phyloConverge Walkthrough

Elysia Saputra

This walkthrough contains instructions for using *phyloConverge* to perform convergence analysis on genomic elements. Before following the walkthrough, please make sure that *phyloConverge* and its dependencies have been successfully installed.

```
library(phyloConverge)
library(rphast)
library(RERconverge)
pcpath = find.package('phyloConverge')
```

Data Input and Formatting

phyloConverge requires the following input data:

- A neutral tree model describing neutral substitution rates. This tree model should be a model file (.mod) that can be estimated from sequence alignment, for example using the *phyloFit* function in the PHAST package or rphast. Neutral tree models are usually estimated from sites that are expected to evolve neutrally, such as fourfold-degenerate sites. Please refer to tutorials for phyloFit for examples of how to generate neutral tree models.
- The names of the foreground species with the convergent phenotype. Foreground species can be extant (tip) species or ancestral.
- Multiple sequence alignment to be analyzed. Alignment files should either be in the MAF or FASTA format.

Analysis Walkthrough

Loading input data

To start the analysis, we need to provide *phyloConverge* with input data. The **neutral tree model** and the **alignment** can be loaded using the `read.tm` and `read.msa` functions in the *rphast* package, respectively. The `read.msa` function also accepts alignments in the MAF format. Species names in the tree must match species names in the alignment. For the purposes of this vignette, we will use an alignment dataset previously published by Roscito et al. (Nature Communications 2018), with *mm10* as the reference species. The conserved non-coding element (CNE) *CNE031689* is used for illustration.

```
# Loading neutral tree model
neutral_tree_file = 'eyeStudy.tree.mod'
neutralMod_path = paste0(pcpath, '/data/', neutral_tree_file)
neutralMod = read.tm(neutralMod_path)
```

```
# Loading alignment
alnfile = 'CNE031689.fa'
alnpath = paste0(pcpath, '/data/', alnfile)
msa = read.msa(alnpath)
```

Foreground species names are supplied in the form of a character vector. In the example below, 6 extant species and 1 ancestral species are specified as foregrounds. Currently, *phyloConverge* is not yet implemented to account for trait loss events, meaning that the daughter species of an ancestral foreground species must also be specified as foregrounds.

```
foregrounds = c('nanGal1', 'chrAsi1', 'conCri1', 'hetGla2', 'mm10', 'rn5', 'mm10-rn5')
```

If the phenotype input is in the form of a binary tree, or if the correct nomenclature of the ancestral species is not known, the function `getForegroundsFromTree` can be used to extract the names of the foreground species while ensuring that the ancestral nodes are named correctly.

```
foreground_tree_file = 'foreground_tree.RDS'
foreground_tree = readRDS(paste0(pcpath, '/data/', foreground_tree_file))

# Extracting foreground names from binary input tree
foreground_names = getForegroundsFromTree(foreground_tree)
print(foreground_names)
#> [1] "mm10"     "rn5"      "nanGal1"  "hetGla2"  "conCri1"  "chrAsi1"  "mm10-rn5"
```

Performing phenotype permulation

A key operation employed by *phyloConverge* to make predictions of phenotypic association with high specificity is an empirical bias correction strategy called permulation, a portmanteau of 'permutation' and 'simulation'. Permulation is a phylogeny-aware 'trait permutation' strategy in which Brownian motion phylogenetic simulations are performed to generate null phenotypes that match the true observed phenotype in terms of the number of foreground species and the phylogenetic dependence among the foreground species. The null phenotypes are then used to compute empirical p-values that quantify the significance of the convergent rate shifts measured.

Before running `phyloConverge`, we need to generate the **permulated foreground species**, which can be done using the `getPermulatedPhenotypes` function. `getPermulatedPhenotypes` requires the following input arguments:

- `foregrounds`: A character vector containing the tip (and ancestral) foreground species
- `neutraltree`: A tree object containing the tree topology and branch lengths estimated from the neutral tree model, which is contained in the neutral tree model (.mod) file
- `num_perms`: The number of permulations. Note that the number of permulations used will affect the resolution of empirical p-values obtained (e.g., 500 permulations results in p-value resolution of 0.002, etc.). While computation time increases with increasing number of permulations, we recommend 500-1000 permulations.
- `root_species`: The species to root the trees on.
- `output_mod`: set as "names" to output null species character vectors, or "trees" to output binary phenotype trees.

The code below shows an example for generating 5 permulated phenotypes. As we can see, the null phenotypes that are generated also consists of 6 tip species and 1 ancestral species, with a covariance structure that was computed from the observed data.

```
num_perms = 5
root_species = "mm10"

neutral_tree = read.tree(text=neutralMod$tree)

permulated_foregrounds = getPermulatedPhenotypes(foregrounds, neutral_tree, num_perms,
                         root_species, output_mod="names")
print(permulated_foregrounds)
#> [[1]]
#> [1] "speTri2"          "hetGla2"          "cavPor3"          "ochPri3"
#> [5] "otoGar3"          "xenTro7"          "hetGla2-cavPor3"
#>
#> [[2]]
#> [1] "mm10"             "nanGal1"          "speTri2"          "hetGla2"
#> [5] "cavPor3"          "macFas5"          "hetGla2-cavPor3"
#>
#> [[3]]
#> [1] "micOch1"      "hg19"          "macFas5"      "monDom5"      "galGal4"
#> [6] "xenTro7"      "hg19-macFas5"
#>
#> [[4]]
#> [1] "rn5"              "micOch1"          "triMan1"          "galGal4"
#> [5] "anoCar2"          "xenTro7"          "galGal4-anoCar2"
#>
#> [[5]]
#> [1] "ochPri3"          "equCab2"          "loxAfr3"          "triMan1"
#> [5] "monDom5"          "galGal4"          "loxAfr3-triMan1"
```

Computing phenotypic associations with *phyloConverge*

After computing the permulated phenotypes, we can use *phyloConverge* to compute phenotypic associations of genomic elements. The following sections describes how *phyloConverge* can be used to compute convergence scores from alignments at different length scales.

Scoring an entire element

The first function that can be used to compute convergence score with the *phyloConverge* method is the `phyloConverge` function, which takes the following input arguments:

- `foregrounds`: A character vector containing the tip (and ancestral) foreground species
- `permulated_foregrounds`: A list of character vectors containing the null foreground species (i.e., the output of `getPermulatedPhenotypes`)
- `neutralMod`: The neutral tree model
- `maf`: an MSA object containing the sequence alignment
- `refseq`: The reference genome of the alignment

- `feature`: a feature object containing information on chromosomes, coordinates, and names (optional) of the features to be scores (default NULL)
- `alpha`: The target Type I error to control (default 0.05)
- `min.fg`: The minimum number of foreground species required to score for convergence (default 2)
- `method`: Scoring method for *phyloP* (default "LRT")
- `mode`: Scoring mode for *phyloP* (default "CONACC")
- `adapt`: A Boolean flag for performing adaptive permulation (default TRUE)

The `phyloConverge` function can be used to compute the convergence score of an entire input alignment (e.g., if the alignment file is a short alignment of a conserved non-coding element), specifically by setting `feature` as NULL.

In the example below, we are using 500 pre-computed permulated phenotypes. By default, `phyloConverge` uses an 'adaptive' strategy for computing permulation (empirical) p-values and scores, in which permulation is terminated once the target significance level has been surpassed. This strategy speeds up the computation, although user can choose to run the complete permulation by setting the `adapt` input argument to FALSE.

When `adapt` = TRUE, use the `alpha` input argument to specify the significance level to control. For example, if `alpha` = 0.05, `phyloConverge` will accurately compute permulation p-values <= 0.05, while some inaccuracies are allowed for permulation p-values outside of the rejection threshold.

The `phyloConverge` output is a data frame object with 4 columns:

- permPval: permulation p-value
- corr_score: bias-corrected score (negative denotes convergent rate deceleration, positive denotes convergent rate acceleration)
- uncorr_score: uncorrected score (negative denotes convergent rate deceleration, positive, denotes convergent rate acceleration)
- feature: feature names (defaults to featureX, but will be the same as the specified names of the features is the `feature` input argument is supplied)

```
permulated_foregrounds_file = 'permulated_foregrounds_500perms.RDS'
permulated_foregrounds = readRDS(paste0(pcpath, '/data/', permulated_foregrounds_file))

# adaptive
element_score_adaptive = phyloConverge(foregrounds, permulated_foregrounds, neutralMod,
                     msa, refseq="mm10")
#> [1] "Scoring feature 1 / 1 feature1"
print(element_score_adaptive)
#>   permPval corr_score uncorr_score  feature
#> 1     0.65  0.1870866    0.3009945 feature1

# non-adaptive
element_score_nonadp = phyloConverge(foregrounds, permulated_foregrounds, neutralMod,
                     msa, refseq="mm10", adapt=F)
#> [1] "Scoring feature 1 / 1 feature1"
print(element_score_nonadp)
#>    permPval corr_score uncorr_score  feature
#> 1 0.5856574  0.2323564    0.3009945 feature1
```

Scoring subregion(s) of an alignment

The `phyloConverge` function can also be used to score specific subregions in an alignment, which can be done by supplying a data frame in the BED file format containing the coordinates of the subregions to score. The data frame must be converted into a feature object before being supplied to `phyloConverge`, as follows:

```
# Scoring a single subregion
bed = data.frame("chr"="chr1", "start"=5, "end"=20, "name"="feature1")
print(bed)
#>    chr start end     name
#> 1 chr1     5  20 feature1
feature = convertBedToFeature(bed, "mm10")
subregion_score = phyloConverge(foregrounds, permulated_foregrounds, neutralMod,
                msa, "mm10", feature, adapt=F)
#> [1] "Scoring feature 1 / 1 feature1"
print(subregion_score)
#>   permPval corr_score uncorr_score  feature
#> 1 0.310757 -0.5075791   -0.9272003 feature1

# Scoring multiple subregions
bed = data.frame("chr"=rep("chr1",3), "start"=c(5, 25, 50), "end"=c(20, 35, 100),
         "name"=paste0("feature", c(1,2,3)))
print(bed)
#>    chr start end     name
#> 1 chr1     5  20 feature1
#> 2 chr1    25  35 feature2
#> 3 chr1    50 100 feature3
features = convertBedToFeature(bed, "mm10")

multiple_subregion_scores = phyloConverge(foregrounds, permulated_foregrounds,
                          neutralMod, msa, "mm10", features, adapt=T)
#> [1] "Scoring feature 1 / 3 feature1"
#> [1] "Scoring feature 2 / 3 feature2"
#> [1] "Scoring feature 3 / 3 feature3"
print(multiple_subregion_scores)
#>     permPval corr_score uncorr_score  feature
#> 1 0.27083333 -0.5672979  -0.92720034 feature1
#> 2 0.52000000 -0.2839967  -0.02822138 feature2
#> 3 0.09960159  1.0017337   1.32386316 feature3
```

Scanning an element

phyloConverge can be used to scan a given alignment and compute local convergent rate shift signals for arbitrary small resolutions in a sliding window manner. This is performed by the `scanWithPhyloConverge` function, which takes the following input arguments:

- `maf`: an MSA object containing the sequence alignment
- `foregrounds`: a character vector containing the names of the foreground species
- `permulated_foregrounds`: a list object containing the sets of fake foreground species generated by permulation
- `neutralMod`: neutral nucleotide substitution model

- `refseq`: reference species of the alignment
- `offset`: offset for the coordinates in the alignment. If NULL (default), the offset specified in the MSA object is used
- `stride`: interval (in bp) of positions of sliding windows to be scored (default 1, meaning scoring every nucleotide)
- `window`: size (in bp) of the sliding windows to be scored (default 1)
- `alpha`: target Type I error to control (default 0.05)
- `min.fg`: minimum number of foreground species required
- `adapt`: Boolean flag for performing adaptive permulation (adapt=TRUE for adaptive permulation, FALSE for complete permulations, default TRUE)

The function returns a data frame specifying the start and end coordinates of the sliding windows, as well as their corresponding phyloConverge scores. The example below shows the output of scanning *CNE031689* with strides (step sizes) of 5 and sliding window sizes of 10.

```
scanning_output = scanWithPhyloConverge(msa, foregrounds, permulated_foregrounds,
                    neutralMod, refseq="mm10", stride=10, window=10)
#> [1] 1
#> [1] "Scoring feature 1 / 1 feature1"
#> [1] 2
#> [1] "Scoring feature 1 / 1 feature1"
#> [1] 3
#> [1] "Scoring feature 1 / 1 feature1"
#> [1] 4
#> [1] "Scoring feature 1 / 1 feature1"
#> [1] 5
#> [1] "Scoring feature 1 / 1 feature1"
#> [1] 6
#> [1] "Scoring feature 1 / 1 feature1"
#> [1] 7
#> [1] "Scoring feature 1 / 1 feature1"
#> [1] 8
#> [1] "Scoring feature 1 / 1 feature1"
#> [1] 9
#> [1] "Scoring feature 1 / 1 feature1"
print(scanning_output)
#>   start end  corr_score
#> 1     1  10  0.39794001
#> 2    11  20 -0.76971808
#> 3    21  30 -0.09017663
#> 4    31  40 -0.48265374
#> 5    41  50 -0.81547557
#> 6    51  60  0.51444558
#> 7    61  70  0.06214791
#> 8    71  80  0.79454167
#> 9    81  90  0.97241648
```