Expanded ancestral state reconstruction (incl. diagnostics)

Rachel Blow

Jan 2021

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 <- ("SM_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 =
             147,
                                  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] 96.0375
```

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

```
nodes_output <- outputmc[,22:217]
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" = 51:99, 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     51 0.0076701010 0.3256016127 0.6619304197 0.0047978626
## 2     52 0.0135781822 0.6153434825 0.3656854056 0.0053929135
## 3     53 0.0047292951 0.9839735502 0.0068390461 0.0044580821
## 4     54 0.0045396320 0.9841867736 0.0068528452 0.0044207424
## 5     55 0.0017605074 0.9934166420 0.0027865391 0.0020363115
## 6     56 0.0026690000 0.9906914184 0.0040554075 0.0025841724
## 7     57 0.0051608604 0.9819029922 0.0075458569 0.0053902769
## 8     58 0.0025166071 0.9894252952 0.0034575299 0.0046005490
## 9     59 0.0015646589 0.9925634556 0.0019085337 0.0039633183
## 10    60 0.0017114441 0.9902365516 0.0018596737 0.0061923117
## 11    61 0.0021869638 0.9785060960 0.0011341896 0.0181727317
## 12    62 0.0016974746 0.9848000558 0.0017444569 0.0117579569
## 13    63 0.0011066073 0.9962827493 0.0016031249 0.0010074908
## 14    64 0.0011711647 0.9960604769 0.0016906558 0.0010776859
## 15    65 0.0007637345 0.9974192335 0.0011132204 0.0007037887
## 16    66 0.0004367944 0.9985243758 0.0006357644 0.0004030402
## 17    67 0.0055273091 0.8687032872 0.0039309351 0.1218384844
## 18    68 0.0117377076 0.4927144088 0.0071932282 0.4883546647
## 19    69 0.0002905412 0.0005635548 0.0001001885 0.9990456517
## 20    70 0.0003650741 0.0005861865 0.0001259018 0.9989227591
## 21    71 0.0014177466 0.9952096135 0.0020833965 0.0012892217
## 22    72 0.0138353796 0.3435479412 0.6373002359 0.0053164338
## 23    73 0.0052770484 0.0190206225 0.9722395165 0.0034628011
## 24    74 0.0062623433 0.1077662512 0.8817144724 0.0042569383
```

```
## 25   75 0.0012865341 0.0010650971 0.9968882209 0.0007601269
## 26   76 0.0107648814 0.2705430870 0.7134187431 0.0052732833
## 27   77 0.0396094084 0.6497731577 0.3027693296 0.0078481070
## 28   78 0.0952715667 0.8685209364 0.0243266019 0.0118808918
## 29   79 0.0028571795 0.9902744791 0.0041779235 0.0026903942
## 30   80 0.0011766945 0.9961190261 0.0016458929 0.0010583649
## 31   81 0.0047277619 0.9839237653 0.0069315189 0.0044169370
## 32   82 0.0016385195 0.9955289032 0.0017282332 0.0011043190
## 33   83 0.0041910184 0.9935376556 0.0013675731 0.0009037334
## 34   84 0.0272610349 0.9670939192 0.0034617821 0.0021832505
## 35   85 0.3339607504 0.6273326341 0.0258228482 0.0128837611
## 36   86 0.0006984778 0.9976444145 0.0010097320 0.0006473585
## 37   87 0.0160553796 0.3495044990 0.6251045650 0.0093355526
## 38   88 0.0024597309 0.9916135754 0.0036001379 0.0023265330
## 39   89 0.0009303235 0.9968566271 0.0013512975 0.0008617431
## 40   90 0.0026631348 0.0020460704 0.9936522672 0.0016385118
## 41   91 0.0061929601 0.0046307950 0.9867768079 0.0023994195
## 42   92 0.0226251857 0.0134033918 0.9599132946 0.0040581237
## 43   93 0.0638623874 0.0393031339 0.8912444158 0.0055900713
## 44   94 0.3005706623 0.0311459261 0.6534381319 0.0148452814
## 45   95 0.0066558496 0.1368925631 0.8522263529 0.0042252246
## 46   96 0.0113057256 0.3039328028 0.6776811427 0.0070803118
## 47   97 0.0037736439 0.0033504275 0.9904168351 0.0024590797
## 48   98 0.0017620291 0.0015386360 0.9955918725 0.0011074258
## 49   99 0.0020658267 0.0019206236 0.9946386711 0.0013748655
```

2) Build tree displaying ancestral state reconstruction (Fig. 6)

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

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

Read in pastis summary tree from run1

```
con_tree <- read.nexus("SM_pastis/IschnuraExpanded.nexus.mcc.tree")
```

Create a new tip label variable in tree data

```
relabelling <- data.frame("label" = fortify(con_tree)$label[1:50])
relabelling$label2 <- sub("_", "␣", relabelling$label)
relabelling$label2 <- plyr::revalue(relabelling$label2,
                                    c("Amorphostigma␣armstrongi" = "Ischnura␣
                                       armstrongi",
                                      "Amorphostigma␣sp." = "Ischnura␣sp.",
                                      "Rhodischnura␣nursei" = "Ischnura␣
                                         nursei",
                                      "Ischnura␣abyssinica" =  "Ischnura␣
                                         abyssinica␣*",
                                      "Ischnura␣sp.a" = 'Ischnura␣sp.␣"a"␣␣*'
                                         ,
                                      "Ischnura␣aralensis" = "Ischnura␣
                                         aralensis␣*",
                                      "Ischnura␣chingaza" = "Ischnura␣
                                         chingaza␣*",
```

```
                                      "Ischnura␣cyane" = "Ischnura␣cyane␣*",
                                     "Ischnura␣forcipata" = "Ischnura␣
                                        forcipata␣*",
                                     "Ischnura␣genei" = "Ischnura␣genei␣*",
                                     "Ischnura␣indivisa" = "Ischnura␣indivisa
                                        ␣*",
                                     "Ischnura␣rubilio" = "Ischnura␣rubilio␣*
                                        "))

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

```
## Warning: ‘mutate_()‘ is deprecated as of dplyr 0.7.0.
## Please use ‘mutate()‘ instead.
## See vignette('programming') for more help
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_warnings()‘ to see where this warning was
   generated.
```

Read in female morph state data

```
extant_state <- read.table("SM_BayesTraits/ExpandedFMorph.txt")
```

Call "plyr" and "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"= 1:50,
                           arrange(extant_state, match(extant_state$V1,
                                                       fortify(con_tree)$
                                                          label[1:50])))
```

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

```
con_tree <- full_join(con_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(con_tree)$x[51:99],
                         "y" = fortify(con_tree)$y[51:99])
```

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" 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(con_tree) +
  geom_tiplab(aes(label = label2), offset = 0.1) +
  coord_cartesian(clip="off") +
  geom_tippoint(aes(x = x+0.05, 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("FigS2.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."        "RecNode51.P.0."   "RecNode51.P.1."
##  [25] "RecNode51.P.2."   "RecNode51.P.3."   "RecNode52.P.0."
##  [28] "RecNode52.P.1."   "RecNode52.P.2."   "RecNode52.P.3."
##  [31] "RecNode53.P.0."   "RecNode53.P.1."   "RecNode53.P.2."
##  [34] "RecNode53.P.3."   "RecNode54.P.0."   "RecNode54.P.1."
##  [37] "RecNode54.P.2."   "RecNode54.P.3."   "RecNode55.P.0."
##  [40] "RecNode55.P.1."   "RecNode55.P.2."   "RecNode55.P.3."
##  [43] "RecNode56.P.0."   "RecNode56.P.1."   "RecNode56.P.2."
##  [46] "RecNode56.P.3."   "RecNode57.P.0."   "RecNode57.P.1."
##  [49] "RecNode57.P.2."   "RecNode57.P.3."   "RecNode58.P.0."
##  [52] "RecNode58.P.1."   "RecNode58.P.2."   "RecNode58.P.3."
##  [55] "RecNode59.P.0."   "RecNode59.P.1."   "RecNode59.P.2."
##  [58] "RecNode59.P.3."   "RecNode60.P.0."   "RecNode60.P.1."
```

```
##  [61] "RecNode60.P.2."   "RecNode60.P.3."   "RecNode61.P.0."
##  [64] "RecNode61.P.1."   "RecNode61.P.2."   "RecNode61.P.3."
##  [67] "RecNode62.P.0."   "RecNode62.P.1."   "RecNode62.P.2."
##  [70] "RecNode62.P.3."   "RecNode63.P.0."   "RecNode63.P.1."
##  [73] "RecNode63.P.2."   "RecNode63.P.3."   "RecNode64.P.0."
##  [76] "RecNode64.P.1."   "RecNode64.P.2."   "RecNode64.P.3."
##  [79] "RecNode65.P.0."   "RecNode65.P.1."   "RecNode65.P.2."
##  [82] "RecNode65.P.3."   "RecNode66.P.0."   "RecNode66.P.1."
##  [85] "RecNode66.P.2."   "RecNode66.P.3."   "RecNode67.P.0."
##  [88] "RecNode67.P.1."   "RecNode67.P.2."   "RecNode67.P.3."
##  [91] "RecNode68.P.0."   "RecNode68.P.1."   "RecNode68.P.2."
##  [94] "RecNode68.P.3."   "RecNode69.P.0."   "RecNode69.P.1."
##  [97] "RecNode69.P.2."   "RecNode69.P.3."   "RecNode70.P.0."
## [100] "RecNode70.P.1."   "RecNode70.P.2."   "RecNode70.P.3."
## [103] "RecNode71.P.0."   "RecNode71.P.1."   "RecNode71.P.2."
## [106] "RecNode71.P.3."   "RecNode72.P.0."   "RecNode72.P.1."
## [109] "RecNode72.P.2."   "RecNode72.P.3."   "RecNode73.P.0."
## [112] "RecNode73.P.1."   "RecNode73.P.2."   "RecNode73.P.3."
## [115] "RecNode74.P.0."   "RecNode74.P.1."   "RecNode74.P.2."
## [118] "RecNode74.P.3."   "RecNode75.P.0."   "RecNode75.P.1."
## [121] "RecNode75.P.2."   "RecNode75.P.3."   "RecNode76.P.0."
## [124] "RecNode76.P.1."   "RecNode76.P.2."   "RecNode76.P.3."
## [127] "RecNode77.P.0."   "RecNode77.P.1."   "RecNode77.P.2."
## [130] "RecNode77.P.3."   "RecNode78.P.0."   "RecNode78.P.1."
## [133] "RecNode78.P.2."   "RecNode78.P.3."   "RecNode79.P.0."
## [136] "RecNode79.P.1."   "RecNode79.P.2."   "RecNode79.P.3."
## [139] "RecNode80.P.0."   "RecNode80.P.1."   "RecNode80.P.2."
## [142] "RecNode80.P.3."   "RecNode81.P.0."   "RecNode81.P.1."
## [145] "RecNode81.P.2."   "RecNode81.P.3."   "RecNode82.P.0."
## [148] "RecNode82.P.1."   "RecNode82.P.2."   "RecNode82.P.3."
## [151] "RecNode83.P.0."   "RecNode83.P.1."   "RecNode83.P.2."
## [154] "RecNode83.P.3."   "RecNode84.P.0."   "RecNode84.P.1."
## [157] "RecNode84.P.2."   "RecNode84.P.3."   "RecNode85.P.0."
## [160] "RecNode85.P.1."   "RecNode85.P.2."   "RecNode85.P.3."
## [163] "RecNode86.P.0."   "RecNode86.P.1."   "RecNode86.P.2."
## [166] "RecNode86.P.3."   "RecNode87.P.0."   "RecNode87.P.1."
## [169] "RecNode87.P.2."   "RecNode87.P.3."   "RecNode88.P.0."
## [172] "RecNode88.P.1."   "RecNode88.P.2."   "RecNode88.P.3."
## [175] "RecNode89.P.0."   "RecNode89.P.1."   "RecNode89.P.2."
## [178] "RecNode89.P.3."   "RecNode90.P.0."   "RecNode90.P.1."
## [181] "RecNode90.P.2."   "RecNode90.P.3."   "RecNode91.P.0."
## [184] "RecNode91.P.1."   "RecNode91.P.2."   "RecNode91.P.3."
## [187] "RecNode92.P.0."   "RecNode92.P.1."   "RecNode92.P.2."
## [190] "RecNode92.P.3."   "RecNode93.P.0."   "RecNode93.P.1."
## [193] "RecNode93.P.2."   "RecNode93.P.3."   "RecNode94.P.0."
## [196] "RecNode94.P.1."   "RecNode94.P.2."   "RecNode94.P.3."
## [199] "RecNode95.P.0."   "RecNode95.P.1."   "RecNode95.P.2."
## [202] "RecNode95.P.3."   "RecNode96.P.0."   "RecNode96.P.1."
## [205] "RecNode96.P.2."   "RecNode96.P.3."   "RecNode97.P.0."
## [208] "RecNode97.P.1."   "RecNode97.P.2."   "RecNode97.P.3."
## [211] "RecNode98.P.0."   "RecNode98.P.1."   "RecNode98.P.2."
## [214] "RecNode98.P.3."   "RecNode99.P.0."   "RecNode99.P.1."
## [217] "RecNode99.P.2."   "RecNode99.P.3."   "RJRates...Mean"
## [220] "X"
```

```
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

```
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

```
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

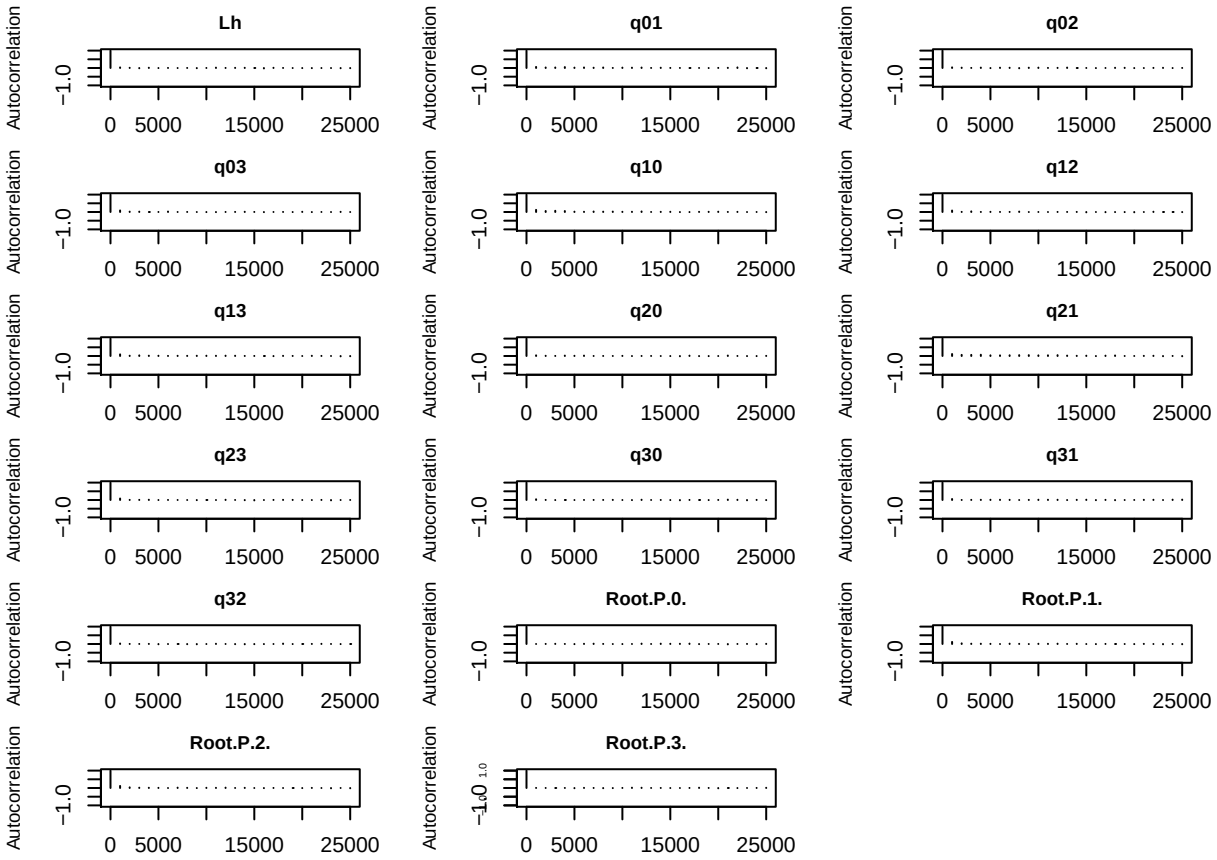
a) Autocorrelation Calculate autocorrelation between draws

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

```
##   Iteration          Lh         q01         q02         q03         q10
## 0.999625000 0.035911687 0.062667638 0.038067168 0.090123284 0.124370613
##         q12         q13         q20         q21         q23         q30
## 0.085763786 0.094406468 0.040058428 0.114400791 0.086967778 0.042871411
##         q31         q32   Root.P.0.   Root.P.1.   Root.P.2.   Root.P.3.
## 0.057581405 0.031142577 0.001423097 0.121751328 0.120304627 0.008348584
```

Present autocorrelation in plot form

```
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
##  7263.352  4907.674  6120.958  6248.476  4674.233  6227.469  6068.171
   6308.547
##         q21        q23        q30        q31        q32 Root.P.0. Root.P.1.
   Root.P.2.
##  2994.227  6719.007  7341.338  7127.968  7515.828  8000.000  5974.841
   5990.608
## Root.P.3.
##  8000.000
```

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           1
## q01                 1           1
## q02                 1           1
## q03                 1           1
## q10                 1           1
## q12                 1           1
## q13                 1           1
## q20                 1           1
## q21                 1           1
## q23                 1           1
## q30                 1           1
## q31                 1           1
## q32                 1           1
## Root.P.0.           1           1
## Root.P.1.           1           1
## Root.P.2.           1           1
## Root.P.3.           1           1
```

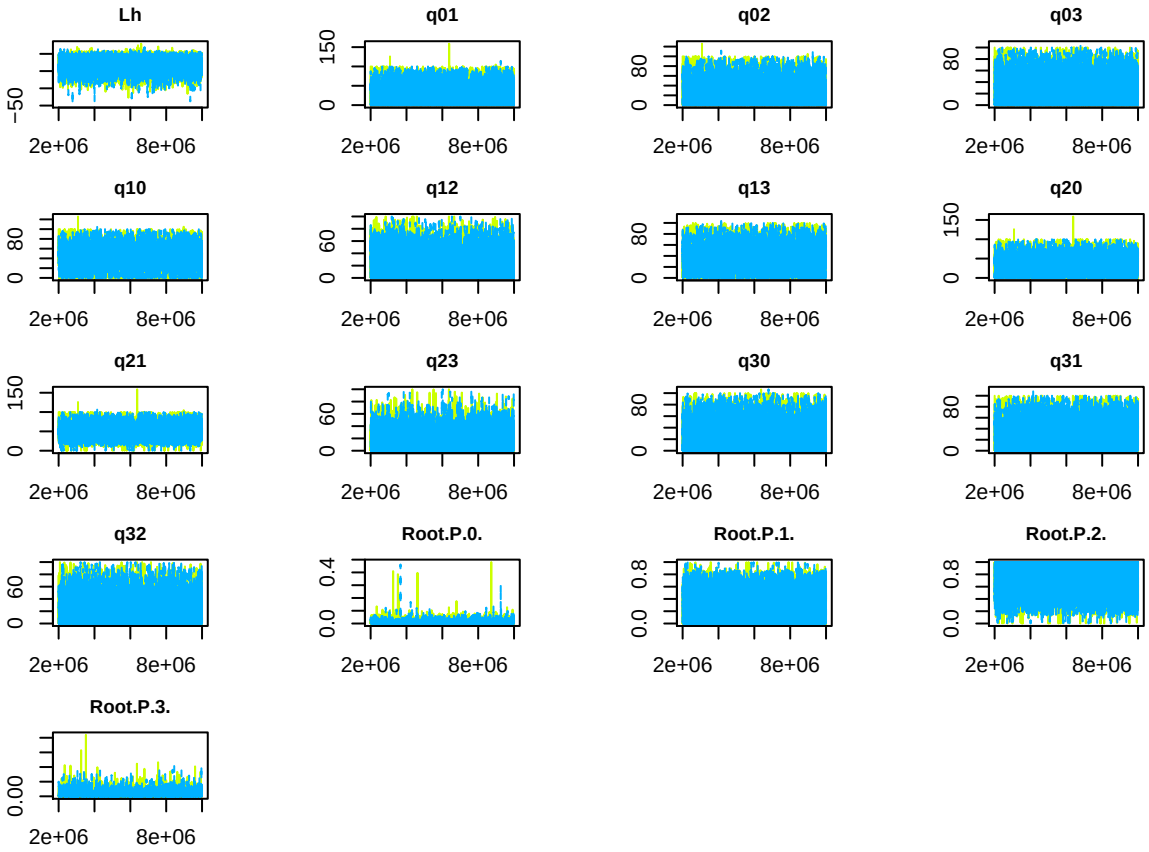
b) Plotting traces Get data into correct format for plotting traces from both runs simultaneously

```
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 "ggforce" for paginate function

```
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
```