BEAST_output_analysis_pipeline

Luiz Max Carvalho

25 January 2018

Preparation

First, let's specify the folder in which

```
folder <- "../examples/denv4/poor/"
ntrees <- 1000
```

Continuous parameters

The aim in this first section is to analyse the samples obtained for continuous parameters such as the evolutionary rate, (log) population sizes and the transition transversion rate parameter in the HKY model (\(\kappa\)).

Without further ado, let's load in the `.log` files and compute convergence diagnostics:

```
Logs <- getLogs(folder)
```

```
## Took 6.9  to load 3  log files
```

```
StepSize <- tail(Logs[[1]], 1)$state / ntrees
ProcessedLogs <- process_logs(Logs, burnin  = 10) ## using the "usual" 10% warm-up/burnin here
```

```
## Took 0.228  to process 3  log files
```

Here's a selection of parameters of interest:

```
ParametersOfInterest <- c("posterior", "prior",
                          "treeModel.rootHeight", "treeLength",
                          "skygrid.logPopSize1",
                          "CP1.alpha",
                          "CP1.kappa", "CP2.kappa",
                          "meanRate", "coefficientOfVariation",  "covariance")
```

and let's check whether the processed `.log` are correctly formed and ready for analysis

```
check_continuity(logs = ProcessedLogs, pars = ParametersOfInterest)
```

```
## all good, analysis can proceeed
```

```
## [1] TRUE
```

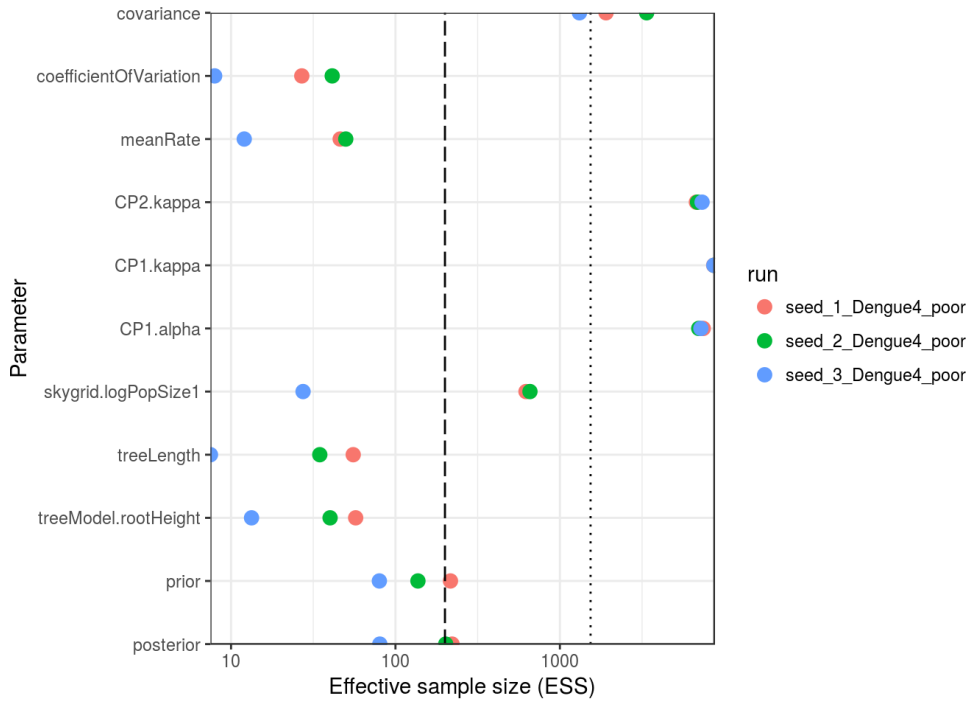
Yep. Seems so. Let's compute and look at the effective sample sizes, both univariate and multivariate:

```
( univariateESSs <- get_univariate_ESS(ProcessedLogs, pars = ParametersOfInterest) )
```

```
## Took 0.394  to compute univariate ESS for  3  log files
```

```
## $seed_1_Dengue4_poor.log
##              posterior                  prior   treeModel.rootHeight
##               220.91394              216.05858               57.27885
##             treeLength    skygrid.logPopSize1             CP1.alpha
##                55.34284              622.21883            7446.86934
##              CP1.kappa              CP2.kappa               meanRate
##             8620.01176             6763.23446               46.11481
## coefficientOfVariation             covariance
##                26.93589             1908.68995
##
## $seed_2_Dengue4_poor.log
##              posterior                  prior   treeModel.rootHeight
##               202.13019              137.00821               40.03462
##             treeLength    skygrid.logPopSize1             CP1.alpha
##                34.65403              657.72827            6992.82321
##              CP1.kappa              CP2.kappa               meanRate
##             8695.13737             6876.45783               49.86930
## coefficientOfVariation             covariance
##                41.23136             3370.53935
##
## $seed_3_Dengue4_poor.log
##              posterior                  prior   treeModel.rootHeight
##               80.332919              79.821737               13.325899
##             treeLength    skygrid.logPopSize1             CP1.alpha
##                7.496819              27.414127            7202.707626
##              CP1.kappa              CP2.kappa               meanRate
##            8640.485744            7325.688621               12.031224
## coefficientOfVariation             covariance
##                7.949814            1315.329584
```

```
ESSForPlot <- data.table::melt(
  data.table::rbindlist(
    lapply(seq_along(univariateESSs), function(i){
      x <- univariateESSs[[i]]
      res <- data.frame(
        matrix(x, nrow = 1),
        gsub(".log", "", names(univariateESSs)[i])
      )
      names(res) <- c(names(x), "run")
      return(res)
    } )
  ), id.vars = "run", variable.name = "parameter"
)
ggplot(data = ESSForPlot, aes(y = parameter, colour = run, x = value)) +
  geom_point(size = 3) +
  scale_y_discrete("Parameter", expand = c(0, 0)) +
  scale_x_log10("Effective sample size (ESS)", expand = c(0, 0)) +
  geom_vline(xintercept = 200, linetype = "longdash") +
  geom_vline(xintercept = mcmcse::minESS(p = 1, alpha = .05, eps = .1), linetype = "dotted") +
  theme_bw()
```



```
# Minimum ESS in each run
sapply(univariateESSs, summaryMin)
```

```
##                    seed_1_Dengue4_poor.log
## "coefficientOfVariation:26.9358936845359"
##                    seed_2_Dengue4_poor.log
##              "treeLength:34.6540298781855"
##                    seed_3_Dengue4_poor.log
##              "treeLength:7.49681885059039"
```

```
# Multivariate ESS (Vats et al. 2015)
mESS <- lapply(ProcessedLogs, function(x) mcmcse::multiESS(x[, ParametersOfInterest]))
unlist(mESS)
```

```
## seed_1_Dengue4_poor.log seed_2_Dengue4_poor.log seed_3_Dengue4_poor.log
##               1299.6141               1212.8027                937.0279
```

```
mcmcse::minESS(p = length(ParametersOfInterest), alpha = .05, eps = .05) ## minimum ESS needed
```

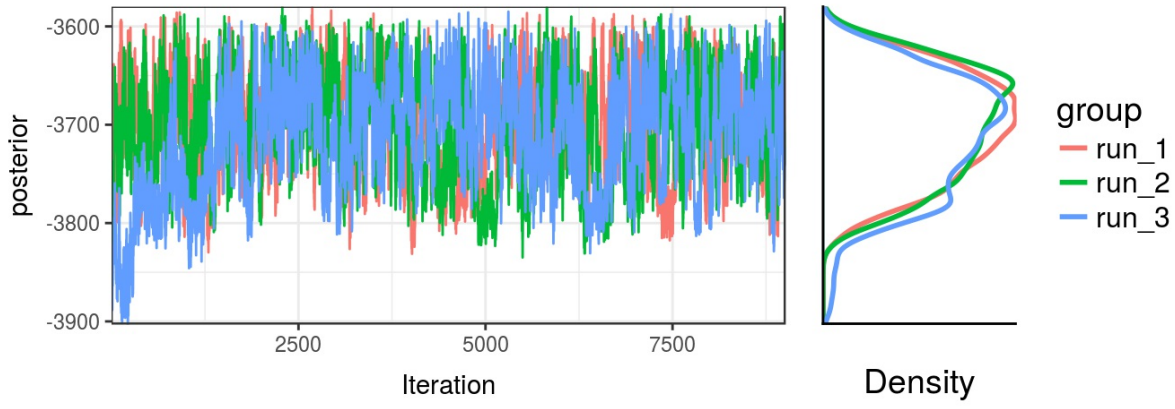
```
## minESS
##   8831
```

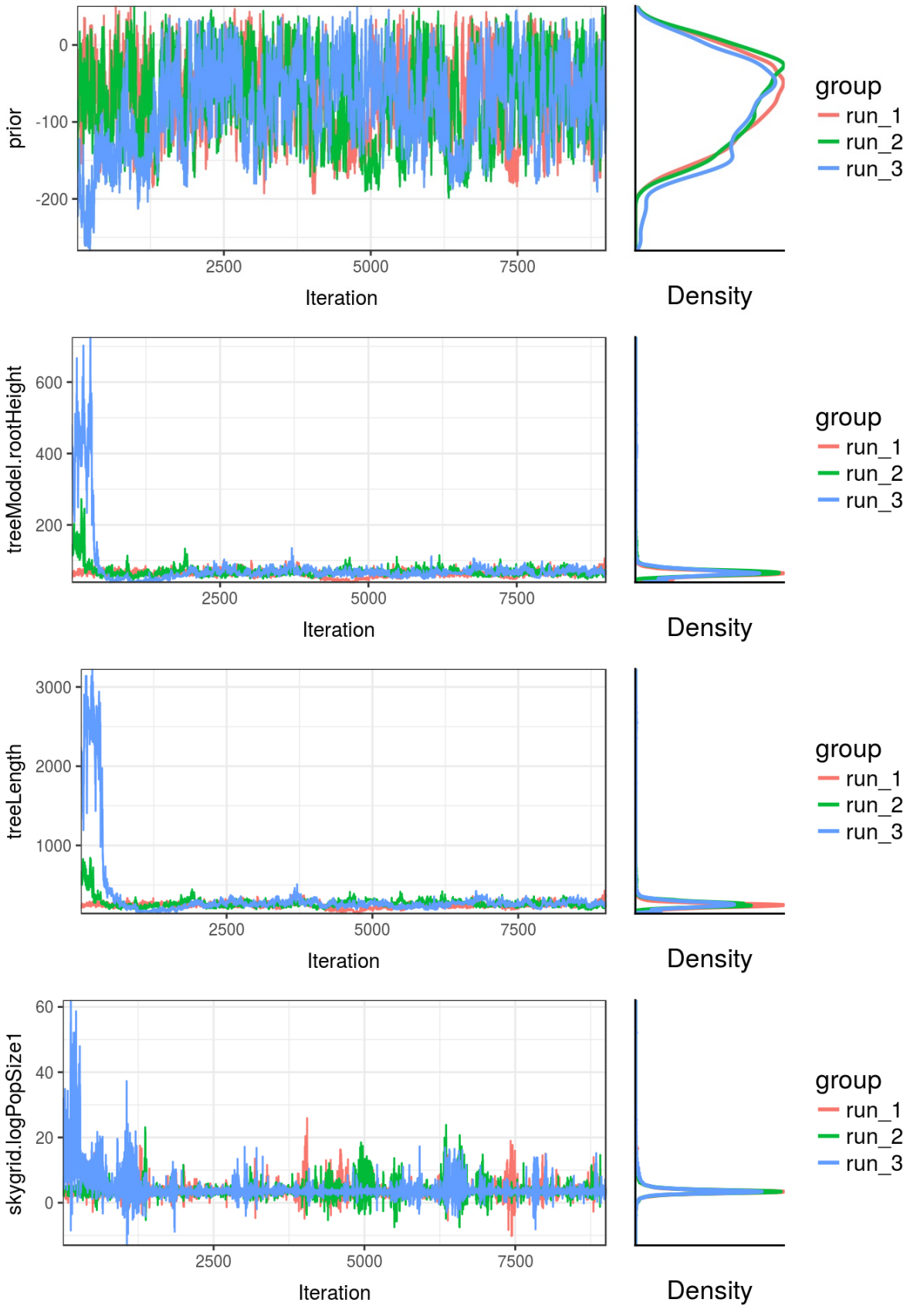
Let's look at the traces and also compute the potential scale reduction factor (PSRF) – both univariate and multivariate:

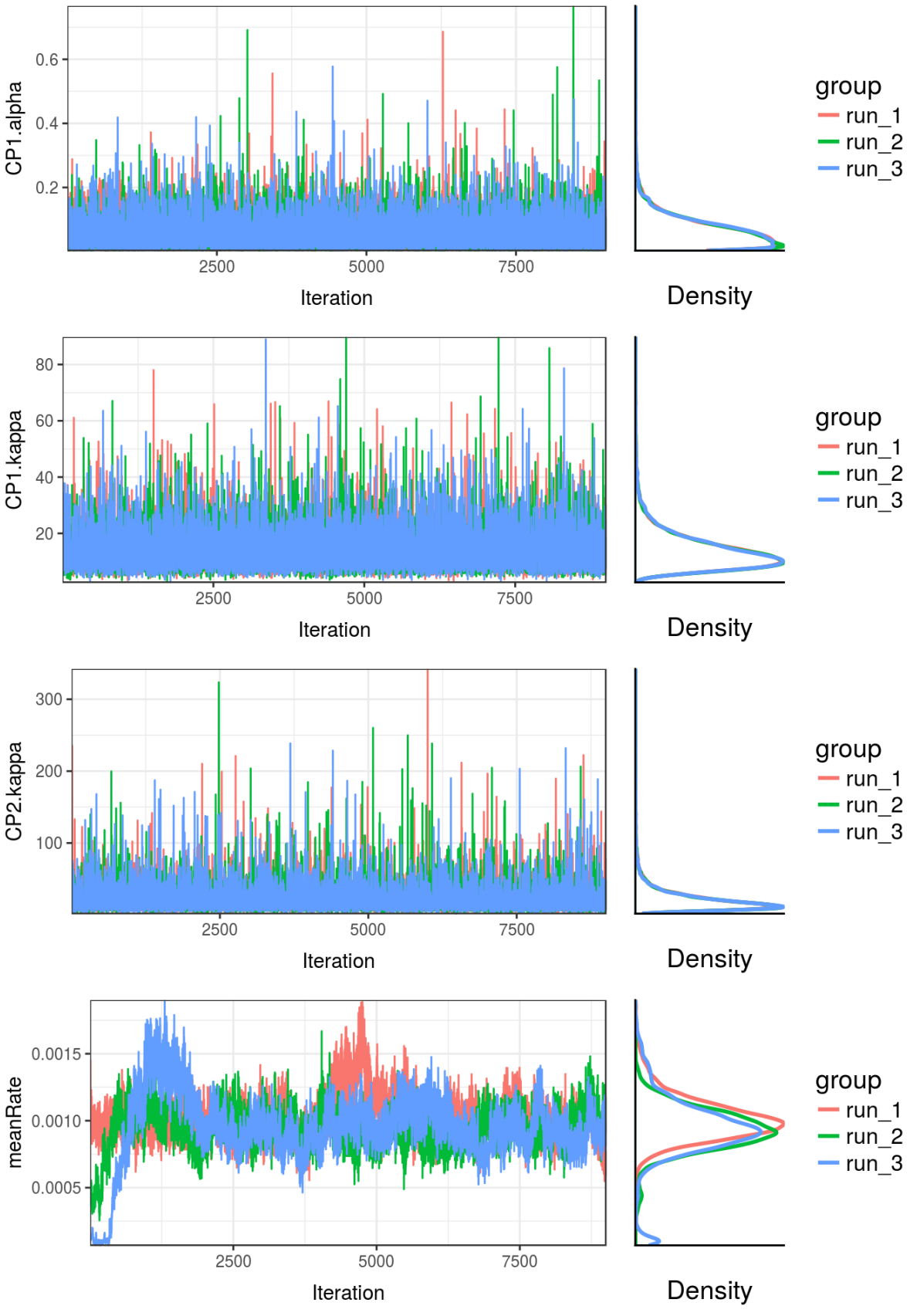
```
contPar.chain.obj <- coda::as.mcmc.list(lapply(ProcessedLogs, function(y) as.mcmc(y[, ParametersOfInterest])))
coda::gelman.diag(contPar.chain.obj)
```

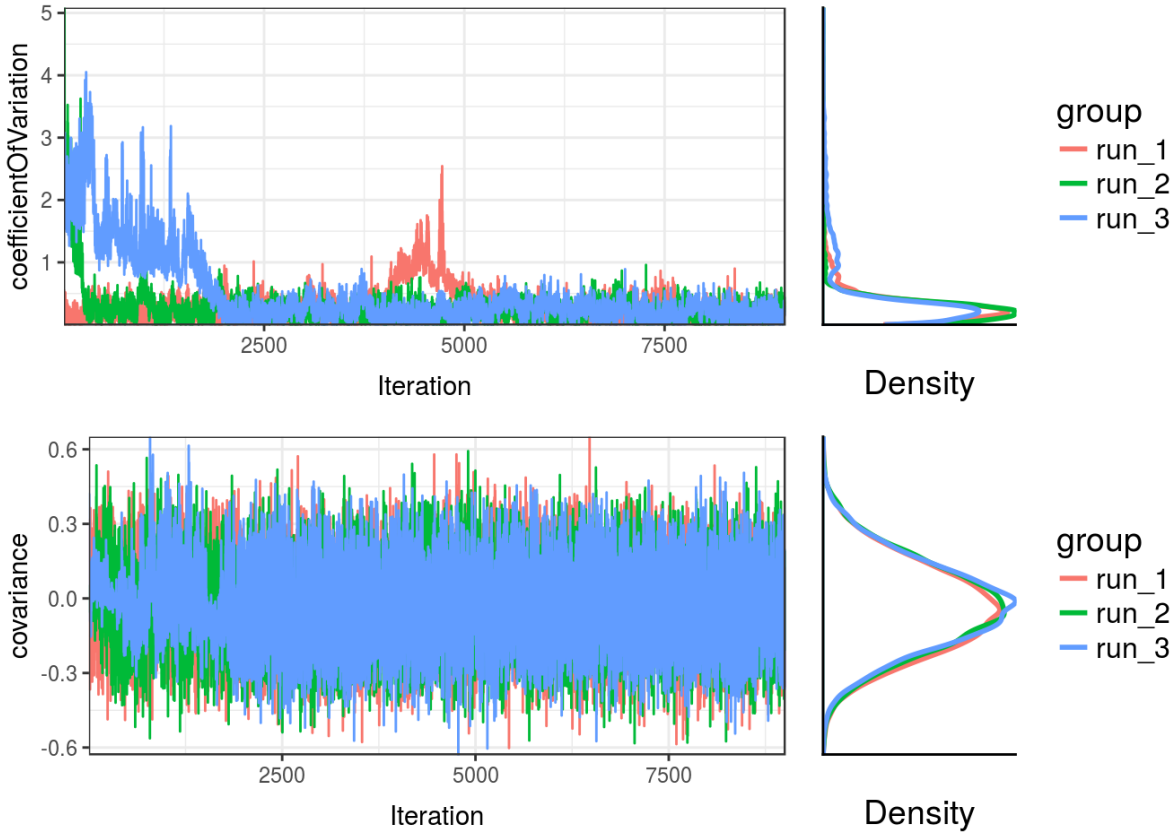
```
## Potential scale reduction factors:
##
##                         Point est. Upper C.I.
## posterior                     1.00       1.01
## prior                         1.00       1.01
## treeModel.rootHeight          1.11       1.33
## treeLength                    1.17       1.49
## skygrid.logPopSize1           1.01       1.02
## CP1.alpha                     1.00       1.00
## CP1.kappa                     1.00       1.00
## CP2.kappa                     1.00       1.00
## meanRate                      1.17       1.49
## coefficientOfVariation        1.12       1.28
## covariance                    1.00       1.00
##
## Multivariate psrf
##
## 1.13
```

```
plot_trace_new(contPar.chain.obj)
```




prior
0
-100
-200
2500
5000
7500
Iteration
Density
group
run_1
run_2
run_3
treeModel.rootHeight
600
400
200
2500
5000
7500
Iteration
Density
group
run_1
run_2
run_3
treeLength
3000
2000
1000
2500
5000
7500
Iteration
Density
group
run_1
run_2
run_3
skygrid.logPopSize1
60
40
20
0
2500
5000
7500
Iteration
Density
group
run_1
run_2
run_3


CP1.alpha
0.6
0.4
0.2
2500
5000
7500
Iteration
Density
group
run_1
run_2
run_3
CP1.kappa
80
60
40
20
2500
5000
7500
Iteration
Density
group
run_1
run_2
run_3
CP2.kappa
300
200
100
2500
5000
7500
Iteration
Density
group
run_1
run_2
run_3
meanRate
0.0015
0.0010
0.0005
2500
5000
7500
Iteration
Density
group
run_1
run_2
run_3



Exploration of Phylogenetic space In this section we will explore several diagnostics measures of convergence in phylogenetic (tree) space.

Using lower-triangle matrices of distances between trees produced by `TopologyTracer`, we will compute a multidimensional scaling (MDS) representation of phylogenetic space and then use the `plotGrovesD3()` function in the **trespace** package.

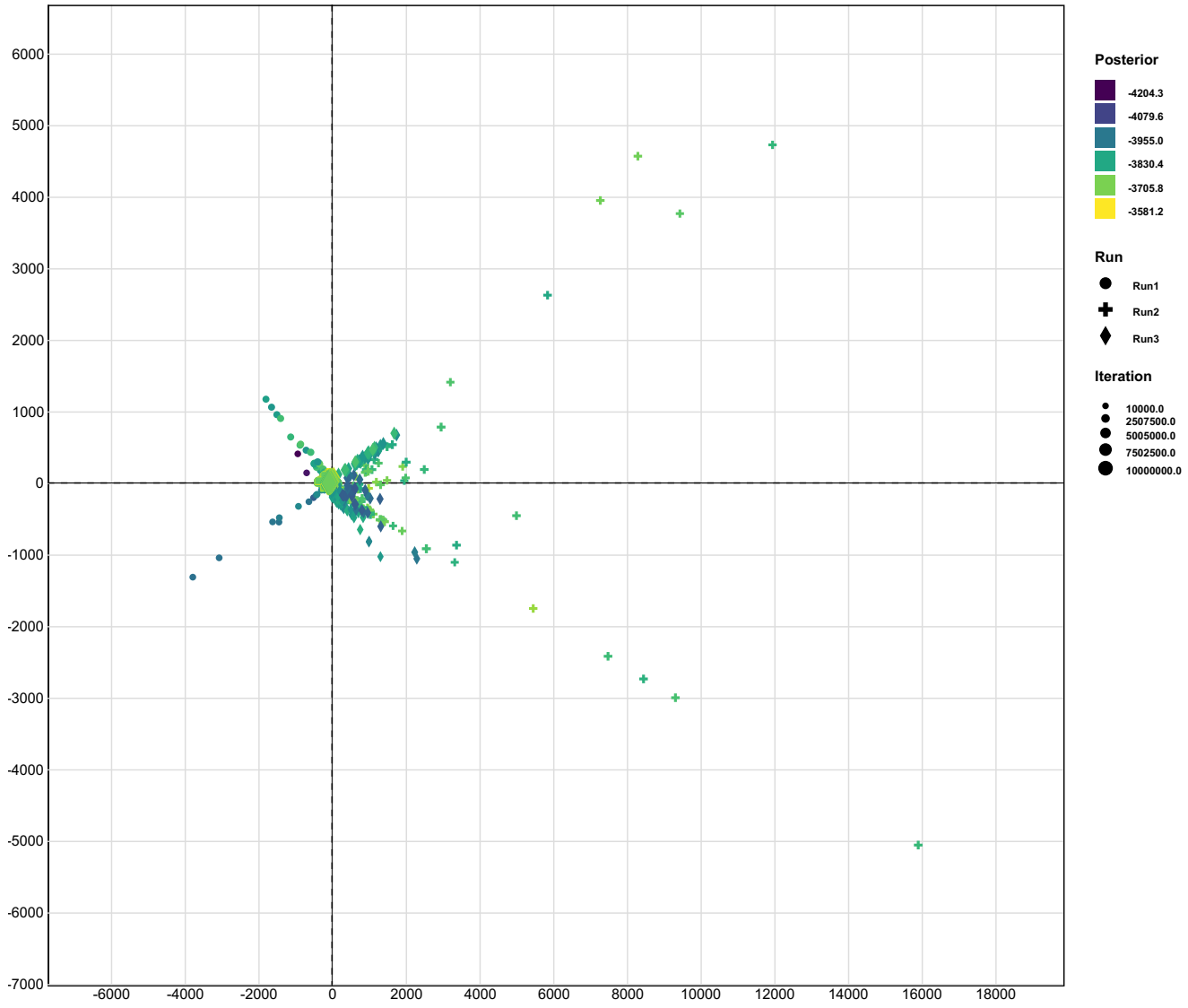
```
LT.list <- getLTs(folder) ## lower triangle matrices
```

```
## Loading required package: parallel
```

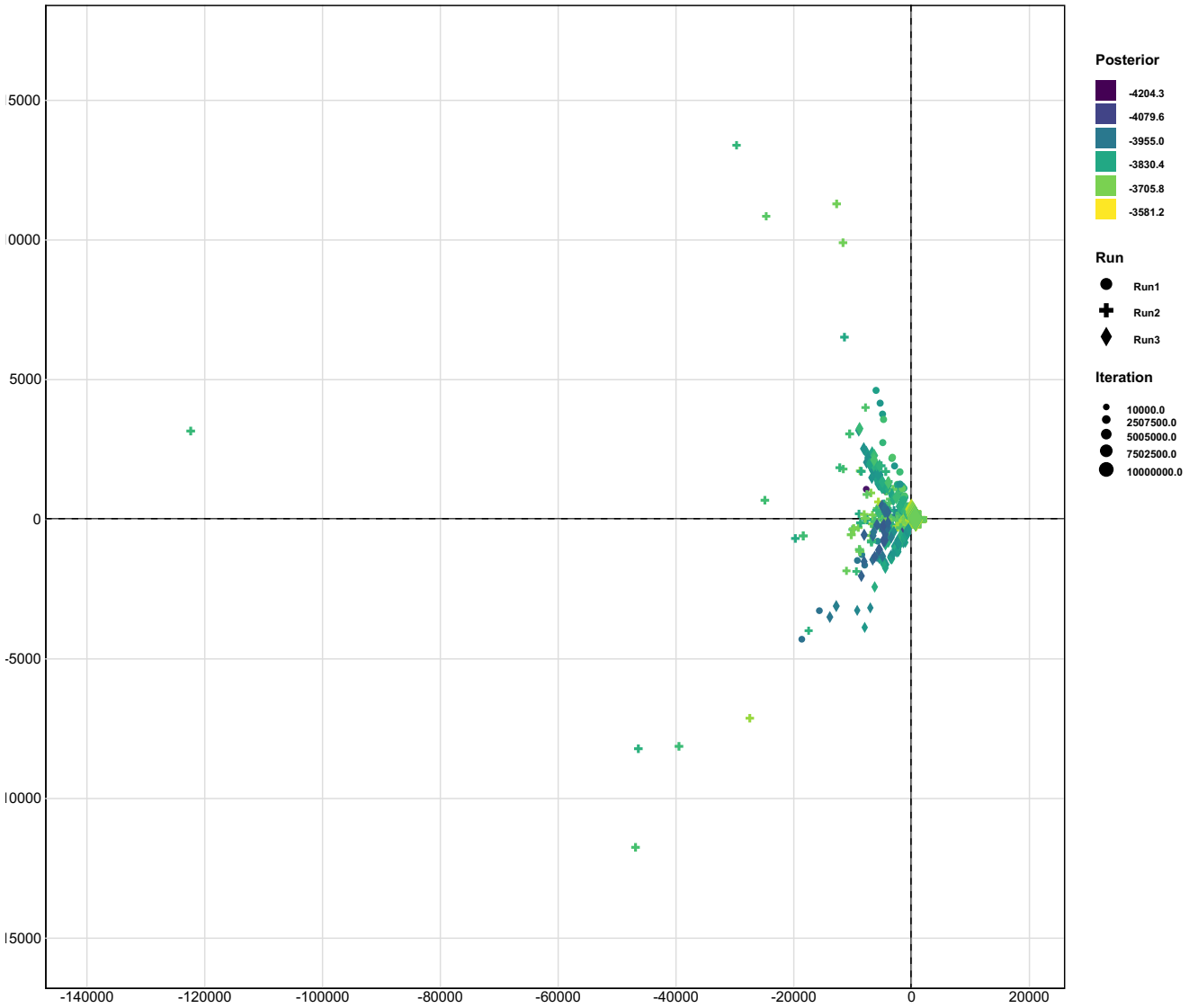
```
Full.list <- lapply(LT.list, make_full)
names(Full.list) <- gsub(".csv", "", names(Full.list))

MDS <- lapply(Full.list, getMDS, step_size = StepSize)

complete.MDS <- lapply(seq_along(MDS), function(i) {
    stem <- gsub(".csv", "", paste(strsplit(names(MDS)[i], "_")[[1]][-1], collapse = "_"))
    j <- grep(stem, names(Logs))
    TheLog <- Logs[[j]][match(seq(0, tail(Logs[[j]], 1)$state, by = tail(Logs[[j]], 1)$state / ntrees), Logs[[j]]
$state), ]
    return(
      data.frame(MDS[[i]],
                 likelihood = TheLog$likelihood,
                 posterior = TheLog$posterior)
    )
  } )
  names(complete.MDS) <- names(MDS)
## Let's grab just the distance matrices from the Kendall-Coljin metric with lambda = 1/2
MDS.Kc.half <- complete.MDS[grep("KChalf", names(complete.MDS))]
plotMDS.list(MDS.Kc.half, exclude = TRUE)
```




```
## Same now for the Steel-Penny metric
MDS.SP <- complete.MDS[grep("SP", names(complete.MDS))]
plotMDS.list(MDS.SP, exclude = TRUE)
```



Now let's look at clade frequencies

```
Clade_info <- get_clade_data(folder,  step_size = StepSize)
Clade_map_summaries <- lapply(Clade_info, function(x) summarise.clademap(x$map))
Clade.uni.ESS <- lapply(Clade_info, function(x) apply(x$map, 2, coda::effectiveSize))
lapply(Clade.uni.ESS,  function(y) mean(y <= 0)) ## proportion of clades stuck
```

```
## $seed_1_Dengue4_poor
## [1] 0.2291667
##
## $seed_2_Dengue4_poor
## [1] 0.247191
##
## $seed_3_Dengue4_poor
## [1] 0.2268041
```

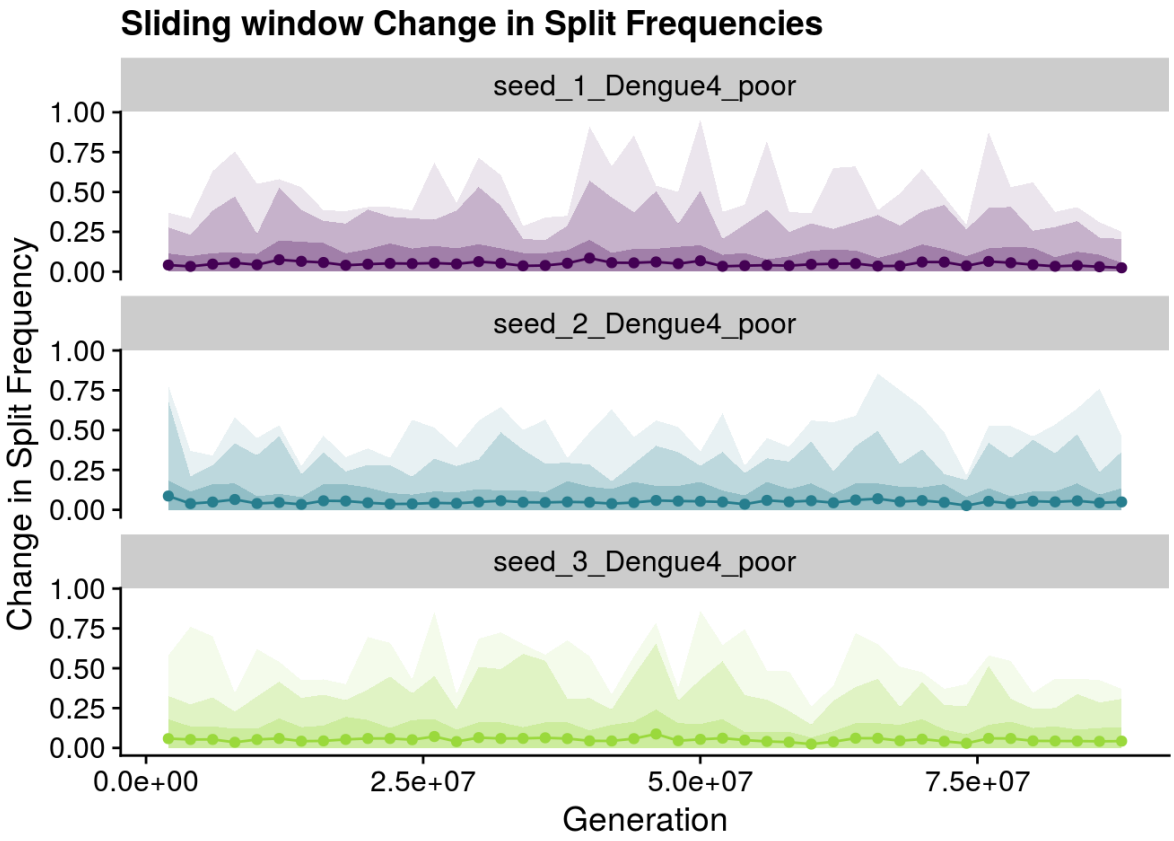
```
lapply(Clade.uni.ESS,  function(y) mean(y[y > 0])) ## mean uESS (given not stuck)
```

```
## $seed_1_Dengue4_poor
## [1] 874.0405
##
## $seed_2_Dengue4_poor
## [1] 917.3195
##
## $seed_3_Dengue4_poor
## [1] 837.3541
```

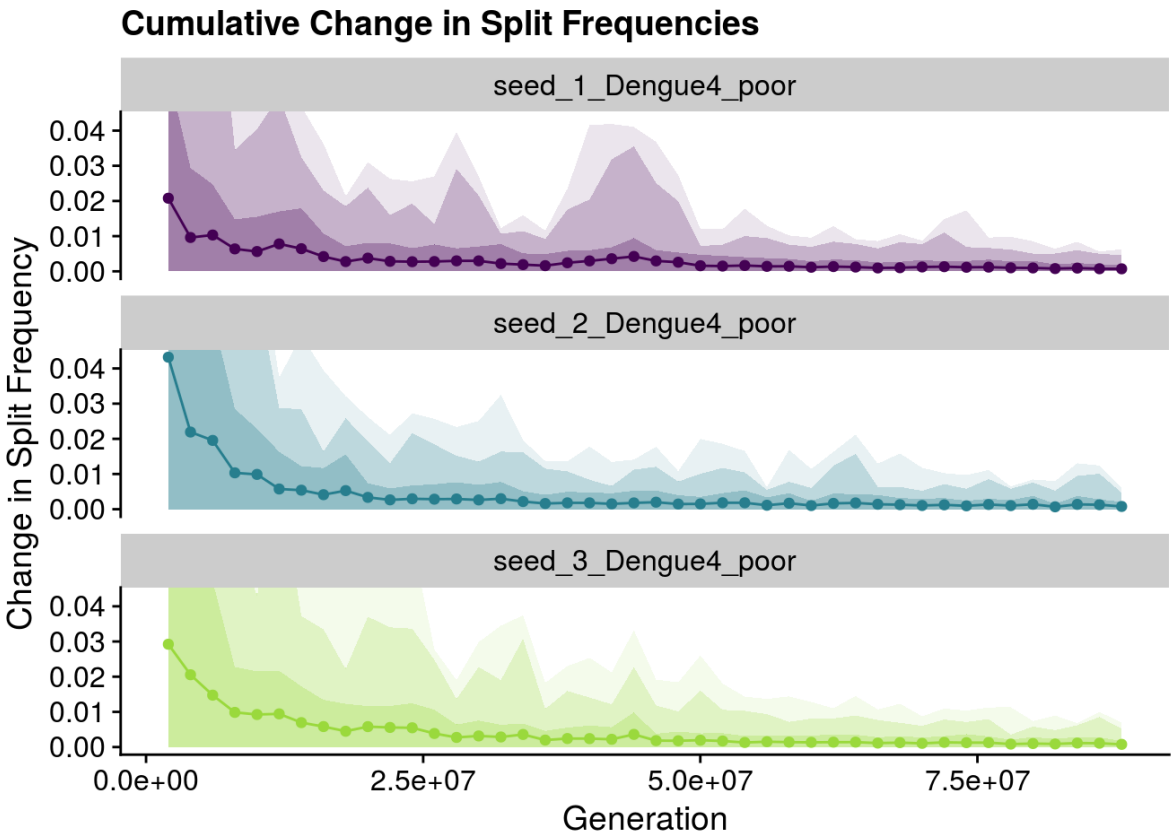
```
lapply(lapply(Clade_map_summaries, function(s) s$transition.nos/s$maximum), mean) ## mean clade switching score
```

```
## $seed_1_Dengue4_poor
## [1] 0.169045
##
## $seed_2_Dengue4_poor
## [1] 0.1837223
##
## $seed_3_Dengue4_poor
## [1] 0.1606856
```

```
SlidingWCladeInfo <- lapply(Clade_info, make_fake_slidefreq, window_size = 200, step_size = StepSize)
makeplot.acsf.sliding_mod(slide.freq.list = SlidingWCladeInfo,
                          facet = TRUE)
```




```
CumulativeCladeInfo <- lapply(Clade_info, make_fake_cumfreq, window_size = 200, step_size = StepSize)
makeplot.acsf.cumulative_mod(cumulative.freq.list = CumulativeCladeInfo,
                             facet = TRUE)
```



Now let's look at clade standard deviations. Here I will be stringent and consider the maximum average standard deviation for each run. A good rule of thumb is to consider convergence is this quantity is below \(0.05\).

```
lapply(CumulativeCladeInfo, function(x) max(rwty:::get.acsf(x)$max))
```

```
## $seed_1_Dengue4_poor
## [1] 0.185
##
## $seed_2_Dengue4_poor
## [1] 0.39
##
## $seed_3_Dengue4_poor
## [1] 0.29
```

```
  topological.approx.ess_mod(mat = Full.list)
```

```
## [1] "Calculating approximate ESS with sampling intervals from 1 to 100"
```

```
##    operator approx.ess                      chain
## 1         =  141.11897     BC_seed_1_Dengue4_poor
## 2         =  158.12057     BC_seed_2_Dengue4_poor
## 3         =   21.97844     BC_seed_3_Dengue4_poor
## 4         = 1001.00000     CD_seed_1_Dengue4_poor
## 5         =  424.53399     CD_seed_2_Dengue4_poor
## 6         =   63.00512     CD_seed_3_Dengue4_poor
## 7         =   34.87053    KC0_seed_1_Dengue4_poor
## 8         =   44.95587    KC0_seed_2_Dengue4_poor
## 9         =   39.73087    KC0_seed_3_Dengue4_poor
## 10        =  171.91435    KC1_seed_1_Dengue4_poor
## 11        = 1001.00000    KC1_seed_2_Dengue4_poor
## 12        =   21.51045    KC1_seed_3_Dengue4_poor
## 13        =  185.18373 KChalf_seed_1_Dengue4_poor
## 14        = 1001.00000 KChalf_seed_2_Dengue4_poor
## 15        =   20.80479 KChalf_seed_3_Dengue4_poor
## 16        =   63.58256     RF_seed_1_Dengue4_poor
## 17        =  107.52807     RF_seed_2_Dengue4_poor
## 18        =   39.10279     RF_seed_3_Dengue4_poor
## 19        =  159.75779     SP_seed_1_Dengue4_poor
## 20        = 1001.00000     SP_seed_2_Dengue4_poor
## 21        <   13.15303     SP_seed_3_Dengue4_poor
```

Conclusion

Runs did **not** converge. From the traceplots it is clear that the mixing was poor and we obtained very low ESSs (uni and multivariate), PSRFs above 1.1. etc. MDS clearly shows runs did not explore the same space and in addition there are many low posterior trees.

Extra: bash scripts and data processing

Let's see what the contents look like – you'll need to have a similar folder structure and file naming:

```
system(paste("ls -sh", folder), intern = TRUE)
```

```
##  [1] "total 281M"
##  [2] " 9.4M BC_seed_1_Dengue4_poor.csv"
##  [3] " 9.4M BC_seed_2_Dengue4_poor.csv"
##  [4] " 9.3M BC_seed_3_Dengue4_poor.csv"
##  [5] " 9.3M CD_seed_1_Dengue4_poor.csv"
##  [6] " 9.4M CD_seed_2_Dengue4_poor.csv"
##  [7] " 9.4M CD_seed_3_Dengue4_poor.csv"
##  [8] " 1.7M clade matrix_seed_1_Dengue4_poor.cmap"
##  [9] " 1.6M clade matrix_seed_2_Dengue4_poor.cmap"
## [10] " 1.8M clade matrix_seed_3_Dengue4_poor.cmap"
## [11] "  12K cladetable_seed_1_Dengue4_poor.txt"
## [12] "  12K cladetable_seed_2_Dengue4_poor.txt"
## [13] "  12K cladetable_seed_3_Dengue4_poor.txt"
## [14] "  48K Dengue4_poor.xml"
## [15] " 8.0K full"
## [16] " 8.0K get_cmaps.sh"
## [17] " 8.0K get_distance_matrices_full.sh"
## [18] " 8.0K get_distance_matrices.sh"
## [19] " 8.0K get_equally_spaced_subsamples.sh"
## [20] " 8.9M KC0_seed_1_Dengue4_poor.csv"
## [21] " 8.9M KC0_seed_2_Dengue4_poor.csv"
## [22] " 9.0M KC0_seed_3_Dengue4_poor.csv"
## [23] " 9.3M KC1_seed_1_Dengue4_poor.csv"
## [24] " 9.3M KC1_seed_2_Dengue4_poor.csv"
## [25] " 9.3M KC1_seed_3_Dengue4_poor.csv"
## [26] " 9.3M KChalf_seed_1_Dengue4_poor.csv"
## [27] " 9.3M KChalf_seed_2_Dengue4_poor.csv"
## [28] " 9.3M KChalf_seed_3_Dengue4_poor.csv"
## [29] "1000K nohup_Dengue4_poor_1"
## [30] "1000K nohup_Dengue4_poor_2"
## [31] "1000K nohup_Dengue4_poor_3"
## [32] " 2.5M RF_seed_1_Dengue4_poor.csv"
## [33] " 2.5M RF_seed_2_Dengue4_poor.csv"
## [34] " 2.5M RF_seed_3_Dengue4_poor.csv"
## [35] " 8.0K run_beast_gnuParallel_2.0.sh"
## [36] "  16M seed_1_Dengue4_poor.log"
## [37] " 8.0K seed_1_Dengue4_poor.ops"
## [38] " 1.6M seed_1_Dengue4_poor.strees"
## [39] "  16M seed_1_Dengue4_poor.trees"
## [40] "  16M seed_2_Dengue4_poor.log"
## [41] " 8.0K seed_2_Dengue4_poor.ops"
## [42] " 1.6M seed_2_Dengue4_poor.strees"
## [43] "  16M seed_2_Dengue4_poor.trees"
## [44] "  16M seed_3_Dengue4_poor.log"
## [45] " 8.0K seed_3_Dengue4_poor.ops"
## [46] " 1.6M seed_3_Dengue4_poor.strees"
## [47] "  16M seed_3_Dengue4_poor.trees"
## [48] " 9.4M SP_seed_1_Dengue4_poor.csv"
## [49] " 9.3M SP_seed_2_Dengue4_poor.csv"
## [50] " 9.3M SP_seed_3_Dengue4_poor.csv"
```

The `.sh` files you see are used to process the `.trees` files so they can be further analysed. Most of the code featured in this section is just very simple `bash` code using the classes in BEAST to do the heavy lifting. First, let's look at the code for downsampling the `.trees` files, `get_equally_spaced_subsamples.sh`:

```
#java -Xmx4096m -cp /path/to/beast-mcmc/build/dist/beast.jar dr.app.tools.LogCombiner > /usr/bin/logcombiner
for file in *.trees
do
stem=$(basename $file .trees)
logcombiner -trees -resample 100000 -renumber $file $stem.rtrees
echo "$file is done"
done
```

Be sure to choose your `resample` argument so as to obtain ~1000 trees. For larger trees (with, say, >500 taxa) you might want to lower this to

200 trees or so, hence increase `resample`. Now we will use `TopologyTracer` to get (a) the distance of each tree to the first tree in the chain (by default, you can change the focal tree with `-tree`) and (b) a lower-triangle matrix of tree distances. We will employ the Robinson-Foulds, Steel-Penny (aka path distance) and Kendall-Coljin. The idea of computing the ESS of the distance to the focal tree was developed by Lanfear et al. (2016), who call it "pseudo-ESS". The code for `get_distance_logs.sh` is:

```
# java -Xmx4096m -cp /path/to/beast-mcmc/build/dist/beast.jar dr.app.tools.TopologyTracer > /usr/bin/treemetrics
for file in *.trees
do
stem=$(basename $file .trees)
treemetrics $file $stem.tmlog
done
```

And here are the contents of `get_distance_matrices.sh`:

```
# java -Xmx4096m -cp /home/max/beast-myfork/build/dist/beast.jar dr.app.tools.TopologyTracer > /usr/bin/treemetri
cs
do_distmat() {
  file=$1
  stem=$(basename $file .trees)
echo "Processing $file \n"
treemetrics -burninTrees 0 -pairwise -metric kc -lambda 0 $file KC0_$stem.csv
treemetrics -burninTrees 0 -pairwise -metric kc -lambda 0.5 $file KChalf_$stem.csv
treemetrics -burninTrees 0 -pairwise -metric kc -lambda 1 $file KC1_$stem.csv
treemetrics -burninTrees 0 -pairwise -metric sp  $file SP_$stem.csv
rm $stem.aug
}
export -f do_distmat
parallel --nice 10 --max-procs 10 do_distmat ::: *.trees
```