Ancestral state reconstruction (incl. diagnostics)

Beatriz Willink Rachel Blow

May 2020

Analysis of BayesTraits output

Output from each independent run should be saved into its own subdirectory within the BayesTraits directory.

Set your working directory to the parent directory containing all of your MCMC analyses

```
knitr::opts_knit$set(root.dir = '..' )
```

1) Read in data as mcmc objects

Read in your traces and make sure they are named correctly

```
path <- ("BayesTraits/")
directories <- c("run1/","run2/")
filename <- "IschMultiState.Log.txt"

for (i in directories){
  x = c(path, i, filename)
  assign(paste("run", which(directories==i), sep=""),
         as.data.frame(read.delim(paste(x, collapse =""), header=T, skip =
             129,
                                  na.strings = "--")))
}
```

Set the variables that will be used to create your MCMC object

```
thin <- run1$Iteration[2] - run1$Iteration[1]
start <- min(run1$Iteration)
end <- max(run1$Iteration)
```

Remove variables we are not interested in

```
output <- run1[ , !(names(run1) %in% c("X", "Model.string"))]
```

Divide rates by 100 to get realistic values (because ScaledTrees to 0.001 to prevent rates getting too small)

```
for(i in 6:ncol(output[,1:17])) {
  output[,i] <- output[,i]/100
}
```

Call the “coda” package for creating multiple mcmc objects

```
require(coda)
```

Create mcmc object

```
outputmc <- mcmc(data = output, start = start, end = end, thin = thin)
```

Calculate the percentage of iterations in which there was only a single parameter

```
(sum(outputmc[,4] == 1, na.rm = TRUE)/length(outputmc[,4]))*100
```

```
## [1] 92.1
```

Summarize each parameter into mean and HPD interval of percent support for each state per node

```
nodes_output <- outputmc[,22:181]
summary_table <- matrix(0,3,ncol(nodes_output))
rownames(summary_table) <- c("Mean", "LowerHPD", "UpperHPD")
colnames(summary_table) <- colnames(nodes_output)

for(i in 1:ncol(nodes_output)){
  summary_table[1,i] <- mean(nodes_output[,i], na.rm =TRUE)
  hpd <- HPDinterval(nodes_output[,i])
  summary_table[2,i] <- hpd[1]
  summary_table[3,i] <- hpd[2]
}
```

Reshape data into mean posterior value by node (row) and state (column)

```
mean_state_per_node <- data.frame("node" = 42:81, matrix(summary_table
   [1,], ncol = 4, byrow = TRUE))
colnames(mean_state_per_node) <- c("node", "FMA", "FPD", "FMH", "FPT")
print(mean_state_per_node)
```

```
##    node          FMA          FPD          FMH          FPT
## 1    42 6.565790e-02 7.504201e-02 8.101822e-01 0.0491178671
## 2    43 9.274950e-02 4.728775e-02 8.072559e-01 0.0527068645
## 3    44 5.120039e-02 2.728475e-02 8.809508e-01 0.0405640207
## 4    45 2.428807e-02 9.627171e-03 9.463384e-01 0.0197463551
## 5    46 3.531183e-02 1.580538e-02 9.208026e-01 0.0280801793
## 6    47 1.777336e-01 9.270024e-02 6.508177e-01 0.0787484276
## 7    48 2.463743e-01 1.411997e-01 5.161916e-01 0.0962344616
## 8    49 6.830667e-02 1.899536e-01 6.900512e-01 0.0516884563
## 9    50 4.539266e-01 8.295473e-02 3.791511e-01 0.0839675974
## 10   51 9.017519e-02 2.134361e-01 6.248634e-01 0.0715252852
## 11   52 7.370373e-02 7.672139e-02 7.984371e-01 0.0511377819
## 12   53 5.701542e-02 4.663767e-02 8.502526e-01 0.0460943197
## 13   54 7.428305e-02 8.988538e-02 7.773171e-01 0.0585144220
## 14   55 4.909961e-02 2.407956e-02 8.880385e-01 0.0387823455
## 15   56 1.440219e-01 1.880322e-01 5.867415e-01 0.0812043822
## 16   57 2.189587e-01 6.865205e-01 3.383505e-02 0.0606857260
## 17   58 1.877773e-02 9.503892e-01 1.187817e-02 0.0189549118
## 18   59 2.286832e-02 9.437721e-01 1.160026e-02 0.0217593602
## 19   60 4.599682e-03 9.919882e-01 7.357941e-04 0.0026762946
## 20   61 5.574331e-03 9.926921e-01 3.473139e-04 0.0013862433
## 21   62 2.074524e-02 9.777176e-01 1.973757e-04 0.0013397804
## 22   63 1.448938e-04 9.996814e-01 3.456607e-05 0.0001391503
## 23   64 4.213800e-01 5.763215e-01 2.195850e-04 0.0020788851
## 24   65 1.178999e-02 9.719373e-01 5.399090e-03 0.0108736573
## 25   66 4.907636e-02 4.437551e-02 8.674836e-01 0.0390645205
## 26   67 8.528026e-02 1.910135e-01 6.581645e-01 0.0655417725
## 27   68 2.977692e-02 1.298142e-02 9.334053e-01 0.0238363410
## 28   69 9.058939e-02 7.211650e-01 9.644028e-02 0.0918053710
## 29   70 5.035718e-02 8.524114e-01 4.144335e-02 0.0557881091
## 30   71 3.314916e-02 9.046198e-01 1.843635e-02 0.0437947336
## 31   72 2.809980e-02 9.144222e-01 1.122436e-02 0.0462536739
## 32   73 2.755034e-02 9.070550e-01 7.038726e-03 0.0583558772
## 33   74 6.801236e-02 6.955341e-01 2.294490e-02 0.2135086116
## 34   75 4.420434e-02 4.400526e-01 1.863992e-03 0.5138790764
## 35   76 1.906032e-04 2.075660e-05 9.546388e-06 0.9997790836
## 36   77 6.699462e-05 6.618309e-06 3.974343e-06 0.9999223906
## 37   78 2.914782e-02 9.224737e-01 1.985723e-02 0.0285212553
## 38   79 2.071764e-02 9.467702e-01 1.134417e-02 0.0211680267
## 39   80 2.172570e-02 9.440430e-01 1.254825e-02 0.0216830240
## 40   81 3.666653e-03 9.915628e-01 1.192997e-03 0.0035775245
```

2) Build tree displaying ancestral state reconstruction (Fig. 5)

Call “treeio” and “ggtree” packages for building phylogenetic tree

```
require(treeio)
require(ggtree)
```

Read in StarBEAST2 summary tree

```
beast_tree <- read.beast("StarBEAST2/summary.tree")
```

Create a new tip label variable in tree data

```
relabelling <- data.frame("label" = fortify(beast_tree)$label[1:41])
relabelling$label2 <- sub("_", "␣", relabelling$label)
relabelling$label2 <- plyr::revalue(relabelling$label2,
                                c("Amorphostigma␣armstrongi" = "Ischnura␣
                                   armstrongi",
                                  "Amorphostigma␣sp." = "Ischnura␣sp.",
                                  "Rhodischnura␣nursei" = "Ischnura␣
                                     nursei"))

beast_tree <- full_join(beast_tree, relabelling, by ="label")
```

Read in female morph state data

```
extant_state <- read.table("BayesTraits/FMorph.txt")
```

Call “dplyr” for manipulating dataframe

```
require(plyr)
require(dplyr)
```

Create dataframe for pie charts at tips (female morph state of extant species as proposed by the literature) and add it to tree data

```
extant_state <- data.frame("node"= 1:41,
                           arrange(extant_state, match(extant_state$V1,
                                                        fortify(beast_tree)$
                                                           label[1:41])))
```

Reorder states into (FMA, FMH, FPD, FPT)

```
extant_state$V2 <- revalue(as.character(extant_state$V2), c("0"="FMA", "1"
   ="FPD", "2"="FMH", "3"="FPT"))
extant_state$V2 <- ordered(extant_state$V2, levels = c("FMA", "FMH", "FPD"
   , "FPT"))
```

Join state data to concensus tree data

```
beast_tree <- full_join(beast_tree, extant_state, by ="node")
```

Create dataframe for pie charts at nodes (mean posterior estimate for each state at each node and percentage of posterior that significantly predicted the dominant state)

```
pie_labels <- data.frame(mean_state_per_node,
                         "x" = fortify(beast_tree)$x[42:81],
                         "y" = fortify(beast_tree)$y[42:81])
```

Reorder pie_label columns into (node, FMA, FMH, FPD, FPT, x, y)

```
pie_labels <- pie_labels[, c(1, 2, 4, 3, 5, 6, 7)]
```

Call “ggplot2”, “ggtree” and “ggimage” for drawing phylogeny

```
require(ggplot2)
require(ggtree)
require(ggimage)
```

Create phylogeny with female morph states as pies and confidence percentage as text labels at nodes

```
p <- ggtree(beast_tree) +
  geom_tiplab(aes(label = label2), offset = 1) +
  coord_cartesian(clip="off") +
  geom_tippoint(aes(x = x+0.7, color = V2), size = 3) +
  theme_tree(plot.margin=margin(6, 150, 6, 20), legend.position = c(0.1,
     0.9)) +
  scale_color_manual(values = c("FMA"="#56B4E9", "FMH"="#009E73", "FPD"="#
     D55E00", "FPT"="#CC79A7"),
                     labels = c("FM(A)",
                                "FM(H)",
                                "FP(D)",
                                "FP(T)"),
                     name = "Female␣State")

pies <- nodepie(pie_labels, cols=2:5, color=c("FMA"="#56B4E9", "FMH"="#009
   E73", "FPD"="#D55E00", "FPT"="#CC79A7"), alpha=0.8)

tree <- inset(p, pies, width=0.09, height=0.09)

ggsave("Fig4.pdf", tree, path = "Figures", width = 7, height = 9)
```

3) Prepare data for diagnostics

Remove any parameters that are not of further interest (i.e. anything that is not an iteration, likelihood or rate parameter)

```
attributes(run1)$names
```

```
##   [1] "Iteration"         "Lh"                "Tree.No"
##   [4] "No.Off.Parmeters" "No.Off.Zero"       "Model.string"
##   [7] "q01"               "q02"               "q03"
##  [10] "q10"               "q12"               "q13"
##  [13] "q20"               "q21"               "q23"
##  [16] "q30"               "q31"               "q32"
##  [19] "Root.P.0."         "Root.P.1."         "Root.P.2."
##  [22] "Root.P.3."         "RecNode42.P.0."    "RecNode42.P.1."
##  [25] "RecNode42.P.2."    "RecNode42.P.3."    "RecNode43.P.0."
##  [28] "RecNode43.P.1."    "RecNode43.P.2."    "RecNode43.P.3."
##  [31] "RecNode44.P.0."    "RecNode44.P.1."    "RecNode44.P.2."
##  [34] "RecNode44.P.3."    "RecNode45.P.0."    "RecNode45.P.1."
##  [37] "RecNode45.P.2."    "RecNode45.P.3."    "RecNode46.P.0."
##  [40] "RecNode46.P.1."    "RecNode46.P.2."    "RecNode46.P.3."
##  [43] "RecNode47.P.0."    "RecNode47.P.1."    "RecNode47.P.2."
##  [46] "RecNode47.P.3."    "RecNode48.P.0."    "RecNode48.P.1."
##  [49] "RecNode48.P.2."    "RecNode48.P.3."    "RecNode49.P.0."
##  [52] "RecNode49.P.1."    "RecNode49.P.2."    "RecNode49.P.3."
##  [55] "RecNode50.P.0."    "RecNode50.P.1."    "RecNode50.P.2."
##  [58] "RecNode50.P.3."    "RecNode51.P.0."    "RecNode51.P.1."
##  [61] "RecNode51.P.2."    "RecNode51.P.3."    "RecNode52.P.0."
##  [64] "RecNode52.P.1."    "RecNode52.P.2."    "RecNode52.P.3."
##  [67] "RecNode53.P.0."    "RecNode53.P.1."    "RecNode53.P.2."
##  [70] "RecNode53.P.3."    "RecNode54.P.0."    "RecNode54.P.1."
##  [73] "RecNode54.P.2."    "RecNode54.P.3."    "RecNode55.P.0."
##  [76] "RecNode55.P.1."    "RecNode55.P.2."    "RecNode55.P.3."
##  [79] "RecNode56.P.0."    "RecNode56.P.1."    "RecNode56.P.2."
##  [82] "RecNode56.P.3."    "RecNode57.P.0."    "RecNode57.P.1."
##  [85] "RecNode57.P.2."    "RecNode57.P.3."    "RecNode58.P.0."
##  [88] "RecNode58.P.1."    "RecNode58.P.2."    "RecNode58.P.3."
##  [91] "RecNode59.P.0."    "RecNode59.P.1."    "RecNode59.P.2."
##  [94] "RecNode59.P.3."    "RecNode60.P.0."    "RecNode60.P.1."
##  [97] "RecNode60.P.2."    "RecNode60.P.3."    "RecNode61.P.0."
## [100] "RecNode61.P.1."    "RecNode61.P.2."    "RecNode61.P.3."
## [103] "RecNode62.P.0."    "RecNode62.P.1."    "RecNode62.P.2."
## [106] "RecNode62.P.3."    "RecNode63.P.0."    "RecNode63.P.1."
## [109] "RecNode63.P.2."    "RecNode63.P.3."    "RecNode64.P.0."
## [112] "RecNode64.P.1."    "RecNode64.P.2."    "RecNode64.P.3."
## [115] "RecNode65.P.0."    "RecNode65.P.1."    "RecNode65.P.2."
## [118] "RecNode65.P.3."    "RecNode66.P.0."    "RecNode66.P.1."
## [121] "RecNode66.P.2."    "RecNode66.P.3."    "RecNode67.P.0."
## [124] "RecNode67.P.1."    "RecNode67.P.2."    "RecNode67.P.3."
## [127] "RecNode68.P.0."    "RecNode68.P.1."    "RecNode68.P.2."
## [130] "RecNode68.P.3."    "RecNode69.P.0."    "RecNode69.P.1."
## [133] "RecNode69.P.2."    "RecNode69.P.3."    "RecNode70.P.0."
## [136] "RecNode70.P.1."    "RecNode70.P.2."    "RecNode70.P.3."
## [139] "RecNode71.P.0."    "RecNode71.P.1."    "RecNode71.P.2."
## [142] "RecNode71.P.3."    "RecNode72.P.0."    "RecNode72.P.1."
## [145] "RecNode72.P.2."    "RecNode72.P.3."    "RecNode73.P.0."
## [148] "RecNode73.P.1."    "RecNode73.P.2."    "RecNode73.P.3."
## [151] "RecNode74.P.0."    "RecNode74.P.1."    "RecNode74.P.2."
## [154] "RecNode74.P.3."    "RecNode75.P.0."    "RecNode75.P.1."
## [157] "RecNode75.P.2."    "RecNode75.P.3."    "RecNode76.P.0."
## [160] "RecNode76.P.1."    "RecNode76.P.2."    "RecNode76.P.3."
## [163] "RecNode77.P.0."   "RecNode77.P.1."   "RecNode77.P.2."
## [166] "RecNode77.P.3."   "RecNode78.P.0."   "RecNode78.P.1."
## [169] "RecNode78.P.2."   "RecNode78.P.3."   "RecNode79.P.0."
## [172] "RecNode79.P.1."   "RecNode79.P.2."   "RecNode79.P.3."
## [175] "RecNode80.P.0."   "RecNode80.P.1."   "RecNode80.P.2."
## [178] "RecNode80.P.3."   "RecNode81.P.0."   "RecNode81.P.1."
## [181] "RecNode81.P.2."   "RecNode81.P.3."   "RJRates...Mean"
## [184] "X"
```

```r
for (i in 1:length(directories)){
  temp <- eval(parse(text=paste("run", i, "[,c(1,2,7:22)]", sep = "")))
  assign(paste("run", i, sep = ""), temp)
}
```

Make your mcmc objects and throw away the burnin

```r
for (i in 1:length(directories)){
  temp <- eval(parse(text=paste("run", i,sep = "")))
  assign (paste ("mcmc",i ,sep = ""),
          mcmc(data = temp , start = start, end = end, thin = thin))
}
```

Put them into a list

```r
mh.list <-mcmc.list()
for (i in 1:length(directories)){
      temp <- eval(parse(text=paste("mcmc", i, sep = "")))
      mh.list[[i]] <- temp
}
```

2) Diagnostics for a single run

a) Autocorrelation

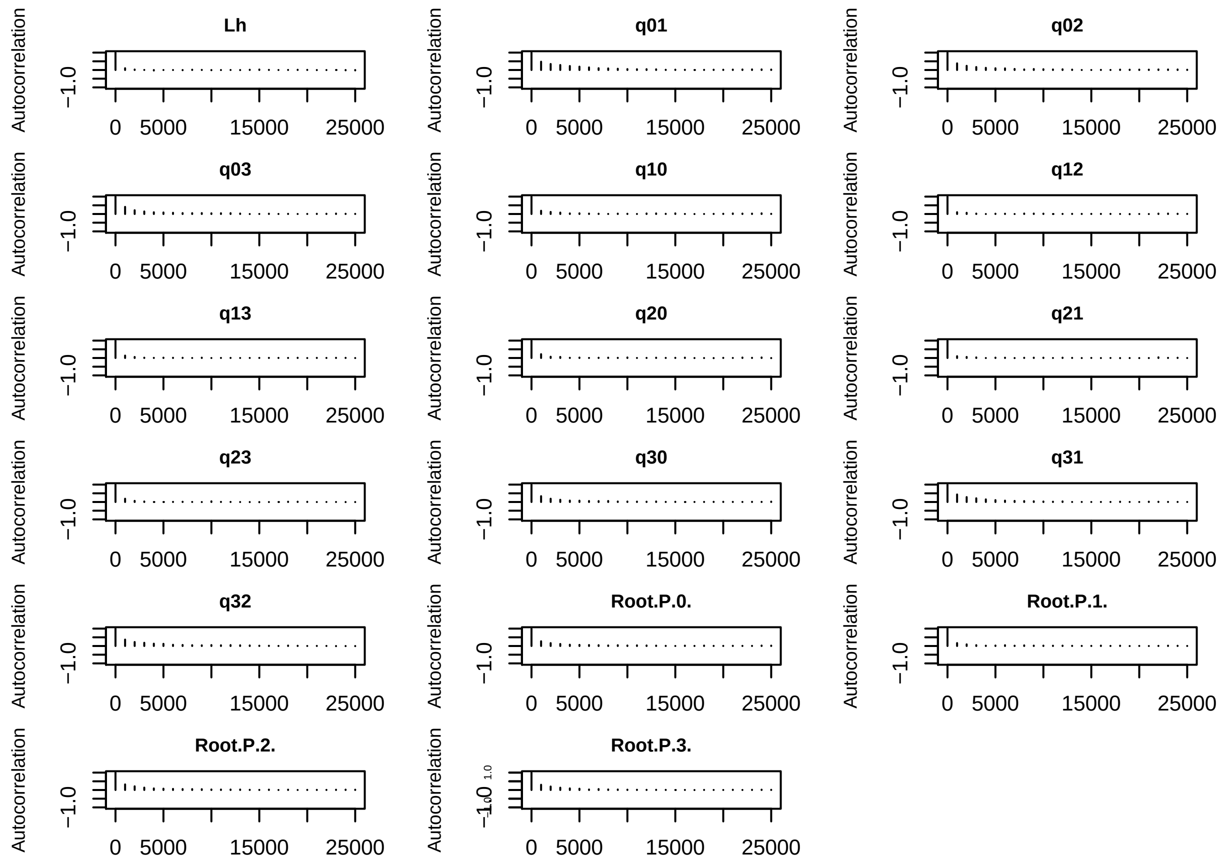
Calculate autocorrelation between draws

```r
diag(autocorr(mcmc1)[2, , ])
```

```
## Iteration        Lh       q01       q02       q03       q10       q12
        q13
## 0.9996250 0.1020494 0.4791905 0.3840774 0.4103537 0.1877256 0.1036359
   0.1402739
##       q20       q21       q23       q30       q31       q32 Root.P.0.
   Root.P.1.
## 0.2205123 0.1036719 0.1889591 0.3361016 0.4347158 0.3690565 0.2731704
   0.1685609
## Root.P.2. Root.P.3.
## 0.3224923 0.3019637
```

Present autocorrelation in plot form

```r
par(mfrow=c(6,3), mar=c(2,4,2,2)+0.1, cex.main = 0.9, cex.lab = 0.9)
autocorr.plot(mcmc1[,-1], lag.max = 25, auto.layout = F)
axis(2,cex.axis=0.5)
```



b) Effective sample size Calculate effective sample size

```
effectiveSize(mcmc1[,-1])
```

```
##         Lh        q01        q02        q03        q10        q12        q13
       q20
##  6696.540  1461.169  2050.247  2145.036  2634.353  4986.226  5561.923
   4351.389
##       q21        q23        q30        q31        q32 Root.P.0. Root.P.1.
   Root.P.2.
##  5505.977  4534.217  2454.345  2082.622  1884.090  2674.737  3951.874
   2423.264
## Root.P.3.
##  2661.313
```

3) Test for convergence of multiple runs

a) Gelman-Rubin statistic Calculate Gelman-Rubin diagnostic of convergence

```
gelman.diag(mh.list,confidence = 0.95, transform=TRUE, autoburnin=FALSE,
            multivariate=FALSE)
```

```
## Potential scale reduction factors:
##
##             Point est. Upper C.I.
## Iteration         NaN         NaN
## Lh               1.00        1.00
## q01              1.01        1.01
## q02              1.01        1.02
## q03              1.00        1.01
## q10              1.01        1.01
## q12              1.02        1.02
## q13              1.00        1.00
## q20              1.03        1.03
## q21              1.01        1.01
## q23              1.00        1.01
## q30              1.00        1.00
## q31              1.00        1.00
## q32              1.01        1.01
## Root.P.0.        1.00        1.00
## Root.P.1.        1.00        1.00
## Root.P.2.        1.00        1.00
## Root.P.3.        1.00        1.00
```

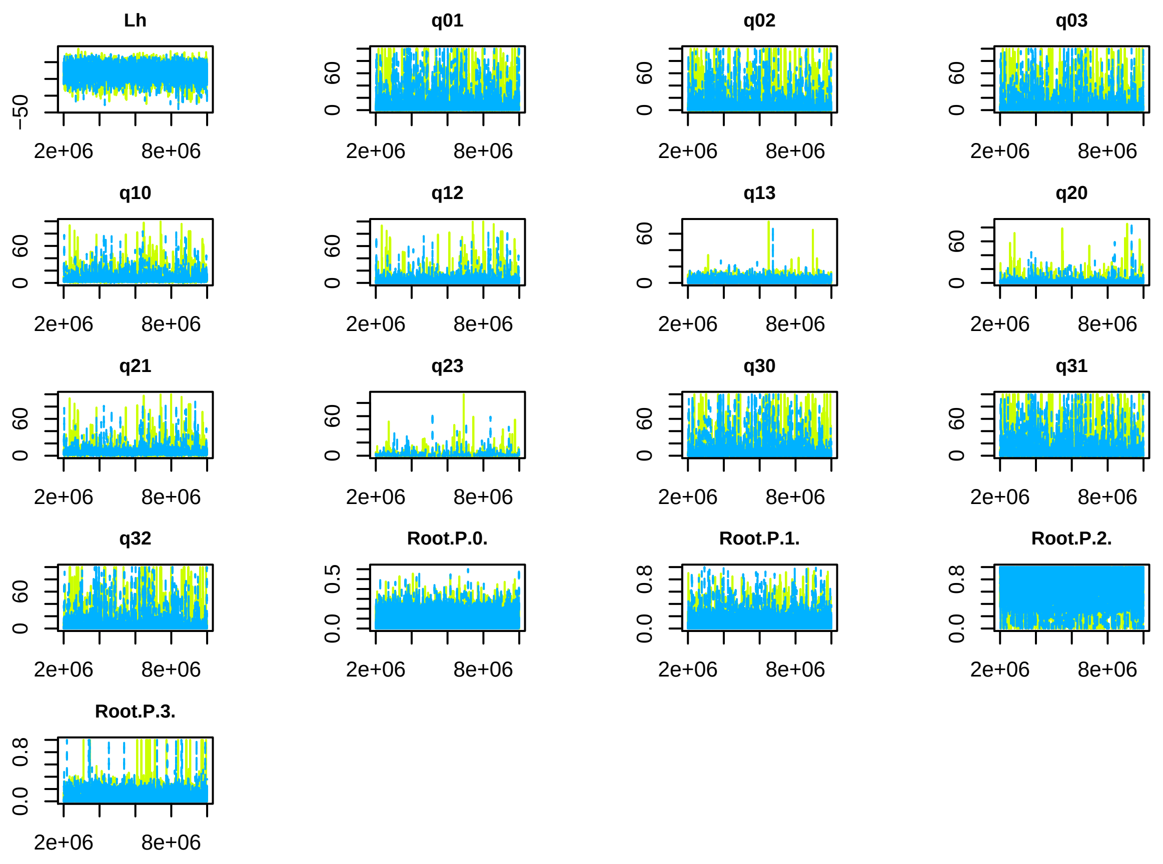
b) Plotting traces Get data into correct format for plotting traces from both runs simultaneously

```
for (i in 2:length(run1)){
  for (j in 1:length(directories)){
    temp <- eval(parse(text=paste("mcmc", j, "[,i]", sep = "")))
    assign(paste("trace_","mcmc", j, colnames(run1[i]), sep=""), temp)
  }
}

temp_list = mcmc.list()
for (i in 2:length(run1)){
  for (j in 1:length(directories)){
    temp_list[[j]] <-  eval(parse(text=paste("trace_mcmc", j , colnames(
       run1[i]),
                                              sep = "")))
    assign(paste("trace_" , colnames(run1[i]), sep = ""), temp_list)
 }
}
```

Plot traces

```
par(mfrow=c(5,4), mar = c(2,4,2,2)+0.1, cex.main = 0.9)
for (i in 2:length(run1)){
  temp <-  eval(parse(text = paste("trace_", colnames(run1[i]), sep = ""))
     )
  traceplot(temp,col = rainbow(3, start=0.2, end=0.9))
  title(main = colnames(run1[i]), ylab = "␣")
}
```



c) Plotting density curves Get data into correct format for plotting density curves of both runs and throw out burnin

```
reshaping <- function(x) {
  y <- rbind(t(x[,-1]))
  result <- setNames(as.data.frame.table(y),c("variable","run","value"))
}

for (i in 1:length(directories)){
  temp <- eval(parse(text=paste("run", i,sep = "")))
  reshape <- reshaping(temp)
  reshape$run <- as.factor(rep(i, nrow(reshape)))
  assign(paste("reshape", i, sep = ""), reshape)
}

plot_df <- rbind(reshape1,reshape2)
```

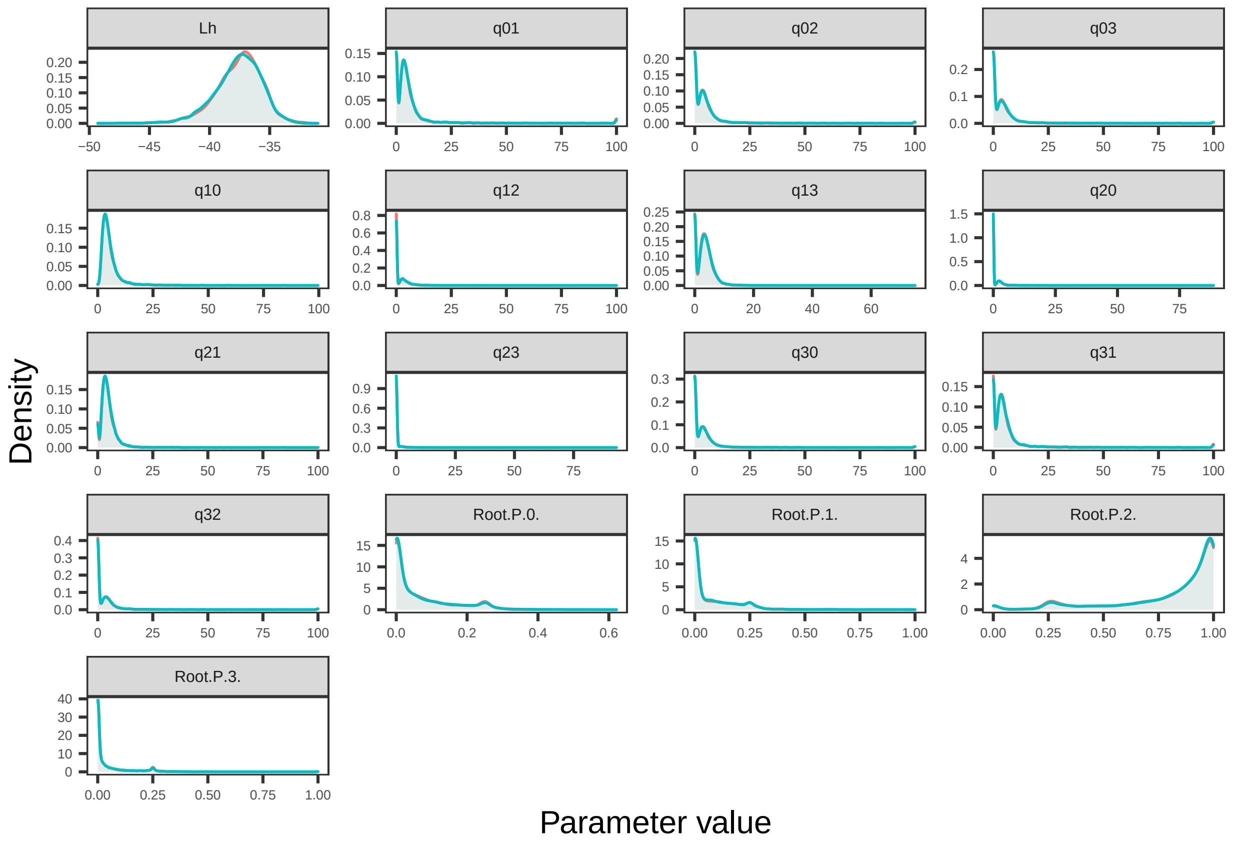
Call the “ggplot2” package for plotting and “ggforce” for paginate function

```
require(ggplot2)
require(ggforce)
```

Plot by each parameter

```
p <- ggplot(plot_df, aes(x= value, fill = run, colour = run)) +
  geom_density(alpha = 0.1) +
  theme_bw(base_size = 12)+
  theme(legend.position = "none", strip.text.x = element_text(size = 6),
        axis.text.y = element_text(size=5), axis.text.x = element_text(
           size=5),
        panel.grid.major = element_blank(), panel.grid.minor = element_
           blank())+
  labs(x="Parameter␣value", y = "Density") +
  facet_wrap(~variable, scales = "free", ncol = 4)

p
```


Lh
q01
q02
q03
q10
q12
q13
q20
q21
q23
q30
q31
q32
Root.P.0.
Root.P.1.
Root.P.2.
Root.P.3.
Density
Parameter value