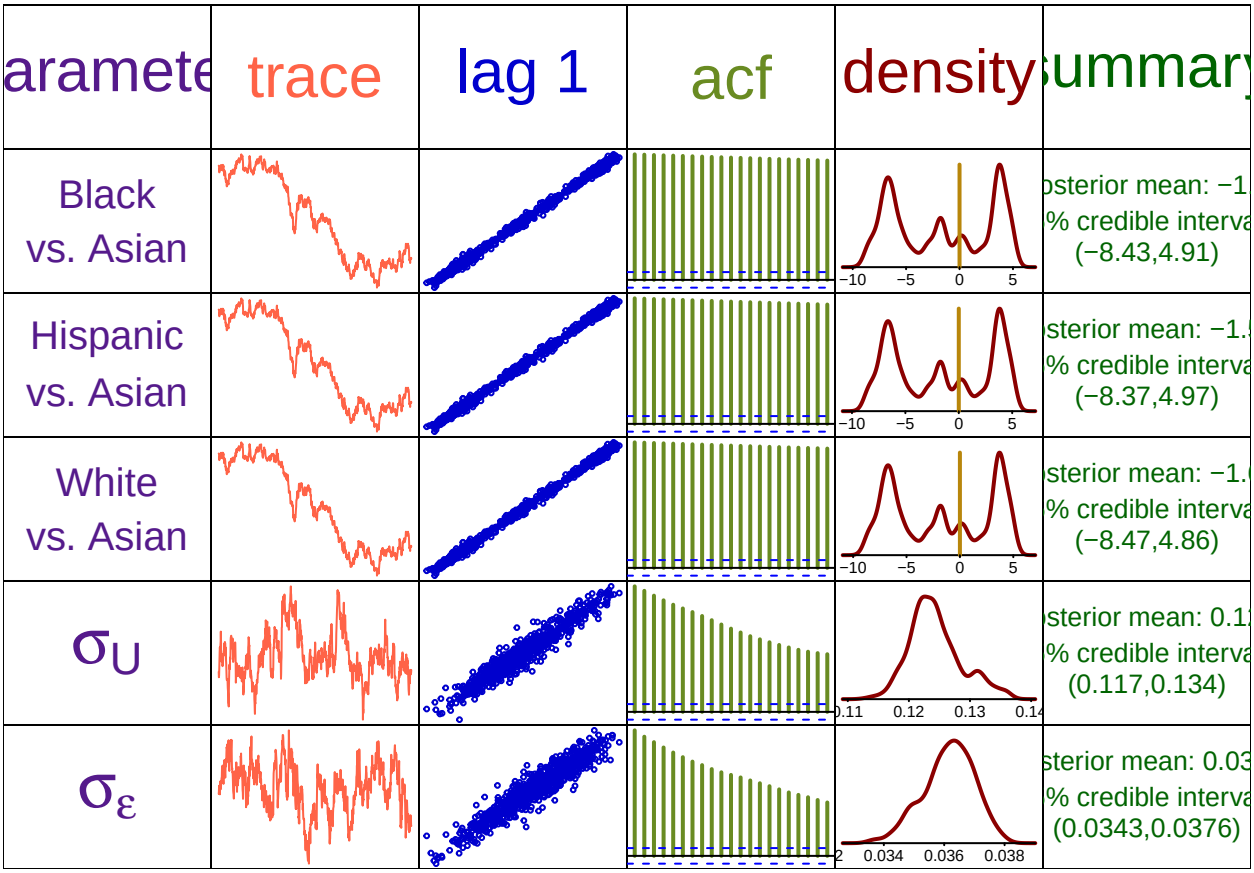
```
# Perform MCMC:

stanObj <-  stan(model_code = addMixModModel,data = allData,warmup = nWarm,
                 iter = (nWarm + nKept),chains = 1,thin = nThin,refresh = 100,
                 fit = stanCompilObj)
```

```
##
## SAMPLING FOR MODEL '7a08aa170de6b569efa14bf9f6fa78f2' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0.000264 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 2.64 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:  100 / 2000 [  5%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
```

```
## Chain 1: Iteration:  300 / 2000 [ 15%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  500 / 2000 [ 25%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  700 / 2000 [ 35%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration:  900 / 2000 [ 45%]  (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1100 / 2000 [ 55%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1300 / 2000 [ 65%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1500 / 2000 [ 75%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1700 / 2000 [ 85%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 1900 / 2000 [ 95%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 214.249 seconds (Warm-up)
## Chain 1:                251.139 seconds (Sampling)
## Chain 1:                465.388 seconds (Total)
## Chain 1:
```

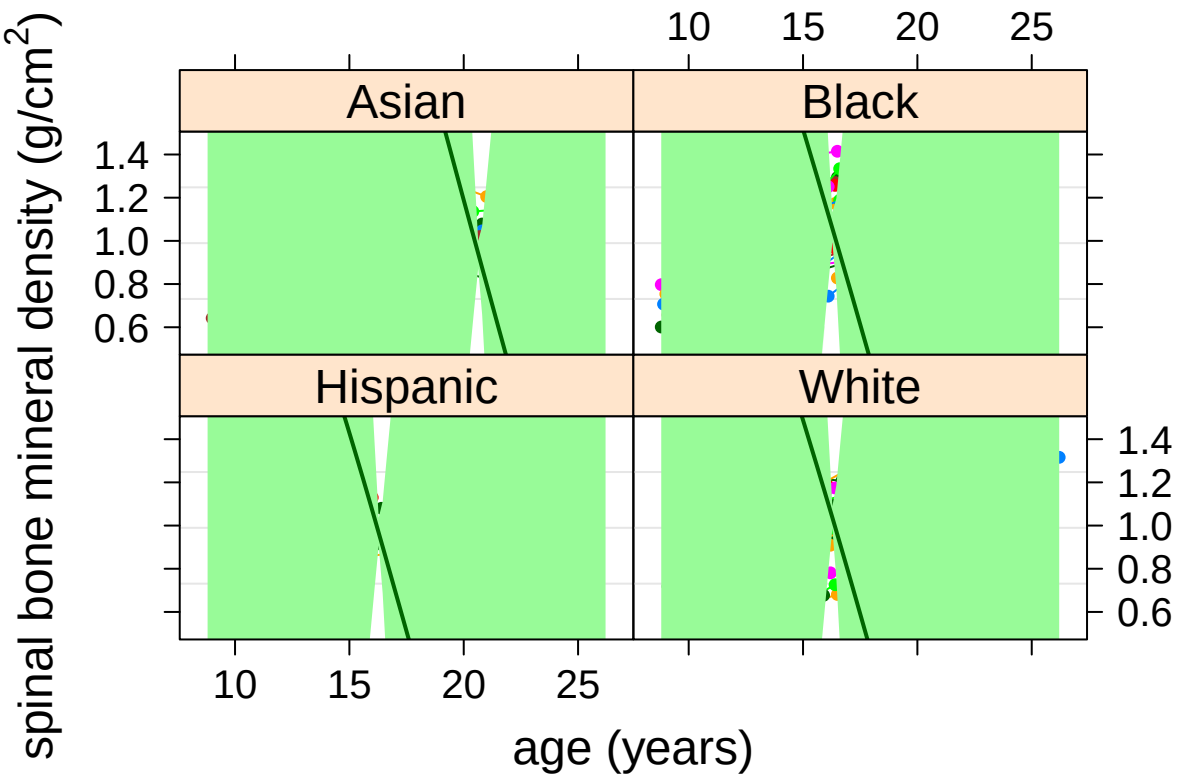
```r
# Save and extract relevant MCMC samples:

beta0MCMC <- as.vector(extract(stanObj,"beta[1]",permuted = FALSE))
beta1MCMC <- as.vector(extract(stanObj,"beta[2]",permuted = FALSE))
beta2MCMC <- as.vector(extract(stanObj,"beta[3]",permuted = FALSE))
beta3MCMC <- as.vector(extract(stanObj,"beta[4]",permuted = FALSE))
beta4MCMC <- as.vector(extract(stanObj,"beta[5]",permuted = FALSE))

sigmaUMCMC <- as.vector(extract(stanObj,"sigmaU",permuted = FALSE))
sigmaEpsMCMC <- as.vector(extract(stanObj,"sigmaEps",permuted = FALSE))


UMCMC <- NULL
for (k in 1:numGrp)
{
   charVar <- paste("U[",as.character(k),"]",sep = "")
   UMCMC <- rbind(UMCMC,extract(stanObj,charVar,permuted = FALSE))
}

uMCMC_asian <- NULL
for (k in 1:ncZ)
{
   charVar <- paste("u_asian[",as.character(k),"]",sep = "")
   uMCMC_asian <- rbind(uMCMC_asian,extract(stanObj,charVar,permuted = FALSE))
}

uMCMC_black <- NULL
for (k in 1:ncZ)
{
```

```
  charVar <- paste("u_black[",as.character(k),"]",sep = "")
  uMCMC_black <- rbind(uMCMC_black,extract(stanObj,charVar,permuted = FALSE))
}

uMCMC_hispanic <- NULL
for (k in 1:ncZ)
{
  charVar <- paste("u_hispanic[",as.character(k),"]",sep = "")
  uMCMC_hispanic <- rbind(uMCMC_hispanic,extract(stanObj,charVar,permuted = FALSE))
}

uMCMC_white <- NULL
for (k in 1:ncZ)
{
  charVar <- paste("u_white[",as.character(k),"]",sep = "")
  uMCMC_white <- rbind(uMCMC_white,extract(stanObj,charVar,permuted = FALSE))
}

# Convert to parameters original scale:

beta1MCMCorig <- beta1MCMC*sd.y
beta2MCMCorig <- beta2MCMC*sd.y
beta3MCMCorig <- beta3MCMC*sd.y
beta4MCMCorig <- beta4MCMC*(sd.y/sd.x4)

beta0MCMCorig <-  mean.y + sd.y*beta0MCMC -  mean.x4*beta4MCMCorig

sigmaUMCMCorig <- sigmaUMCMC*sd.y
sigmaEpsMCMCorig <- sigmaEpsMCMC*sd.y

# Do parameters plot:

parms <- list(cbind(beta1MCMCorig,beta2MCMCorig,beta3MCMCorig,
                    sigmaUMCMCorig,sigmaEpsMCMCorig))
parNamesVal <- list(c("Black","vs. Asian"),c("Hispanic","vs. Asian"),
                      c("White","vs. Asian"),c(expression(sigma[U])),
                      c(expression(sigma[epsilon])))
summMCMC(parms,parNames = parNamesVal)
```




```
# Do fitted curves using lattice graphics:

ng <- 101
ylim.val <- c(min(yOrig),max(yOrig))
x4g <- seq(min(x4),max(x4),length = ng)
Xg <- cbind(rep(1,ng),x4g)
Zg <- ZOSull(x4g,intKnots = intKnots,range.x = range.x4)

midCurvsg <- list()
lowCurvsg <- list()
uppCurvsg <- list()

for (ip in 1:4)
{
   if (ip == 1){
     betaSplMCMC <- rbind(beta0MCMC,beta1MCMC)

     fMCMC <-   Xg%*%betaSplMCMC + Zg%*%uMCMC_asian
   }
   if (ip == 2){
     betaSplMCMC <- rbind(beta0MCMC+beta2MCMC,beta1MCMC)
     fMCMC <-   Xg%*%betaSplMCMC + Zg%*%uMCMC_black

   }
   if (ip == 3){
     betaSplMCMC <- rbind(beta0MCMC+beta3MCMC,beta1MCMC)
```

```
    fMCMC <-   Xg%*%betaSplMCMC + Zg%*%uMCMC_hispanic

  }
  if (ip == 4){
    betaSplMCMC <- rbind(beta0MCMC+beta4MCMC,beta1MCMC)
    fMCMC <-   Xg%*%betaSplMCMC + Zg%*%uMCMC_white

  }


  fMCMCorig <- fMCMC*sd.y + mean.y
  credLowerOrig <- apply(fMCMCorig,1,quantile,0.025)
  credUpperOrig <- apply(fMCMCorig,1,quantile,0.975)
  fhatgOrig <- apply(fMCMCorig,1,mean)

  x4gOrig <- x4g*sd.x4 + mean.x4

  midCurvsg[[ip]] <- apply(fMCMCorig,1,mean)
  lowCurvsg[[ip]] <- apply(fMCMCorig,1,quantile,0.025)
  uppCurvsg[[ip]] <- apply(fMCMCorig,1,quantile,0.975)
}

fitFig <- xyplot(yOrig~x4Orig|ethnicity,groups = idnum,
                 as.table = TRUE,data = femSBMD,
                 strip = strip.custom(par.strip.text = list(cex = 1.5)),
                 par.settings = list(layout.heights = list(strip = 1.6)),
                 scales = list(cex = 1.25),
                 xlab = list("age (years)",cex = 1.5),
                 ylab = list(expression(paste(
                 "spinal bone mineral density (g/c",m^2,")")),cex = 1.5),
                 subscripts = TRUE,
                 panel = function(x,y,subscripts,groups)
                 {
                    panel.grid()
                    if (any(femSBMD$ethnicity[subscripts] == "Asian"))    panNum <- 1
                    if (any(femSBMD$ethnicity[subscripts] == "Black"))    panNum <- 2
                    if (any(femSBMD$ethnicity[subscripts] == "Hispanic")) panNum <- 3
                    if (any(femSBMD$ethnicity[subscripts] == "White"))    panNum <- 4

                    panel.superpose(x,y,subscripts,groups,type = "b",pch = 16)

                    panel.polygon(c(x4gOrig,rev(x4gOrig)),c(lowCurvsg[[panNum]],rev(uppCurvsg[[panNum]]
                                  col = "palegreen",border = FALSE)

                    panel.xyplot(x4gOrig,midCurvsg[[panNum]],lwd = 2,type = "l",col = "darkgreen")
                 })

print(fitFig)
```

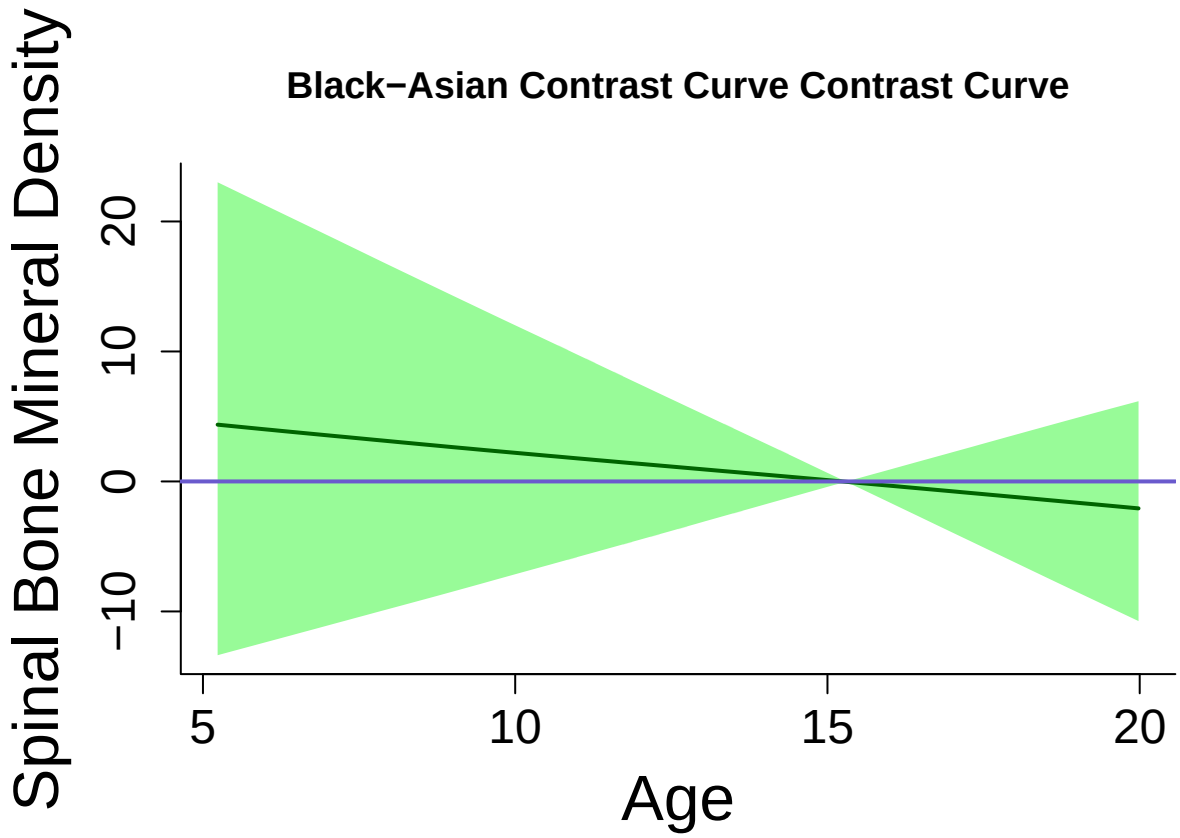



```
########## End of femSBMDbayes ##########
```

Black-Asian Contrast Curve

```
# Plot contrast curve fit:

estFunCol <- "darkgreen";
varBandCol <- "palegreen"

betaSplMCMC <- rbind(beta0MCMC,beta1MCMC)
ContrastMCMC1 <- Xg%*%betaSplMCMC + Zg%*%(uMCMC_black-uMCMC_asian)
credLower1 <- apply(ContrastMCMC1,1,quantile,0.025)
credUpper1 <- apply(ContrastMCMC1,1,quantile,0.975)
Contrastg1 <- apply(ContrastMCMC1,1,mean)

# Convert to original units:

ContrastgOrig1 <- sd.y*Contrastg1
credLowerOrig1 <- sd.y*credLower1
credUpperOrig1<- sd.y*credUpper1

par(mfrow = c(1,1),mai = c(1.02,0.9,0.82,0.42))
plot(0,0,type = "n",bty = "l",xlim = range(xgOrig),
     range(c(credLowerOrig1,credUpperOrig1)),
```

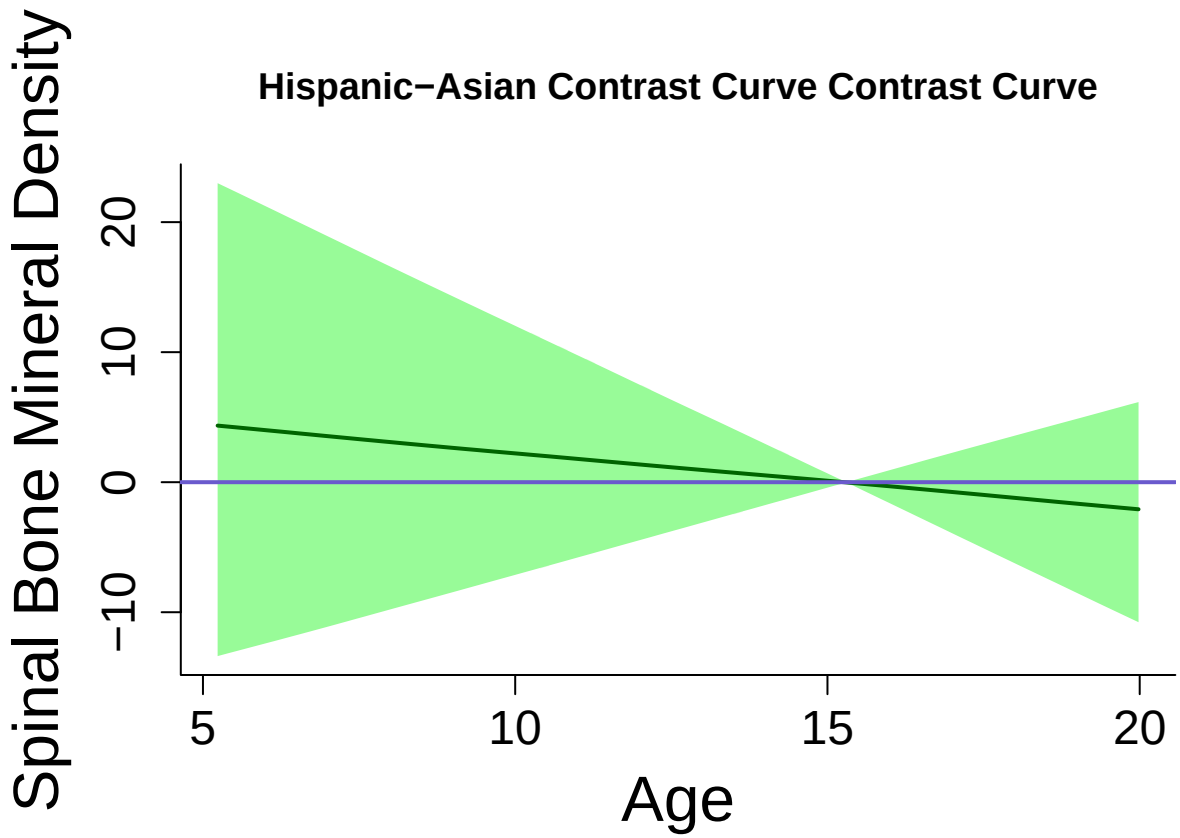
```
     main="Black-Asian Contrast Curve Contrast Curve",
     xlab = "Age",
     ylab = " Spinal Bone Mineral Density",
     cex.lab = cex.labVal,cex.axis = cex.axisVal)

polygon(c(xgOrig,rev(xgOrig)),c(credLowerOrig1,rev(credUpperOrig1)),
        col = varBandCol,border = FALSE)

lines(xgOrig,ContrastgOrig1,lwd = 2,col = estFunCol)
abline(0,0,col = "slateblue",lwd = 2)
```



Hispanic-Asian Contrast Curve

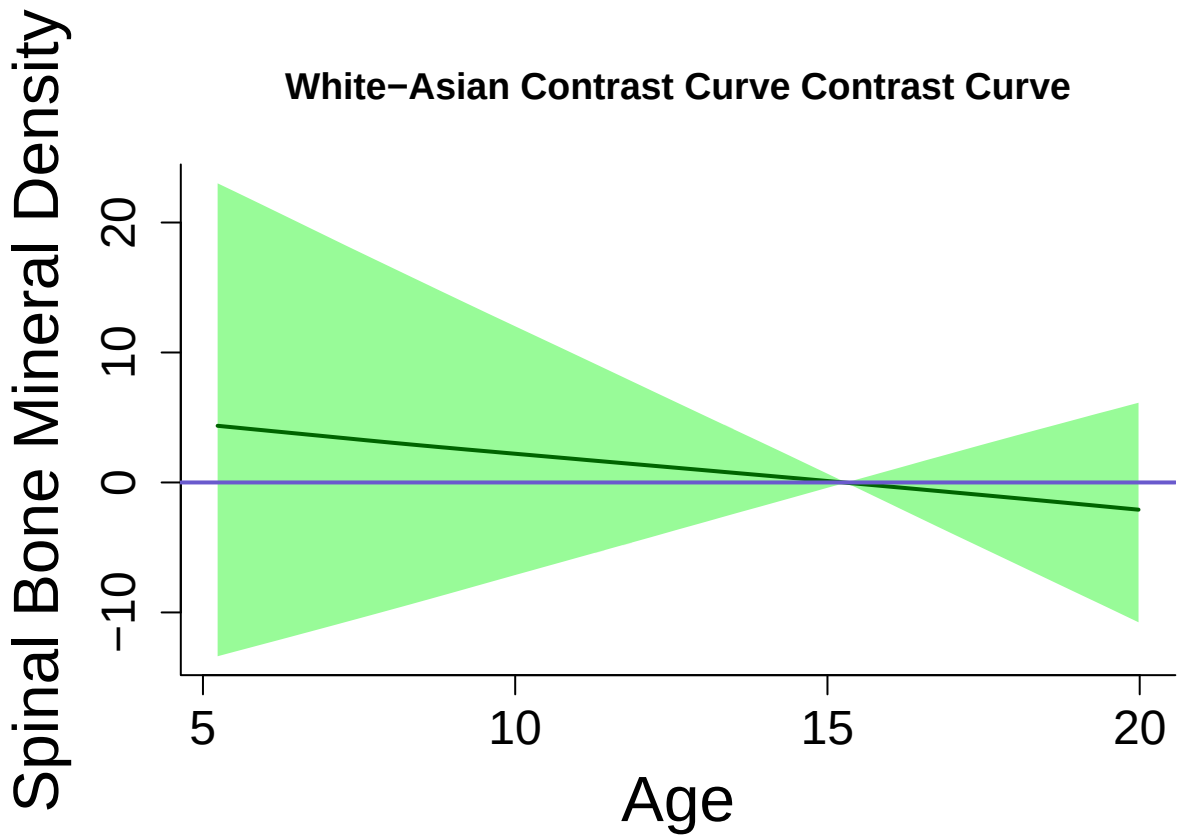
```
# Plot contrast curve fit:

estFunCol <- "darkgreen";
varBandCol <- "palegreen"
betaSplMCMC <- rbind(beta0MCMC,beta1MCMC)
ContrastMCMC2 <- Xg%*%betaSplMCMC + Zg%*%(uMCMC_hispanic-uMCMC_asian)

credLower2 <- apply(ContrastMCMC2,1,quantile,0.025)
credUpper2 <- apply(ContrastMCMC2,1,quantile,0.975)
Contrastg2 <- apply(ContrastMCMC2,1,mean)
```

```
# Convert to original units:

ContrastgOrig2 <- sd.y*Contrastg2
credLowerOrig2 <- sd.y*credLower2
credUpperOrig2<- sd.y*credUpper2

par(mfrow = c(1,1),mai = c(1.02,0.9,0.82,0.42))
plot(0,0,type = "n",bty = "l",xlim = range(xgOrig),
     range(c(credLowerOrig2,credUpperOrig2)),
     main="Hispanic-Asian Contrast Curve Contrast Curve",
     xlab = "Age",
     ylab = " Spinal Bone Mineral Density",
     cex.lab = cex.labVal,cex.axis = cex.axisVal)

polygon(c(xgOrig,rev(xgOrig)),c(credLowerOrig2,rev(credUpperOrig2)),
        col = varBandCol,border = FALSE)

lines(xgOrig,ContrastgOrig2,lwd = 2,col = estFunCol)
abline(0,0,col = "slateblue",lwd = 2)
```



White-Asian Contrast Curve

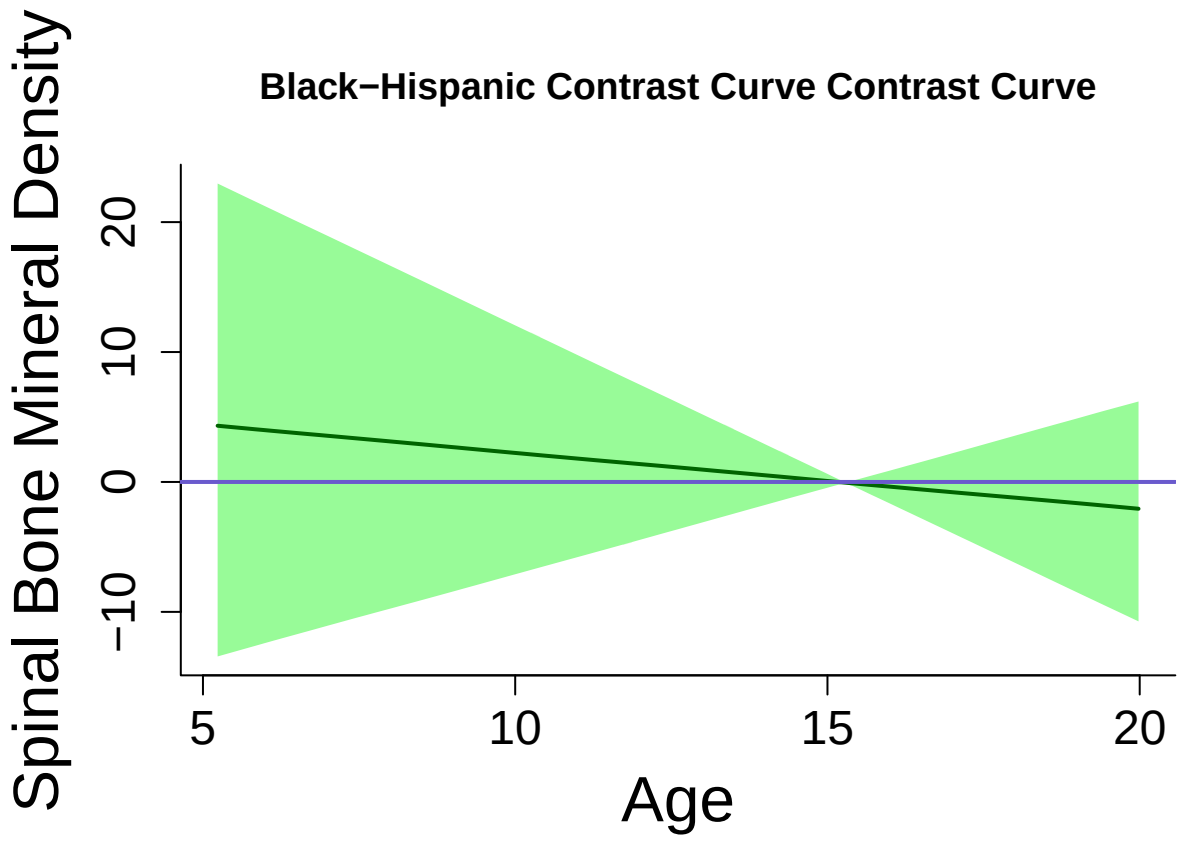
```
# Plot contrast curve fit:

estFunCol <- "darkgreen";
varBandCol <- "palegreen"
betaSplMCMC <- rbind(beta0MCMC,beta1MCMC)
ContrastMCMC3 <- Xg%*%betaSplMCMC + Zg%*%(uMCMC_white-uMCMC_asian)

credLower3 <- apply(ContrastMCMC3,1,quantile,0.025)
credUpper3 <- apply(ContrastMCMC3,1,quantile,0.975)
Contrastg3 <- apply(ContrastMCMC3,1,mean)

# Convert to original units:

ContrastgOrig3 <- sd.y*Contrastg3
credLowerOrig3 <- sd.y*credLower3
credUpperOrig3<- sd.y*credUpper3

par(mfrow = c(1,1),mai = c(1.02,0.9,0.82,0.42))
plot(0,0,type = "n",bty = "l",xlim = range(xgOrig),
     range(c(credLowerOrig3,credUpperOrig3)),
     main="White-Asian Contrast Curve Contrast Curve",
     xlab = "Age",
     ylab = " Spinal Bone Mineral Density",
     cex.lab = cex.labVal,cex.axis = cex.axisVal)

polygon(c(xgOrig,rev(xgOrig)),c(credLowerOrig3,rev(credUpperOrig3)),
        col = varBandCol,border = FALSE)

lines(xgOrig,ContrastgOrig3,lwd = 2,col = estFunCol)
abline(0,0,col = "slateblue",lwd = 2)
```



Black-Hispanic Contrast Curve

```
# Plot contrast curve fit:

estFunCol <- "darkgreen";
varBandCol <- "palegreen"
betaSplMCMC <- rbind(beta0MCMC,beta1MCMC)
ContrastMCMC4 <- Xg%*%betaSplMCMC + Zg%*%(uMCMC_black-uMCMC_hispanic)

credLower4 <- apply(ContrastMCMC4,1,quantile,0.025)
credUpper4 <- apply(ContrastMCMC4,1,quantile,0.975)
Contrastg4 <- apply(ContrastMCMC4,1,mean)

# Convert to original units:

ContrastgOrig4 <- sd.y*Contrastg4
credLowerOrig4 <- sd.y*credLower4
credUpperOrig4<- sd.y*credUpper4

par(mfrow = c(1,1),mai = c(1.02,0.9,0.82,0.42))
plot(0,0,type = "n",bty = "l",xlim = range(xgOrig),
     range(c(credLowerOrig4,credUpperOrig4)),
     main="Black-Hispanic Contrast Curve Contrast Curve",
     xlab = "Age",
```

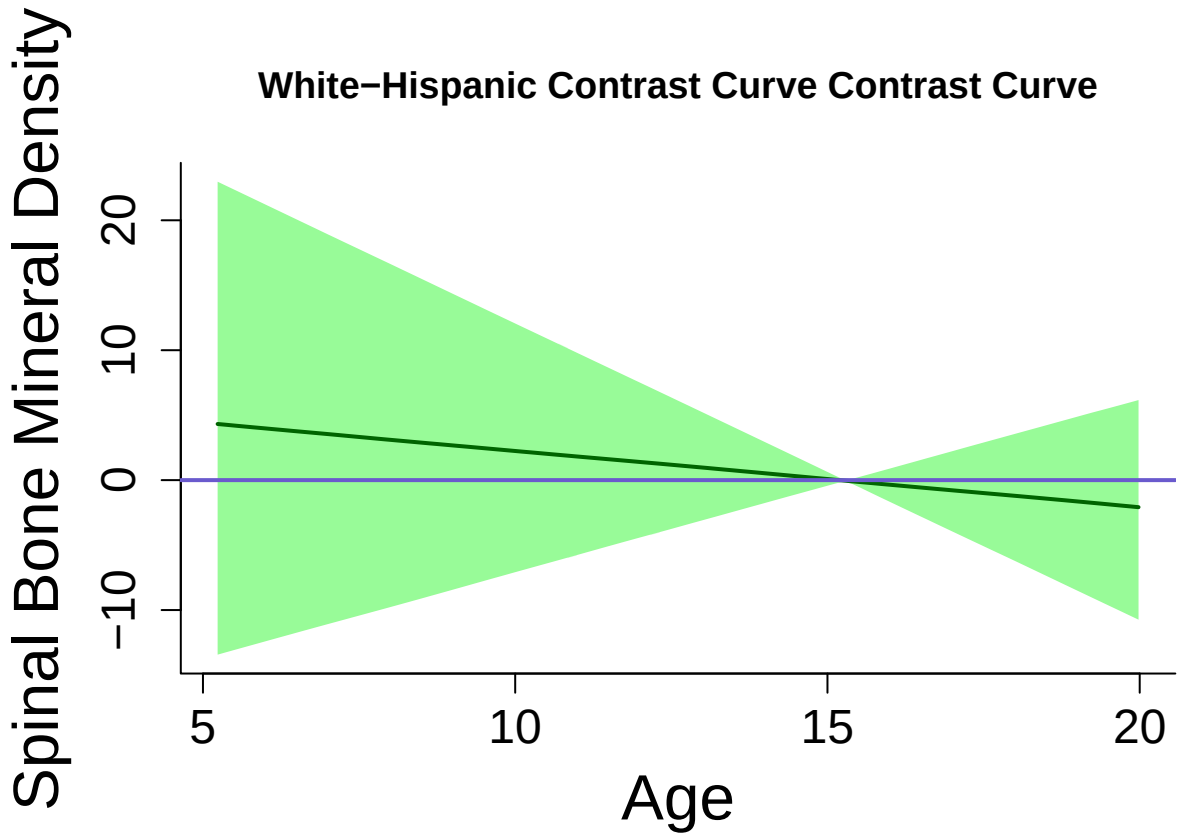
```
     ylab = " Spinal Bone Mineral Density",
     cex.lab = cex.labVal,cex.axis = cex.axisVal)

polygon(c(xgOrig,rev(xgOrig)),c(credLowerOrig4,rev(credUpperOrig4)),
        col = varBandCol,border = FALSE)

lines(xgOrig,ContrastgOrig4,lwd = 2,col = estFunCol)
abline(0,0,col = "slateblue",lwd = 2)
```



White-Hispanic Contrast Curve

```
# Plot contrast curve fit:

estFunCol <- "darkgreen";
varBandCol <- "palegreen"
betaSplMCMC <- rbind(beta0MCMC,beta1MCMC)
ContrastMCMC5 <- Xg%*%betaSplMCMC + Zg%*%(uMCMC_white-uMCMC_hispanic)

credLower5 <- apply(ContrastMCMC5,1,quantile,0.025)
credUpper5 <- apply(ContrastMCMC5,1,quantile,0.975)
Contrastg5 <- apply(ContrastMCMC5,1,mean)

# Convert to original units:
```

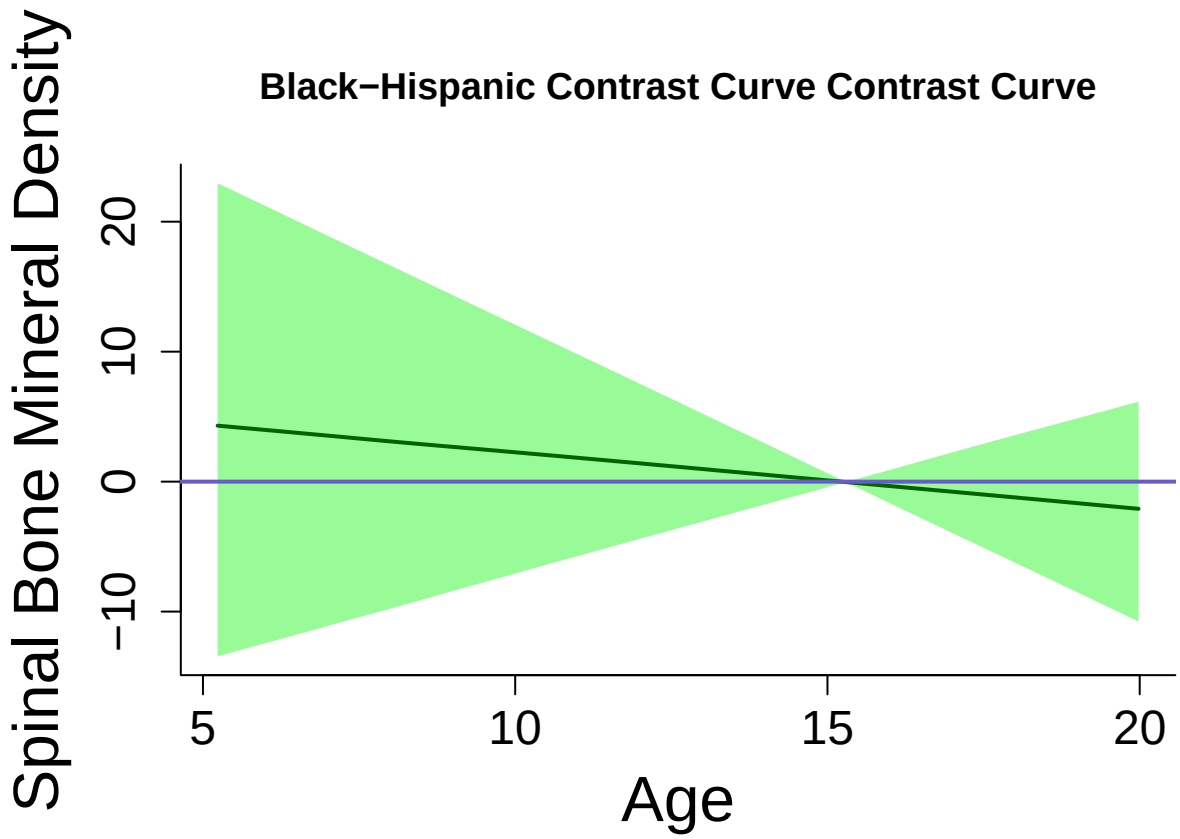
```
ContrastgOrig5 <- sd.y*Contrastg5
credLowerOrig5 <- sd.y*credLower5
credUpperOrig5<- sd.y*credUpper5

par(mfrow = c(1,1),mai = c(1.02,0.9,0.82,0.42))
plot(0,0,type = "n",bty = "l",xlim = range(xgOrig),
     range(c(credLowerOrig5,credUpperOrig5)),
     main="White-Hispanic Contrast Curve Contrast Curve",
     xlab = "Age",
     ylab = " Spinal Bone Mineral Density",
     cex.lab = cex.labVal,cex.axis = cex.axisVal)

polygon(c(xgOrig,rev(xgOrig)),c(credLowerOrig5,rev(credUpperOrig5)),
        col = varBandCol,border = FALSE)

lines(xgOrig,ContrastgOrig5,lwd = 2,col = estFunCol)
abline(0,0,col = "slateblue",lwd = 2)
```



White-Black Contrast Curve

```
# Plot contrast curve fit:

estFunCol <- "darkgreen";
```

```
varBandCol <- "palegreen"
betaSplMCMC <- rbind(beta0MCMC,beta1MCMC)
ContrastMCMC6 <- Xg%*%betaSplMCMC + Zg%*%(uMCMC_white-uMCMC_black)

credLower6 <- apply(ContrastMCMC6,1,quantile,0.025)
credUpper6 <- apply(ContrastMCMC6,1,quantile,0.975)
Contrastg6 <- apply(ContrastMCMC6,1,mean)

# Convert to original units:

ContrastgOrig6 <- sd.y*Contrastg6
credLowerOrig6 <- sd.y*credLower6
credUpperOrig6<- sd.y*credUpper6

par(mfrow = c(1,1),mai = c(1.02,0.9,0.82,0.42))
plot(0,0,type = "n",bty = "l",xlim = range(xgOrig),
     range(c(credLowerOrig6,credUpperOrig6)),
     main="Black-Hispanic Contrast Curve Contrast Curve",
     xlab = "Age",
     ylab = " Spinal Bone Mineral Density",
     cex.lab = cex.labVal,cex.axis = cex.axisVal)

polygon(c(xgOrig,rev(xgOrig)),c(credLowerOrig6,rev(credUpperOrig6)),
        col = varBandCol,border = FALSE)

lines(xgOrig,ContrastgOrig6,lwd = 2,col = estFunCol)
abline(0,0,col = "slateblue",lwd = 2)
```

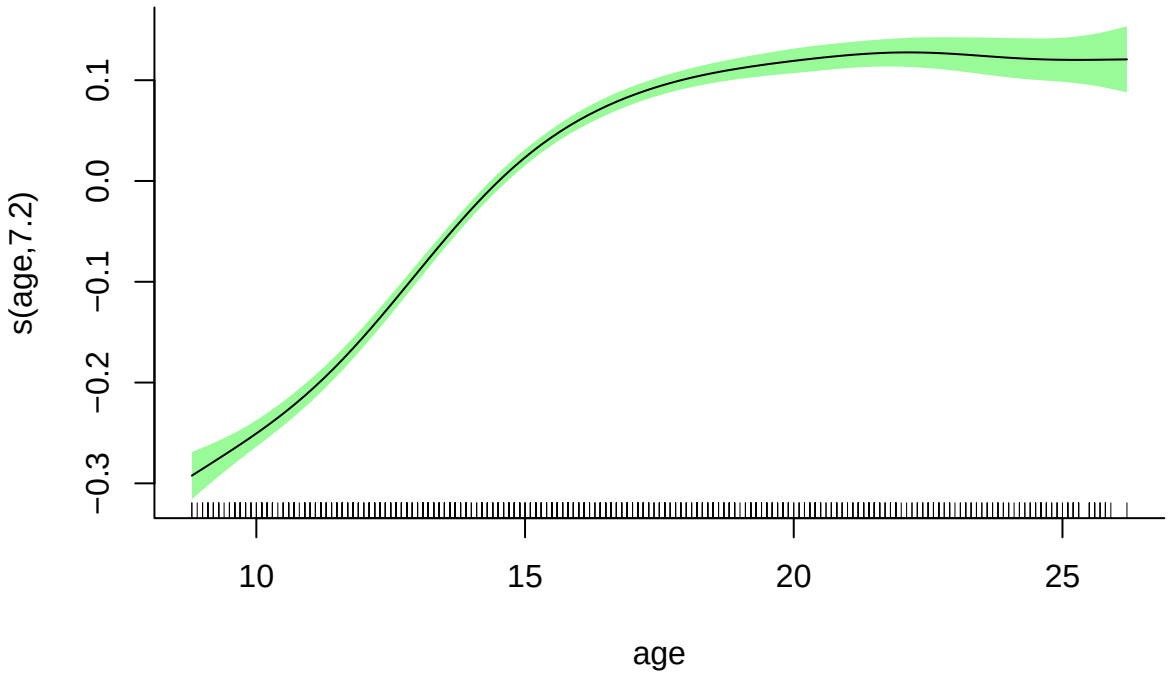


c

```
library(mgcv)
```

```
## This is mgcv 1.8-42. For overview type 'help("mgcv-package")'.
```

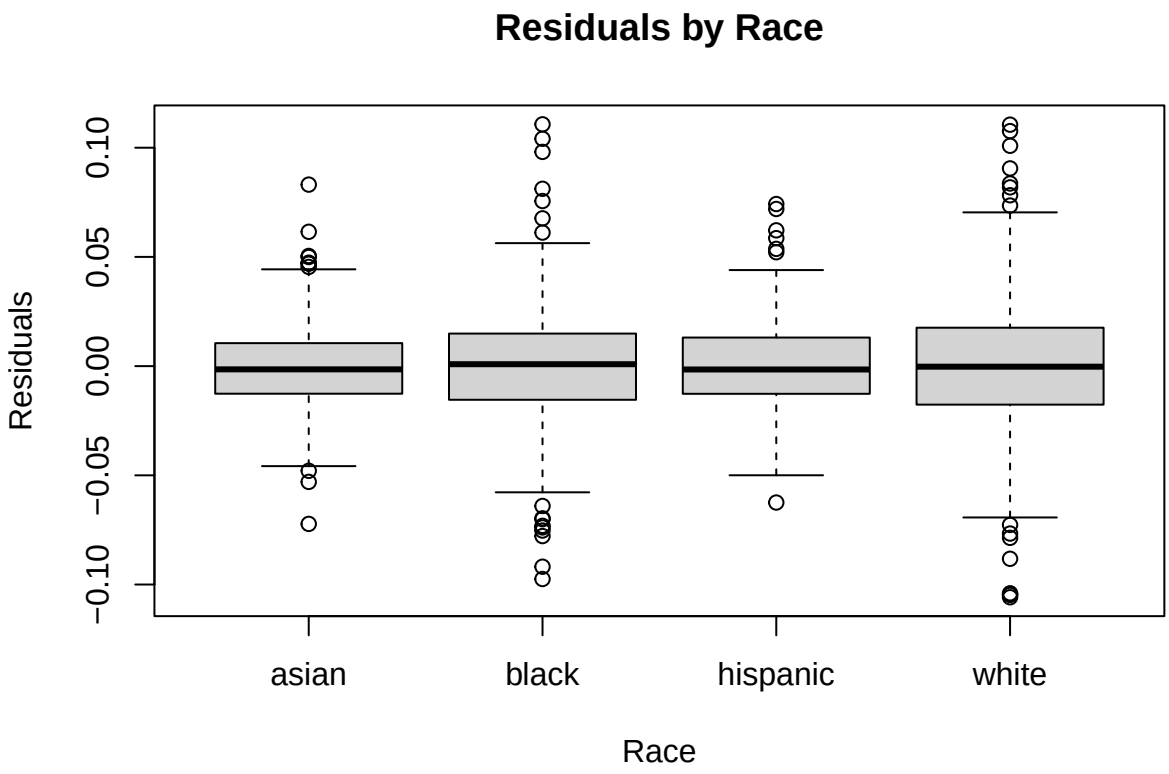
```
fit <- gamm(spnbmd~black+hispanic+white+s(age),random = list(idnum = ~1),data=femSBMD)
plot(fit$gam,shade = TRUE,shade.col = "palegreen",bty = "l")
```




```
summary(fit)
```

```
##     Length Class Mode
## lme 18     lme   list
## gam 31     gam   list
```

```
col_names <- colnames(femSBMD)[5:8]
femSBMD$race <- apply(femSBMD[, 5:8], 1, function(x) {paste(col_names[x == 1], collapse = "-")})
femSBMD$race <- factor(femSBMD$race)
res<-residuals(fit$lme)
boxplot( res ~ race, data = femSBMD, main = "Residuals by Race", xlab = "Race", ylab = "Residuals")
```



It seems that the race which we chose as the pivot fits the best as per the residual, in this case it is asian race as it has the least amount of extremums. Hispanic is close.

d

As we can see in the gamm approach we pivot one group as the referance group but this is not done on the MCMC approach. THis is the difference in gamm fitting vs mcmc fitting mechanism. The disadvantage of the gamm and the advantage of MCMC is that it allows more symmetrical measures the random effects for the group.

One advantage of gamm and the disadvantage of mcmc is that mcmc takes posterior distribution and samples from it which makes it computationally very time consuming whilen gam is moderately fast in comparison.