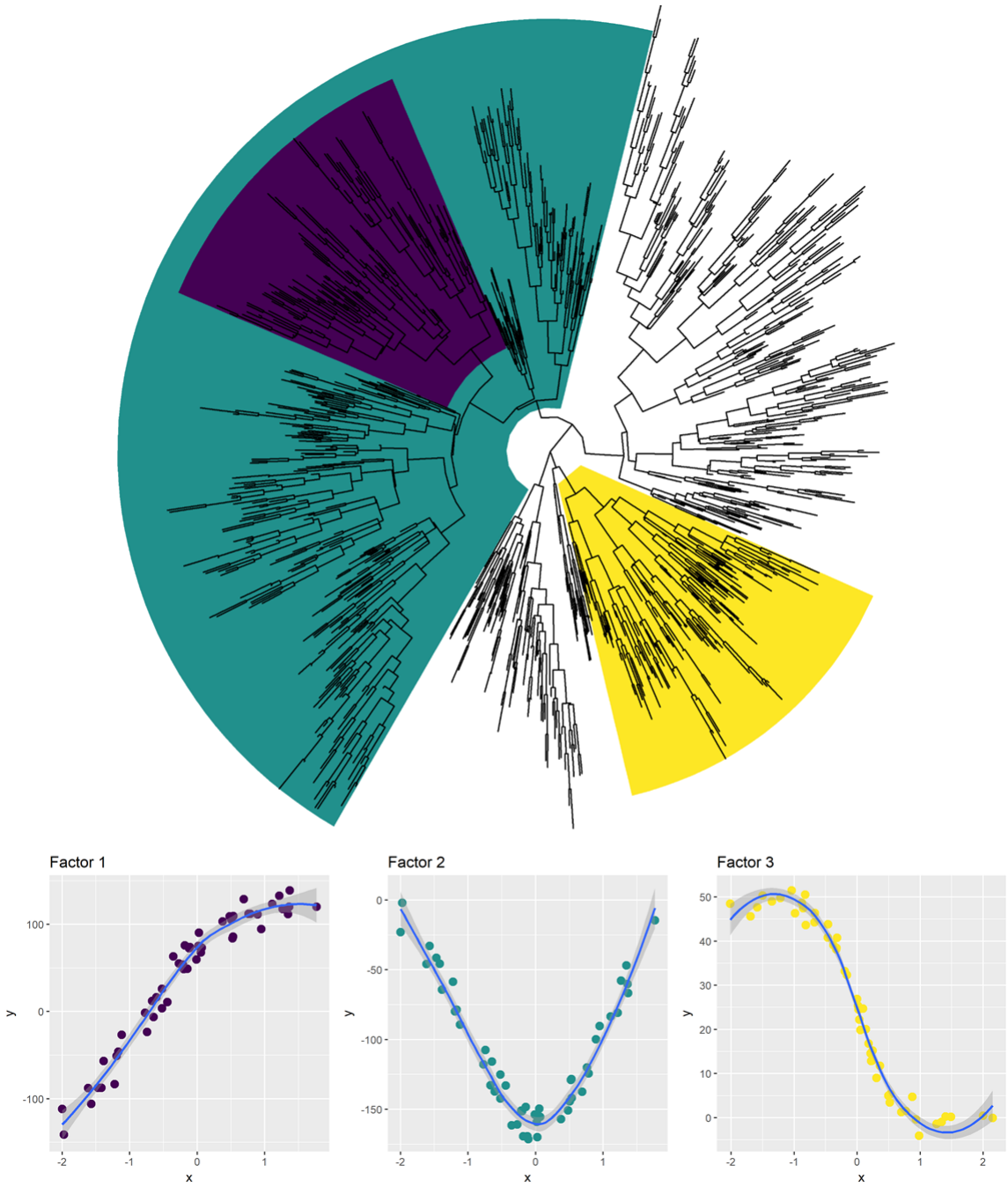
# phylofactor tutorial

*Alex Washburne*



# Introduction

Welcome to `phylofactor`!

This tutorial is a guide to the R package. For detailed knowledge on phylofactorization as an algorithm, check out the first paper in PeerJ which introduced the original concept for compositional data analysis and check out the big paper in Ecological Monographs which expounded the generalized algorithm.

Phylofactorization is an algorithm that breaks apart the phylogeny one edge at a time. The edges chosen maximize some measure of contrast between species on opposite sides of the edges - different means, different associations with meta-data, negative covariances, etc., are all measures of contrast between groups of species. The package `phylofactor` will help you break apart the phylogeny with a variety of contrasts & objective functions, summarize the splits, and visualize the tree.

This tutorial is an entry-level guide to the R package, giving you the big picture of how the package is organized and showcasing some of the main functionality. At any time, you can find a wealth of additional detail with `?` - the package has very detailed help files which contain many more examples than this tutorial.

I hope this tutorial helps you make sense of biological datasets! If you have any feedback on the package, tutorial, or more general math/science, please feel free to contact me at alex.d.washburne@gmail.com.

# Table of contents



# Installing Package

Until the package is submitted to CRAN, installing `phylofactor` will require manually downloading the packages `ggtree` and `Biostrings` from Bioconductor. The installation of `phylofactor` can be done with the following:

```
source("https://bioconductor.org/biocLite.R")
biocLite("ggtree")
biocLite("Biostrings")
install.packages('devtools')
devtools::install_github('reptalex/phylofactor')
```

The most common errors encountered during installation are missing dependencies. Try to read the errors, find out which packages were not found or failed to load, and then install them individually. For example, if the "package.name" package was not found or failed to load, try: `install.packages('package.name')`.

The phylofactor package, loaded below, imports three packages pretty central to phylofactor: `ape`, `data.table`, and `magrittr`.

```
library(phylofactor)
```

# Phylofactorization

To begin, we'll cover terminology, the dataset that comes with `phylofactor`, and the three main functions for phylofactorization: `twoSampleFactor`, `PhyloFactor`, and `gpf`.

Phylofactorization requires the following:

1) a phylogeny
2) a dataset - vector, matrix, or data frame - of a quantity you wish to use to separate species (e.g. abundance, presence/absence, etc.).
3) a taxonomy - (optional) data.frame whose first column contains the species in the phylogeny and whose second column contains a semicolon-delimited taxonomy, e.g.

```
##   OTU_ID
## 1 470382
## 2 432284
##                                                                    taxonomy
## 1    k__Bacteria; p__Firmicutes; c__Clostridia; o__Clostridiales; f__; g__; s__
## 2 k__Bacteria; p__Cyanobacteria; c__Chloroplast; o__Streptophyta; f__; g__; s__
```

4) meta-data (for regression-phylofactorization). A data frame with one row for every sample in the dataset. For generalized phylofactorization in `gpf`, the dataset and meta-data can all be contained in a single data frame as one would input into `glm`.

## Overview

### Terminology - factors, groups, bins

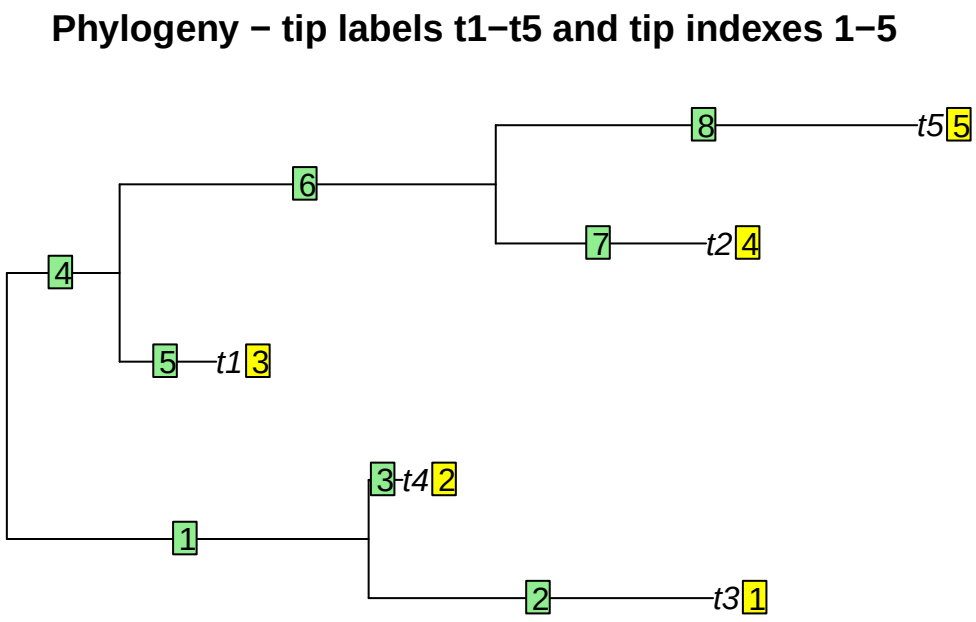
Phylofactorization is a greedy algorithm that iteratively partitions the phylogeny. The partitions are made based on a contrast of two groups separated by an uninterrupted edge or chain of edges. At the bare minimum, the algorithm generates three kinds of information: "factors", "groups", and "bins".

The "factors" in phylofactorization are the iterations - each iteration chooses one edge and there are two disjoint groups separated by that edge. The use of the word "factor" stems from factor analysis as phylofactorization, by choosing edges of interest, can be used to identify latent variables (presumably, traits that arose along the edges) and use them to produce low-rank approximations of our data. The word "factor" also relates to matrix factorization, as there is a set of vectors which can be used to represent our sequence of partitions and thus approxiate a matrix of data by iteratively factoring out the most important of these vectors. Finally, the

word "factor" also relates to "factor contrasts" as used in regression with categorical explanatory variables - the function `gpf` develops categorical variables indicating which side of an edge species lie on, and uses those categorical variables to identify differential abundances and associations with meta-data.

The word "groups" in this package always refers to two-member lists of the indexes on the tree (not the tip-labels, but the numerical indexes) corresponding to the two sets of species being contrasted. If there is a partition separating the first two tips from the last three - {t1,t2|t3,t4,t5} - the groups are {1,2} and {3,4,5}. These groups are produced with the function `getPhyloGroups`, and using that function can get you familiar with the groups in phylofactor objects:

```
set.seed(1)
tree <- rtree(5)
plot.phylo(tree,main='Phylogeny - tip labels t1-t5 and tip indexes 1-5')
tiplabels(1:5,cex=1,adj = -2)
edgelabels()
```




Notice that the sixth edge separates the monophyletic group {4,5} from the paraphyletic group {1,2,3} (i.e. the tips {t2,t5} from {t3,t4,t1}). The sixth member of the function `getPhyloGroups` contains the groups for this edge:

```
getPhyloGroups(tree)[[6]]
```

```
## [[1]]
## [1] 4 5
##
## [[2]]
## [1] 1 2 3
```

The first member in all groups consists of the descendants of the edge of interest

Finally, after a series of iterations, phylofactor will have a set of un-partitioned groups. The "bins" are the sets of species that have not been partitioned from one-another by a particular factor. At the first factor, the tree is partitioned along an edge separating two groups - these two groups are the two bins for the first factor. At the second factor, the two sub-trees corresponding to the two, split groups are searched for an edge separating groups within the sub-trees. One sub-tree is partitioned, yielding three bins. After k factors, there will be k pairs of groups that were contrasted, one group pair for each factor, and k+1 bins of un-partitioned groups.

Factors, groups and bins are important ways to summarize the results from phylofactorization. The phylofactor objects always contain this information.

```
pf <- FTmicrobiome$PF
pf$factors[1:2,1:2]
```

```
##                                 Group1                          Group2
## Factor 1 40 member Monophyletic clade 250 member Monophyletic clade
## Factor 2 16 member Monophyletic clade 234 member Paraphyletic clade
```

```
bins(pf$basis[,1:2]) %>% sapply(length) %>%
  data.frame('bin.sizes'=.)
```

```
##   bin.sizes
## 1       234
## 2        40
## 3        16
```

The first factor separated 40 species from the remaining 250 species. In the second factor, the remaining 250 species were partitioned again, with a 16 member monophyletic clade separated from the remainder. This results in three bins of sizes 234, 40, and 16.

Factors, groups and bins from a simulated dataset are visualized in the table below. When predictions are made with phylofactorization, they are made using the resultant bins (an exception is for compositional data, for which the basis elements are sufficient). Thus, by progressively whittling down the tree into finer and finer bins, phylofactorization gives us a series of progressively more detailed approximations of our data.

### Factors, Groups, and Bins

### Dataset: FTmicrobiome

The dataset provided with the R package is a time-scrambled subset of 10 samples for each of two patients' two body sites - feces and tongue - published in Caporaso et al. (2011). The data, `FTmicrobiome` is provided to help you experiment with phylofactorization.

```
data(FTmicrobiome)
names(FTmicrobiome)
```

```
## [1] "OTUTable" "tree"     "X"        "taxonomy" "PF"       "PF.Fstat"
```

The data contains `OTUTable`, a table of operational taxonomic units (OTUs) whose rows are species and columns are samples. There is also a phylogeny `tree`, meta-data vector `X`,taxonomy `taxonomy`, and two phylofactor objects `PF` and `PF.Fstat`.

### Simulating Data for phylofactorization

For our purposes, we'll simulate data to see how phylofactor functions aim to identify known effects. Once we understand how the main functions work, then we'll do some analyses on the `FTmicrobiome` data. To create

**pf$factors**

| | Group1 | Group2 |
|---|---|---|
| Factor 1 | 51 member Monophyletic clade | 49 member Monophyletic clade |
| Factor 2 | 13 member Monophyletic clade | 38 member Paraphyletic clade |

**pf$bins**

| Bin.Size |
|---|
| 13 |
| 38 |
| 49 |

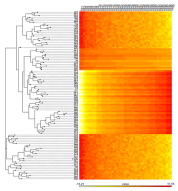
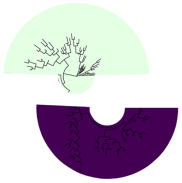
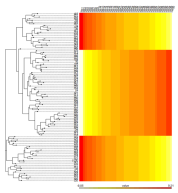
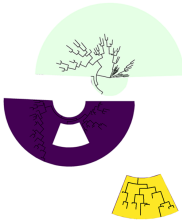
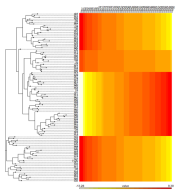
| Factor | pf.tree | Group1 | Group2 | bins | pf.predict |
|---|---|---|---|---|---|
| 0 |  | NA | NA |  | Raw Data  |
| 1 |  |  |  |  |  |
| 2 |  |  |  |  |  |

Figure 1:

a simulated dataset with a working taxonomy, we'll grab a random sample of 100 species from `FTmicrobiome` - that way we'll have their phylogeny and matching taxonomy.

```
set.seed(1)
tree <- FTmicrobiome$tree
species <- sample(tree$tip.label,100)
tree <- drop.tip(tree,setdiff(tree$tip.label,species))
tree$edge.length <- rep(1,Nedge(tree))
## this makes prettier visualizations. For ultrametric trees, I recommend phytools::force.ultrametric
taxonomy <- FTmicrobiome$taxonomy
```

For the demonstrations below, we'll use two or three known clades that we aim to extract. The third clade will be a sub-clade of clade 1.

```
clade1 <- phangorn::Descendants(tree,128,'tips')[[1]]
clade2 <- phangorn::Descendants(tree,186,'tips')[[1]]
clade3 <- phangorn::Descendants(tree,131,'tips')[[1]]
paste('clade1 has',length(clade1),'species, clade2 has',length(clade2),'species and clade3 has',length(
```

```
## [1] "clade1 has 48 species, clade2 has 12 species and clade3 has 19 species"
```

### twoSampleFactor

The simplest dataset one can phylofactor is a vector of data with one entry for each species. With two vectors of data, one for each group, two-sample tests are used to measure the difference between groups. `twoSampleFactor` is partitions the phylogeny based on the test statistic from two-sample tests of the groups separated by edges. Given a vector of data, such as body-mass, population size, or presence/absence in an environment, `twoSampleFactor` partitions the vector using a `TestFunction`, such as a t-test, Wilcox test, Fisher exact test, or a customized test function of your design.

Consider log-normally distributed body size across our 100 species. If we simulate `clade1` having unusually large body size and `clade2` having unusually low body size, `twoSampleFactor` can identify our clades from the data using the default two-sample test in `twoSampleFactor`: a two-sample t-test of equal variances.

```
set.seed(1)
BodySize <- rlnorm(100)
BodySize[clade1] <- rlnorm(length(clade1))*4
BodySize[clade2] <- rlnorm(length(clade2))/4

logBodySize <- log(BodySize)
pf_twoSample <- twoSampleFactor(logBodySize,tree,nfactors=2)
```

```
##
 1 factor completed in 0.000203 minutes.    Estimated time of completion: 2019-02-25 06:33:45

 2 factors completed in 0.00055 minutes.     Estimated time of completion: 2019-02-25 06:33:45
```

```
pf_twoSample
```

```
##        phylofactor object from function twoSampleFactor
##        ------------------------------------------------
## Method                    : contrast
## Number of species         : 100
## Number of factors         : 2
## Largest non-remainder bin : 48
## Number of singletons      : 0
## Paraphyletic Remainder    : 40 species
```

```
##
## ------------------------------------------------------------
## Factor Table:
##                                 Group1                        Group2
## Factor 1 48 member Monophyletic clade 52 member Monophyletic clade
## Factor 2 12 member Monophyletic clade 40 member Paraphyletic clade
```

Note: you can see that the twoSampleFactor function printed a summary of elapsed time and an estimated time of completion. For large datasets, phylofactorization can be computationally intensive and such time estimates can let you know if you have enough time to go for a run or grab a cup of coffee (or both!) before the results are in. Efficiency is key.

For Gaussian data (log-body-size simulated here), a t-test of equal variances is an appropriate two-sample test. However, for many data such a test is not appropriate. For example, presence/absence data contain only 0's and 1's and such data for two groups may be better contrasted by Fisher's exact test to see if there's the same rate of 1's in each group. Fisher's exact test is built into `twoSampleFactor` via the input `method="Fisher"`. We can also parallelize the computation with the argument `ncores`.

```
presence <- rbinom(100,1,.5)
presence[clade1] <- rbinom(length(clade1),1,.9)
presence[clade2] <- rbinom(length(clade2),1,.1)

pf_fisher <- twoSampleFactor(presence,tree,nfactors=2,method = "Fisher",ncores = 2)
```

```
pf_fisher
```

```
##        phylofactor object from function twoSampleFactor
##        ------------------------------------------------
## Method                     : Fisher
## Number of species          : 100
## Number of factors          : 2
## Largest non-remainder bin  : 51
## Number of singletons       : 0
## Paraphyletic Remainder     : 37 species
##
## ------------------------------------------------------------
## Factor Table:
##                                 Group1                        Group2
## Factor 1 51 member Monophyletic clade 49 member Monophyletic clade
## Factor 2 12 member Monophyletic clade 37 member Paraphyletic clade
```

Two-sample phylofactorization can also be done with customized objective functions, including customized objective functions that have access to the phylogeny (e.g. for contrasting ancestral states reconstructed for contrasted groups in a sub-tree). All that is detailed in `? twoSampleFactor`.

## PhyloFactor

What if we have observations of a quantity, such as abundance or body size, across many samples and want to know the associations of our data with some meta-data, such as latitude?

The function `PhyloFactor` is built to analyze matrices of data - rows are species and columns are samples - by projecting the data onto contrast basis elements (referred to as balancing elements in compositional data analysis). Projecting standard Gaussian data onto contrast basis elements yields a t-statistic, like that used in twoSampleFactor, for each column in our matrix. Projecting log relative abundance data yields an isometric log-ratio transform commonly used for compositional data analysis. If that t-statistic changes across samples,

it means the two groups are changing relative to one-another, and measuring the magnitude or predictability of that change can be used as a measure of contrast between two groups.

`PhyloFactor` revolves around projecting data, or some transform of the data, onto contrast basis elements produced with the function `ilrvec`:

```
ilrvec(list(c(4,5),c(1,2,3)),n=5)
```

```
## [1] -0.3651484 -0.3651484 -0.3651484  0.5477226  0.5477226
```

If we project the data from species 1-5 on the vector above, it will take a scaled mean of species 1-3 and subtract it from a (differently) scaled mean of species 4 and 5. The scaling constant is chosen to stabilize the variance of our difference-of-means. For a sequence of binary partitions, these vectors produce a basis:

```
v1 <- ilrvec(list(c(4,5),c(1,2,3)),n=5)  ## separate {1,2,3|4,5}
v2 <- ilrvec(list(1,c(2,3)),n=5)         ## separate {1|2,3}
basis <- matrix(c(v1,v2),ncol=2,byrow=F)
basis
```

```
##            [,1]       [,2]
## [1,] -0.3651484  0.8164966
## [2,] -0.3651484 -0.4082483
## [3,] -0.3651484 -0.4082483
## [4,]  0.5477226  0.0000000
## [5,]  0.5477226  0.0000000
```

and the basis is orthonormal:

```
t(basis) %*% basis
```

```
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1
```

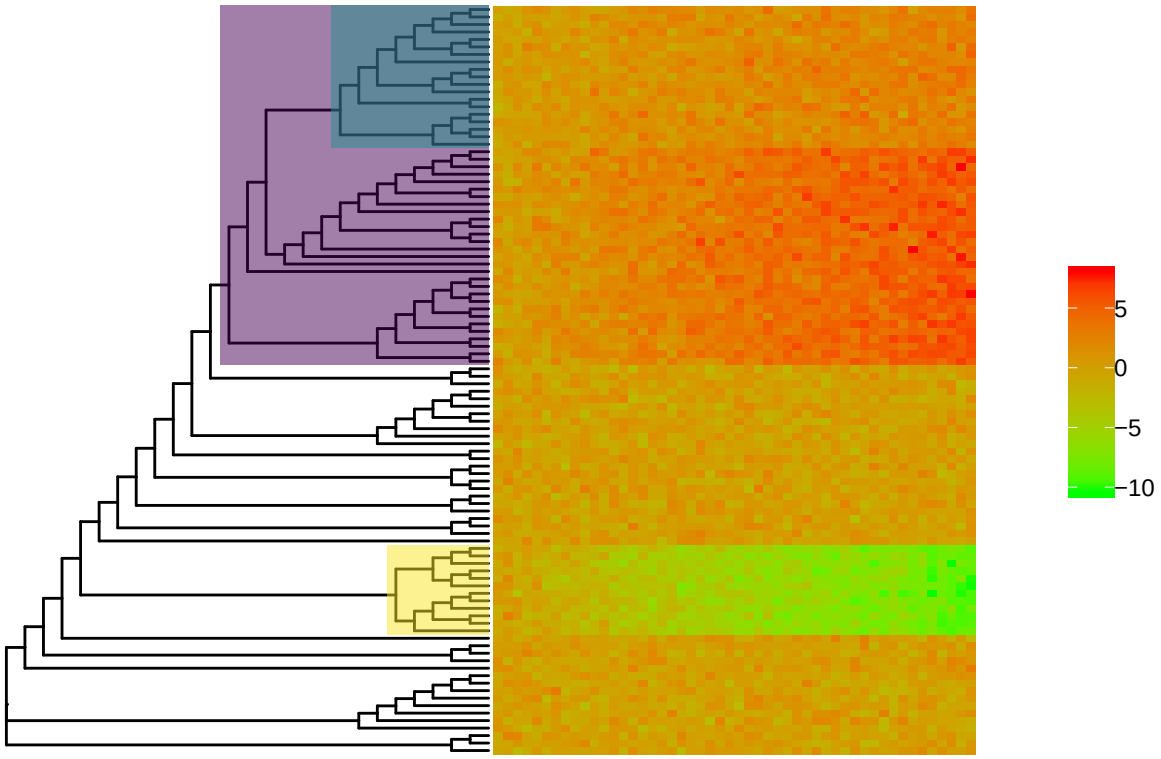
Contrast basis elements made with `ilrvec` represent differences of means between two groups of species; every edge in the phylogeny sepearates the tree into two disjoint groups that can be used to make an `ilrvec` and partitions can be made based on objective functions of the projection of data onto the vector. This flexible style of analysis, centered around projecting raw or transformed data directly onto vectors from `ilrvec`, is the heart of the function `PhyloFactor`.

Projections onto the contrast basis allow a suite of analyses, ranging from phylogenetic-PCA implemented in `PhyCA` to glm and gam-based analyses. The default for `PhyloFactor` is a wrapper around `glm` and accepts many of the same inputs as optional arguments.

We'll simulate a dataset in which `clade1` has body size that increases with meta-data (latitude), `clade2` has body size that decreases with latitude and `clade3`, a monophyletic subset of species from `clade1`, has body size that increases with latitude but not as strongly as the rest in `clade1`.

```
set.seed(1)
m=length(species)
n=50 #number of samples
MetaData <- data.frame('latitude'=runif(n,0,90))
BodySize <- matrix(rlnorm(m*n),nrow=m)
rownames(BodySize) <- tree$tip.label                    # This step is necessary for PhyloFactor
for (spp in clade1){
  BodySize[spp,] <- rlnorm(n,meanlog=MetaData$latitude/15)
}
for (spp in clade2){
  BodySize[spp,] <- rlnorm(n,meanlog=-MetaData$latitude/10)
}
```

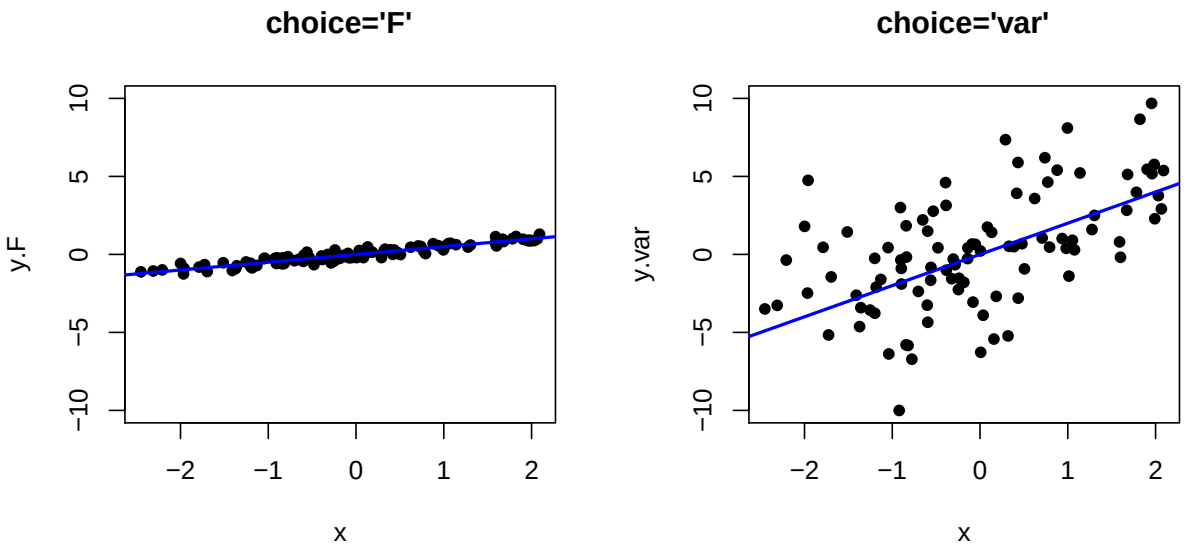
```
for (spp in clade3){
  BodySize[spp,] <- rlnorm(n,meanlog=MetaData$latitude/30)
}

cols <- viridis::viridis(3)
pf.heatmap(tree=tree,Data=log(BodySize[,order(MetaData$latitude)]),color=NA)+
  ggtree::geom_hilight(128,fill=cols[1])+
  ggtree::geom_hilight(131,fill=cols[2])+
  ggtree::geom_hilight(186,fill=cols[3])
```




Our dataset has clear patterns along a latitudinal gradient, but different clades have different associations with latitude. Regression of latitude on the contrasts of body sizes can partition the phylogeny and identify these three clades with distinct BodySize~latitude associations.

`PhyloFactor` requires specifying a formula like those used in `glm`. The projections of the data onto contrast basis elements must be referred to as `Data` in the input `frmla`. In other words, `Data~latitude` will predict body size with latitude, whereas `latitude~Data` will predict latitude with body sizes.

Phylofactorization iterates over all the edges in the phylogeny, each one producing a model using our `ilrvec`-projected data and our metadata - how do we pick the best model? The two default options for regression-based phylofactorization are `choice='var'`, which maximizes the explained variance, and `choice='F'`, which maximizes the F-statistic from regression (the ratio of explained to unexplained variance). The difference between the two objective functions is illustrated below:




Setting `choice='F'` finds factors that more predictable through a higher ratio of explained to unexplained variance (the overall F-statistic), whereas setting `choice='var'` finds factors that capture more of the variance in the dataset through a higher explained variance. In a sense, `choice='F'` finds bioindicator clades whereas `choice='var'` finds clades capturing the largest changes in the data. These two objective functions are calculated internally with the function `getStats`.

```
g.F <- glm(y.F~x)
g.var <- glm(y.var~x)
output <- rbind(getStats(g.F),getStats(g.var))
rownames(output) <- c("y.F","y.var")
output
```

```
##               Pval        F ExplainedVar
## y.F   0.000000e+00 819.9783    0.3207093
## y.var 4.607228e-10  47.9439    4.8578972
```

The `Pval` reported is the P-value from an overall F-test of the model.

To find the most predictable phylogenetic factors driving latitude-bodysize associations, we'll use `choice='F'` and a formula `Data~latitude` to find which clades have the most predictable changes in body size with latidue. We can now use `PhyloFactor`:

```
pf_PhyloFactor <- PhyloFactor(BodySize,tree,MetaData,
                              frmla = Data~latitude,nfactors=3,choice='F')
```

```
pf_PhyloFactor
```

```
##        phylofactor object from function PhyloFactor
##        --------------------------------------------
## Method                    : glm
## Choice                    : F
## Formula                   : Data ~ latitude
## Number of species         : 100
## Number of factors         : 3
## Largest non-remainder bin : 48
## Number of singletons      : 0
```

```
## Paraphyletic Remainder    : 40 species
##
## ------------------------------------------------------------
## Factor Table:
##                                Group1                       Group2       F
## Factor 1 12 member Monophyletic clade 88 member Monophyletic clade 5615.80
## Factor 2 48 member Monophyletic clade 40 member Paraphyletic clade 1716.40
## Factor 3 19 member Monophyletic clade 29 member Paraphyletic clade  371.96
##          Pr(>F)
## Factor 1      0
## Factor 2      0
## Factor 3      0
```

The factors are also summarized in our outputted phylofactor object

```
pf_PhyloFactor$factors
```

```
##                                Group1                       Group2
## Factor 1 12 member Monophyletic clade 88 member Monophyletic clade
## Factor 2 48 member Monophyletic clade 40 member Paraphyletic clade
## Factor 3 19 member Monophyletic clade 29 member Paraphyletic clade
##                   F Pr(>F)
## Factor 1 5615.7960      0
## Factor 2 1716.3536      0
## Factor 3  371.9624      0
```

The first factor is most likely our `clade2` which had a negative association with latitude. The model that was used to partition `clade2` from the rest is available in our phylofactor object:

```
pf_PhyloFactor$models[[1]]
```

```
##
## Call:  Data ~ latitude
##
## Coefficients:
## (Intercept)     latitude
##      0.2294      -0.4227
##
## Degrees of Freedom: 49 Total (i.e. Null);  48 Residual
## Null Deviance:       5300
## Residual Deviance: 44.92     AIC: 142.5
```

The third factor is most likely our `clade3`, a subset of species from `clade1` being partitioned from the remainder of `clade1`. Recall that species in `clade3` have body sizes that increase with latidue but at about half the rate of the rest of species in `clade1`. Take a quick quiz to test your understanding: what is the sign (and, extra credit, the magnitude) of the coefficient for the latitude-association going to be in the third model?

```
pf_PhyloFactor$models[[3]]
```

```
##
## Call:  Data ~ latitude
##
## Coefficients:
## (Intercept)     latitude
##     0.09355     -0.11495
##
## Degrees of Freedom: 49 Total (i.e. Null);  48 Residual
```

```
## Null Deviance:       438.8
## Residual Deviance: 50.16     AIC: 148
```

The sign is negative because Group1 (`clade3`) does not increase with latitude as fast as Group2 (the remainder of `clade1`), and so with increasing latitude the difference in means, Group1-Group2, will decrease. Put differently, with increasing latitude the body size of `clade3` relative to the remainder of `clade1` decreases.

On a per-species basis, the mean log-body-size of `clade3` increases at a rate (1/30) half the rate (1/15) of the remainder of species in `clade1`, but the coefficient of regression is not log(1/2). The reason for this disreprancy is due to the fact that `PyloFactor` outputs models from regression with the data projected onto `ilrvec` - the `ilrvec` are constructed with scaling constants to ensure a more fair comparison of big vs. small groups (this scaling constant stabilizes the variance of a differnece of means). The extraction of relative, species-specific rates of increase of body size with latitude is an exercise left to the reader.

The `PhyloFactor` function is wrapped around `glm` and the built-in machinery allows the user to partition the tree based on overall F-statistics or explained variance for any `glm` style model. Mutiple regression, predicting meta-data with our balance contrasts, and more, can all be implemented by passing optional arguments to `glm`. For example, we can do multiple regression predicting a categorical variable (e.g. sample-site) with an offset via weighted binomial regression:

```
MetaData$x <- rnorm(n)
MetaData$z <- rnorm(n)
MetaData$y <- factor(rep(c('a','b'),each=n/2))
wts <- seq(0,1,length.out=n)
ix <- 20:30
pf_demo <- PhyloFactor(BodySize,tree,MetaData,frmla=y~Data+x+offset(z),
                       family=binomial,weights=wts,subset=ix,
                       nfactors=2)
```

```
pf_demo
```

```
##        phylofactor object from function PhyloFactor
##        --------------------------------------------
## Method                    : glm
## Choice                    : var
## Formula                   : y ~ Data + x + offset(z)
## Number of species         : 100
## Number of factors         : 2
## Frac Explained Variance   : 0.00328
## Largest non-remainder bin : 1
## Number of singletons      : 2
## Paraphyletic Remainder    : 98 species
##
## ------------------------------------------------------------------
## Factor Table:
##          Group1                       Group2    ExpVar      F   Pr(>F)
## Factor 1    tip 99 member Monophyletic clade 0.0018403 5.9845 0.025759
## Factor 2    tip 98 member Paraphyletic clade 0.0014356 3.5125 0.080371
```

The weights input to PhyloFactor were used and retained in our models for downstream analysis:

```
data.frame('sample'=ix,
           'model_weights'=pf_demo$models[[1]]$prior.weights,
           'input_weights'=wts[ix])[1:4,]
```

See `? PhyloFactor` for a full tour of the functionality, including the built-in generalized additive model capabilities and customized objective functions (e.g. maximize t-statistic for one explanatory variable while controlling for others).

To recap, `PhyloFactor` partitions the phylogeny based on objective functions of the data projected onto contrast basis elements and flexible, regression-style analyses can be implemented by using the term `Data` in the input `frmla`.

## PhyCA - phylogenetic principal components analysis

For exploratory analyses like principal components analysis, one is less interested in the variance explained by regression and more interested in the directions capturing raw variance of the data. Phylogenetic principal components analysis, `PhyCA`, finds the contrast basis elements which capture most of the variance in the data. While `PhyCA` can be implemented by setting `method='max.var'` in `PhyloFactor`, `PhyCA` is a more minimial, intuitively named function with concise output and more customized summary functions.

```
phyca <- PhyCA(BodySize,tree,ncores=2,ncomponents = 2)
```

```
phyca
```

```
##        phylofactor object from function PhyCA
##        --------------------------------------
## Method                     : max.var
## Number of species          : 100
## Number of factors          : 2
## Frac Explained Variance    : 0.568
## Largest non-remainder bin : 48
## Number of singletons       : 0
## Paraphyletic Remainder     : 40 species
##
## ----------------------------------------------------------------
## Factor Table:
##                                  Group1                         Group2  ExpVar
## Factor 1 12 member Monophyletic clade 88 member Monophyletic clade 0.42636
## Factor 2 48 member Monophyletic clade 40 member Paraphyletic clade 0.14213
```

The third column in the factor table shows the percentage of variance explained by each contrast basis element.

## gpf : Generalized Phylofactorization

What if we don't want to assume our group-aggregated data are Gaussian or easily transformed to Gaussian? For example, presence-absence data are more appropriately analyzed via logistic regression, not least squares. If our data are presence-absences of organisms across a range of environments, generalized phylofactorization can be used to partition the phylogeny along edges separating species with different log-odds of presence, including species whose probabilities of discovery have different associations with meta-data.

The generalized phylofactorization function, `gpf`, enables analysis of our data as exponential family random variables (e.g. Bernoulli, binomial, Poisson, negative binomial, gamma, etc.). Most often, this is done with the help of a surrogate categorical variable, `phylo`, indicating which group a species is in. The `phylo` factor can be used in formulas for regression to identify differential meta-data associations between groups separated by edges.

The `gpf` function implements one of four different algorithms: `mStable`, `phylo`, `CoefContrast`, and `mix`. The algorithms, their input, pros and cons are all summarized below

| Algorithm | Description | Required Input | Optional Input | Pros | Cons |
|---|---|---|---|---|---|
| **phylo** | Constructs categorical variable, `phylo`, to model group-specific associations with meta-data | • Data (data.frame)<br>• tree<br>• frmla.phylo | • PartitioningVariables<br>• advanced model functions (e.g. gam, nls) | • Best able to ID correct edges<br>• Explicit model of between-group differences & within-group shared coefficients<br>• Allows species-specific coefficients | • Slowest algorithm<br>• Scales poorly |
| **mStable** | Aggregates data within Sample x group and uses categorical variable, `phylo`, to model gropu-specific associations with meta-data | • Data (matrix)<br>• tree<br>• frmla.phylo<br>• MetaData | • PartitioningVariables<br>• Data (data.frame with column 'Sample')<br>• advanced model functions (e.g. gam, nls) | • Scales well<br>• Explicit model of between-group differences & within-group shared coefficients<br>• Good ability to ID correct edge for mean-centered data or similar row-means of Data matrix | • Less accurate than `phylo`,<br>• slower than `CoefContrast` |
| **CoefContrast** | From many species-specific models, constructs matrix of `PartitioningVariables`' standardized coefficients. Matrix is projected onto contrast basis elements to identify groups with most different coefficients | • Data (data.frame)<br>• tree<br>• frmla.phylo | • PartitioningVariables<br>• frmla | • Fastest Algorithm<br>• Scales well<br>• Relates to reduced-rank regression | • Less accurate<br>• No explicit model for shared coefficients<br>• Not guaranteed<br>• Limited to glm |
| **mix** | Use `CoefContrast` to identify top edges for consideration in algorithm `phylo` | • Data (data.frame)<br>• tree<br>• frmla.phylo | • PartitioningVariables<br>• frmla<br>• alpha | • CoefContrast speed-up, phylo algorithm accuracy | • Still slower than mStable<br>• Scales like phylo, modulated by alpha<br>• Limited to glm |

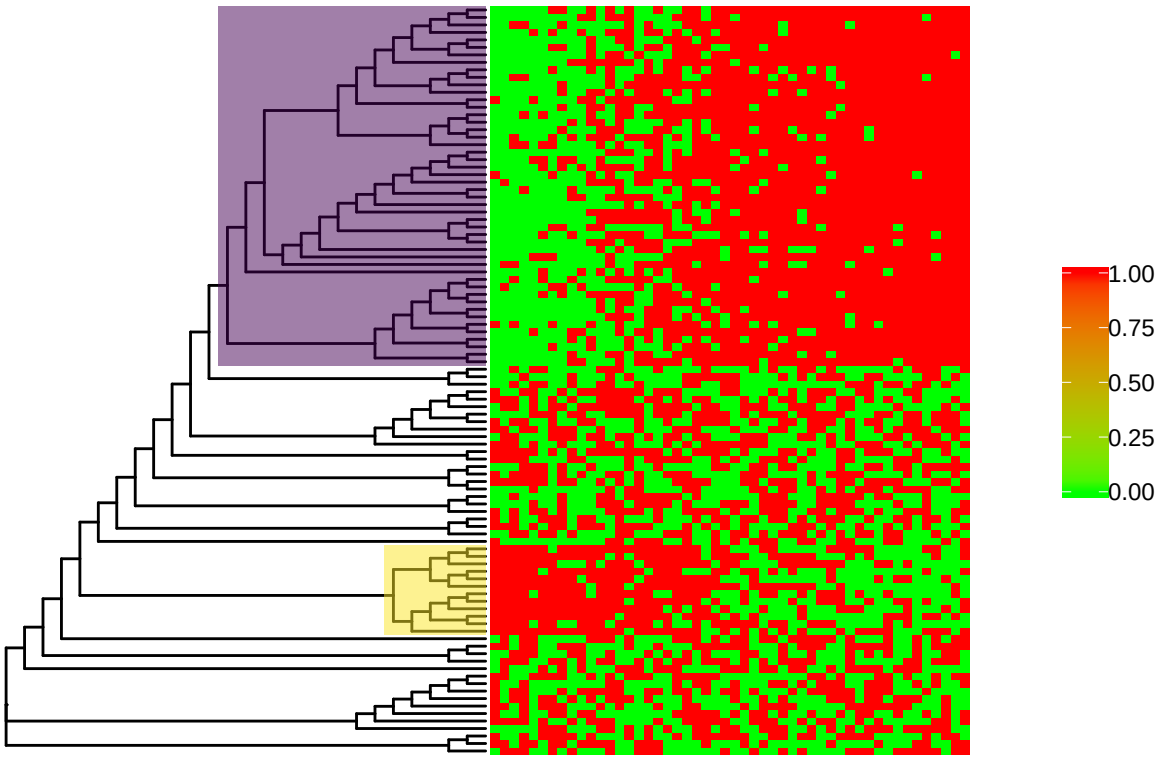
Figure 2: Summary of algorithms for gpf

### Algorithm Summary

### algorithm='mStable'

The mStable algorithm is most like `PhyloFactor` in allowing a matrix-MetaData input pair, so we'll start there.

```
set.seed(2)
ilogit <- function(x) 1/(1+exp(-x))
Presence <- matrix(rbinom(m*n,1,.5),nrow=m)
rownames(Presence) <- tree$tip.label
p1 <- ilogit(.08*(MetaData$latitude-30))
p2 <- ilogit(-.05*(MetaData$latitude-60))
for (spp in clade1){
  Presence[spp,] <- rbinom(n,1,prob=p1)
}
for (spp in clade2){
  Presence[spp,] <- rbinom(n,1,prob=p2)
}
pf.heatmap(tree=tree,Data=Presence,column.order=order(p1))+
  ggtree::geom_hilight(128,fill=cols[1])+
  ggtree::geom_hilight(186,fill=cols[3])
```




For `algorithm=mStable`, the required input is a `frmla.phylo`, which uses the term `phylo` as a two-level factor that will be used to contrast the probabilities of discovery of species in the different groups. For logistic regression, we need two matrices: the `Successes` and the `Failures`, storied in a named list. That list can be input as our `Data`.

```
PA_Data <- list('Successes'=Presence,'Failures'=1-Presence)
MetaData <- data.frame('latitude'=MetaData$latitude)
pf_gpf_mStable <- gpf(PA_Data,tree,frmla.phylo = cbind(Successes,Failures)~phylo*latitude,
                      MetaData=MetaData,family=binomial,nfactors=2,algorithm='mStable')
```

```
pf_gpf_mStable
```

```
##        phylofactor object from function gpf
##        ------------------------------------
## Method                     : gpf
## Algorithm                  : mStable
## Formula                    : cbind(Successes, Failures) ~ phylo * latitude
## Partitioning Variables     : {}
## Number of species          : 100
## Number of factors          : 2
## Largest non-remainder bin  : 48
## Number of singletons       : 0
## Paraphyletic Remainder     : 40 species
##
## -----------------------------------------------------------------
## Factor Table:
##                                Group1                         Group2
```

```
## Factor 1 48 member Monophyletic clade 52 member Monophyletic clade
## Factor 2 12 member Monophyletic clade 40 member Paraphyletic clade
```

The models in our phylofactor object indicate how mStable aggregation interfaces with `glm`:

```
pf_gpf_mStable$models[[1]]
```

```
##
## Call:  cbind(Successes, Failures) ~ phylo * latitude
##
## Coefficients:
##     (Intercept)           phyloS         latitude  phyloS:latitude
##        -2.51670          3.04041          0.08033         -0.08924
##
## Degrees of Freedom: 99 Total (i.e. Null);  96 Residual
## Null Deviance:       1370
## Residual Deviance: 81.94     AIC: 469.5
```

Notice there are 99 null degrees of freedom. We started with `n=50` samples across `m=100` species, where did the 99 degrees of freedom come from? For mStable aggregation of binomial data, the total number of Successes are added within each group {R,S} for each edge, the total number of Failures is added within each group, and a data frame is constructed with a variable, `phylo`, indicating the group {R,S}. Thus, the 99 degrees of freedom come from us having 50 observations of group R and 50 observations of group S, and one degree of freedom lost from the `(Intercept)` term.

The `phylo` factor can be seen in the data for each model:

```
pf_gpf_mStable$models[[1]]$data[c(3,4,9,10),]
```

```
##       Sample Successes Failures phylo  latitude
## 1: Sample 10         7       41     R  5.560764
## 2: Sample 10        34       18     S  5.560764
## 3: Sample 13        40        8     R 61.832056
## 4: Sample 13        25       27     S 61.832056
```

The `phylo` factor indicates whether species are in Group1 (R) or Group2 (S). For sample 10 (the 10th column of our data matrix), there were 31 presences (Successes) in group1 and 17 absences (Failures) - 48 species in total, 31 of which were present. In that same sample, group 2 contained 31 presences & 21 absences. At that low latitude (5.56), there was a higher rate of presence in `phylo==R` than `phylo==S`, annd that result flips for higher latitude (61.83) in sample 13. With increasing latitude, `phylo==R` has a lower probability of being present whereas `phylo==S` a higher probability. Thus, the association between the probability of discovery and latitude is higher for `phylo==S` than for `phylo==R`, explaining the positive coefficient of `phyloS:latitude` in the output model.

The term `phylo` can be used flexibly in a formula. If you don't input `PartitioningVariables`, the function `gpf` will (for algorithms `mStable` and `phylo`), partition the phylogeny based on the sum of deviance from all terms containing `phylo` (including the within-group mean of coefficient `phyloS`). Input `PartitioningVariables` allows you to partition the phylogeny based on different, group-specific associations with one variable while controlling for potentially different group-specific associations with other variables. For example, If we wanted to control for global associations with `temperature` while partitioning based on `latitude`, we'd input

```
frmla.phylo=cbind(Successes,Failures)~temperature+phylo*latitude
```

whereas if we wanted to control for group-specific `temperature` associations but partition based on `latitude`, we'd input

```
frmla.phylo=cbind(Successes,Failures)~phylo*temperature+phylo*latitude, PartitioningVariables='latitude
```

More flexible modelling is available for other algorithms. While `mStable` aggregation aggregates data within groups and loses any species-specific information, it is extremely fast, can be used with generalized additive modelling, and scales well with big data. The other algorithms all allow you to model species-specific effects with the formula variable `Species`.

### algorithm='phylo'

While `mStable` aggregation takes as input a `Data` matrix and `MetaData` data frame similar to `PhyloFactor`, all other algorithms require the input `Data` to be in a single data table that must have the column `Species` along with all other variables used in the `frmla` and `frmla.phylo`. Converting a matrix to the appropriate data frame can be done with the function `matrix.to.phyloframe`.

```
PA_datatable <- matrix.to.phyloframe(Presence,MetaData,data.name = 'Successes')
PA_datatable[,Failures:=1-Successes]
PA_datatable[95:104,]
```

```
##     Species    Sample Successes  latitude Failures
##  1:  564806  Sample 1         1 23.895780        0
##  2:  840961  Sample 1         1 23.895780        0
##  3:  182723  Sample 1         1 23.895780        0
##  4:  186392  Sample 1         0 23.895780        1
##  5:  175509  Sample 1         0 23.895780        1
##  6:  323778  Sample 1         0 23.895780        1
##  7:  296960 Sample 10         0  5.560764        1
##  8:  574263 Sample 10         0  5.560764        1
##  9:  194707 Sample 10         1  5.560764        0
## 10:  191660 Sample 10         0  5.560764        1
```

This data frame allows us to model species-specific effects using the term `Species`. For example, we can control for species-specific means with `frmla.phylo=cbind(Successes,Failures)~Species+phylo*latitude`.

The `algorithm='phylo'` is much slower but can be much more accurate. The robustness of the `algorithm='phylo'` is due to its ability to model differences in species-specific means and variances. For this simulation, we'll use a subset of 1/10 the samples in our original data

```
sample.subset <- paste('Sample',seq(10,50,by=10)) # grab samples 5, 10, 15, ...
pf_phylo <- PA_datatable[Sample %in% sample.subset] %>%
gpf(tree,frmla.phylo=cbind(Successes,Failures)~Species+phylo*latitude,
                family=binomial,algorithm='phylo',nfactors=2,ncores=4)
```

```
pf_phylo
```

```
##        phylofactor object from function gpf
##        ------------------------------------
## Method                    : gpf
## Algorithm                 : phylo
## Formula                   : cbind(Successes, Failures) ~ Species + phylo * latitude
## Partitioning Variables    : {}
## Number of species         : 100
## Number of factors         : 2
## Largest non-remainder bin : 48
## Number of singletons      : 0
## Paraphyletic Remainder    : 40 species
##
## ------------------------------------------------------------
## Factor Table:
```

```
##                                 Group1                         Group2
## Factor 1 48 member Monophyletic clade 52 member Monophyletic clade
## Factor 2 12 member Monophyletic clade 40 member Paraphyletic clade
```

We can indicate the coefficients of latitude-associations for each Group1 relative to Group2 by stripping coefficients from our models:

```
M <- rbind(coef(pf_phylo$models[[1]])[c('latitude')],
coef(pf_phylo$models[[2]])[c('latitude')])
rownames(M) <- paste('Factor',1:2)

cbind(pf_phylo$factors,M)
```

```
##                                 Group1                         Group2
## Factor 1 48 member Monophyletic clade 52 member Monophyletic clade
## Factor 2 12 member Monophyletic clade 40 member Paraphyletic clade
##             latitude
## Factor 1  0.11757189
## Factor 2 -0.08081661
```

The `algorithm='phylo'` is the most accurate and versatile, but it is extremely slow scales poorly to large datasets. Some computational challenges for `algorithm='phylo'` can be overcome with a mixed algorithm, discussed later, which uses a fast algoritmm to narrow down to a subset of edges for `algorithm='phylo'`.

### algorithm='CoefContrast'

The algorithms `mStable` and `phylo` explicitly constructed a variable, `phylo`, that is used to identify differential meta-data associations via factor contrasts. An alternative method for finding differential meta-data associations is to explicitly test for differences in coefficients from regression. The method `CoefContrast` builds a matrix of standardized regression coefficients (the z-statistics obtained from `summary.glm`), one row for each species and one column for each coefficient in `frmla` (not `frmla.phylo`), projects the columns corresponding to `PartitioningVariables` onto contrast basis elements from `ilrvec`, and then picks the edge that maximizes the norm of this projection.

Because `CoefContrast` performs a single regression for each species (and not a single regression for each edge, repeated again for every factor), it is by far the fastest algorithm here and, as it relies on matrix multiplication, it can often be faster without parallelization. However, because it does not explicitly find maximum likelihood solutions for the model with group-specific meta-data associations, it may not be as accurate as `phylo`.

```
pf_gpf_coef <- gpf(PA_datatable,tree,frmla=cbind(Successes,Failures)~latitude,family=binomial,
               PartitioningVariables = 'latitude',algorithm='CoefContrast',nfactors=2)
```

```
pf_gpf_coef
```

```
##        phylofactor object from function gpf
##        ------------------------------------
## Method                    : gpf
## Algorithm                 : CoefContrast
##   Formula                   : cbind(Successes, Failures) ~ latitude
## Partitioning Variable     : latitude
## Number of species         : 100
## Number of factors         : 2
## Largest non-remainder bin : 48
## Number of singletons      : 0
## Paraphyletic Remainder    : 40 species
```

```
##
## ------------------------------------------------------------
## Factor Table:
##                                Group1                        Group2
## Factor 1 48 member Monophyletic clade 52 member Monophyletic clade
## Factor 2 12 member Monophyletic clade 40 member Paraphyletic clade
```

The `CoefContrast` works entirely with the coefficient matrix from species-specific models. These are output in the phylofactor object for `algorithm='CoefContrast'`. Below, I illustrate how these quantities are calculated from species-specific models. The coefficient matrix is simply the matrix of cofficients from the `species.models`:

```
coef(pf_gpf_coef$species.models[[10]])
```

```
## (Intercept)    latitude
##  0.90013866 -0.02415257
```

```
pf_gpf_coef$coefficient.matrix[10,]
```

```
## (Intercept)    latitude
##  0.90013866 -0.02415257
```

and the standard error is calculated with the function `vcov` for an input `glm` object:

```
sqrt(diag(vcov(pf_gpf_coef$species.models[[2]])))['latitude']
```

```
##   latitude
## 0.01174385
```

```
pf_gpf_coef$coefficient.SE[2]
```

```
## [1] 0.6329281
```

### algorithm='mix' (Default)

The algorithm `CoefContrast` is extremely fast but the post-hoc testing of coefficients underperforms the a priori model structure contrasting groups as implemented in the algorithms `mStable` and `phylo`. The default for `gpf` is a mixed algorithm - using `CoefContrast` to identify the top fraction, `alpha`, of the edges as candidates input to `algorithm='phylo'` factorization for each iteration.

For the mixed algorithm, you can input just `frmla.phylo` as the final decision on which edge to partition is made with `algorithm='phylo'`. The coefficient contrasts will obtain candidate edges that maximize the projection of all variables in `frmla.phylo` which interact with the term `phylo` (or, if you input `PartitioningVariables`, it will maximize the differences between `PartitioningVariables'` coefficients across groups).

```
pf_gpf_mix <- gpf(PA_datatable,tree,frmla.phylo = cbind(Successes,Failures)~phylo*latitude,
              family=binomial,nfactors=2,alpha = 0.1)
```

```
pf_gpf_mix
```

```
##        phylofactor object from function gpf
##        ------------------------------------
## Method                   : gpf
## Algorithm                : mix
## Formula                  : cbind(Successes, Failures) ~ phylo * latitude
## Partitioning Variable    : latitude
## Number of species        : 100
## Number of factors        : 2
```

```
## Largest non-remainder bin : 48
## Number of singletons      : 0
## Paraphyletic Remainder    : 40 species
##
## ------------------------------------------------------------
## Factor Table:
##                                 Group1                        Group2
## Factor 1 48 member Monophyletic clade 52 member Monophyletic clade
## Factor 2 12 member Monophyletic clade 40 member Paraphyletic clade
```

The mixed algorithm's phylofactor object indicates the nuances of the method:

```
names(pf_gpf_mix)[5:10]
```

```
## [1] "frmla"              "frmla.phylo"        "species.models"
## [4] "coefficient.matrix" "coefficient.SE"     "models"
```

Output from `algorithm='mix'` contains the output from the two methods it mixes, including coefficient matrices from `CoefContrast` and the models from the `phylo` algorithm.

The alternative algorithms and other input parameters are presented in the Ecological Monograph. For reference, the table of the different algorithms, their brief description, required input, unique optional input, pros and cons is provided above.

# Summary Tools

Some functions are available to quickly and easily obtain summary info from phylofactor objects. Many functions that take phylofactor objects as input start with `pf.`. The most commonly used functions are summarized here, the rest all have help files to indicate which `phylofactor.fcn` they work for. The functions `pf.summary`, `pf.predict` and `pf.plot` will soon be replaced by simpler (and more helpful) functions `summary`, `predict`, and `plot`.

## pf.taxa

Which taxa were split? The taxonomy may not be as well resolved as a phylogeny, and consequently there may be groups that are summarized without a single taxonomic name. To resolve this issue, `pf.taxa` produces a minimal set of shortest unique taxonomic prefixes for each group.

```
pf.taxa(pf_twoSample,taxonomy,factor=1)
```

```
## $group1
## [1] "k__Bacteria; p__Firmicutes; c__Clostridia; o__Clostridiales; f__Veillonellaceae"
## [2] "k__Bacteria; p__Actinobacteria"
## [3] "k__Bacteria; p__Proteobacteria"
## [4] "k__Bacteria; p__Bacteroidetes"
## [5] "k__Bacteria; p__Firmicutes; c__Erysipelotrichi"
## [6] "k__Bacteria; p__Firmicutes; c__Bacilli"
##
## $group2
## [1] "k__Bacteria; p__Firmicutes; c__Clostridia; o__Clostridiales; f__Lachnospiraceae"
## [2] "k__Bacteria; p__Firmicutes; c__Clostridia; o__Clostridiales; f__Ruminococcaceae"
## [3] "k__Bacteria; p__Firmicutes; c__Clostridia; o__Clostridiales; f__"
## [4] "k__Bacteria; p__Firmicutes; c__Clostridia; o__Clostridiales; f__Christensenellaceae"
## [5] "k__Bacteria; p__Fusobacteria"
```

```
## [6] "k__Bacteria; p__Firmicutes; c__Clostridia; o__Clostridiales; f__Clostridiaceae"
## [7] "k__Bacteria; p__Synergistetes"
## [8] "k__Bacteria; p__Firmicutes; c__Clostridia; o__Clostridiales; f__[Tissierellaceae]"
```

The phyla p_Bacteroidetes, p_Proteobacteria, and p_Actinobacteria are unique to group 1 and the phyla Fusobacteria and Synergistetes are unique to group2. Since Firmicutes is found in both groups, finer resolution is needed to obtain unique prefixes. The Firmicutes classes c_Erysipelotrichi and c_Bacilli are unique to group1. The family f_Veillonaceae is unique to group1, and so on.

The shortest-unique-taxonomic-prefix summarization allows one to flexibly describe taxonomic composition of groups.

### summary

Thee function `summary` can incorporate the taxonomy to summarize information from a single factor based on the signal (the objective function) of individual taxa contrasted with the complementary group

```
s <- summary(pf_gpf_coef,taxonomy,factor=1)
s
```

```
##        phylofactor object from function gpf
##        ------------------------------------
## Method                    : gpf
## Algorithm                 : CoefContrast
## Formula                   : cbind(Successes, Failures) ~ latitude
## Factor                    : 1
## Edges Considered          : 197
##
##                                Group1                        Group2
## Factor 1 48 member Monophyletic clade 52 member Monophyletic clade
## ========================================================================
## Taxon Tables:
## Group1:
##                                    Taxon nSpecies   signal
## 1          k__Bacteria; p__Bacteroidetes       15 14.19835
## 2         k__Bacteria; p__Proteobacteria       15 14.09330
## 3 k__Bacteria; p__Firmicutes; c__Bacilli       10 11.92816
##                     .................  3 rows omitted .................
## ------------------------------------------------------------------------
## Group2:
##                                                                                Taxon
## 1 k__Bacteria; p__Firmicutes; c__Clostridia; o__Clostridiales; f__Lachnospiraceae
## 2               k__Bacteria; p__Firmicutes; c__Clostridia; o__Clostridiales; f__
## 3 k__Bacteria; p__Firmicutes; c__Clostridia; o__Clostridiales; f__Ruminococcaceae
##   nSpecies   signal
## 1       22 18.79354
## 2        7 10.53051
## 3       11  9.47026
##                     .................  5 rows omitted .................
```

The default for `summary`, provided a taxonomy, is to find the shortest-unique-prefix labels for each species in each group. The `signal` in the taxon tables is the value of the objective function for a group-pair consisting of taxa fitting that label and the complementary group. For example, there are no Bacteroidetes in Group2, so the Bacteroidetes in Group1, of which there are 15 species, are contrasted with the entirety of Group2. The objective function from a coefficient contrast of those two groups is reported in the `signal`, allowing

users to see the relative importance of taxa for a phylogenetic factor of interest.

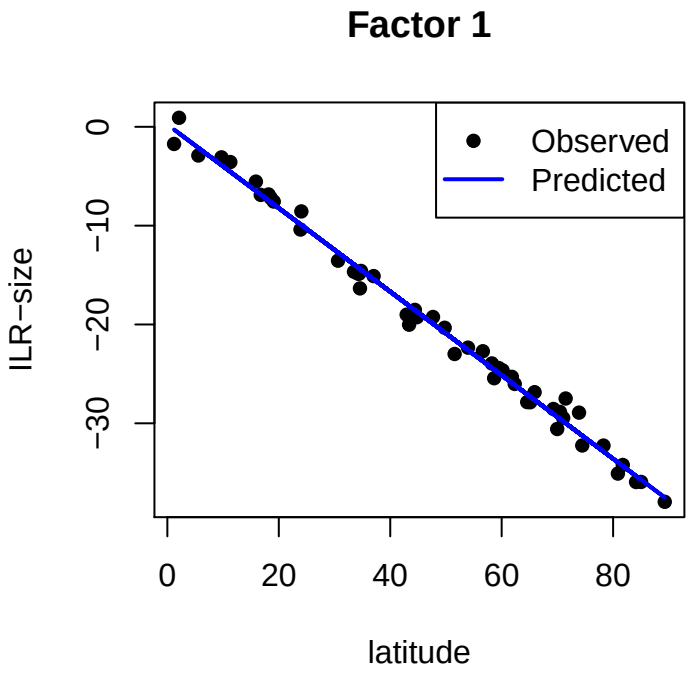
It's also possible to trim the values at a given taxonomic level by inputting a coarse taxonomy. For example, we can create a new taxonomy that bins species at the level of the class and input it into `summary` along with the input `taxon.trimming='taxon'`.

```
tx <- taxonomy$taxonomy %>%
        strsplit(';') %>%
       sapply(FUN=function(tx) paste(tx[1:3],collapse=';'))
family_taxonomy <- data.frame('Species'=taxonomy$OTU_ID,
                              'taxonomy'=tx,
                              stringsAsFactors = F)

s <- summary(pf_gpf_coef,family_taxonomy,1,taxon.trimming='taxon')
s
```

```
##        phylofactor object from function gpf
##        ------------------------------------
## Method                    : gpf
## Algorithm                 : CoefContrast
## Formula                   : cbind(Successes, Failures) ~ latitude
## Factor                    : 1
## Edges Considered          : 197
##
##                                 Group1                         Group2
## Factor 1 48 member Monophyletic clade 52 member Monophyletic clade
## ========================================================================
## Taxon Tables:
## Group1:
##                                                    Taxon nSpecies   signal
## 1          k__Bacteria; p__Bacteroidetes; c__Bacteroidia       14 13.80138
## 2                k__Bacteria; p__Firmicutes; c__Bacilli       10 11.92816
## 3 k__Bacteria; p__Proteobacteria; c__Gammaproteobacteria        9 11.60304
##                      .................  8 rows omitted .................
## ------------------------------------------------------------------------
## Group2:
##                                          Taxon nSpecies    signal
## 1      k__Bacteria; p__Firmicutes; c__Clostridia       48 20.805919
## 2 k__Bacteria; p__Fusobacteria; c__Fusobacteriia        3  4.563411
## 3  k__Bacteria; p__Synergistetes; c__Synergistia        1  3.140890
```

The summary function also provides useful `data`, with the precise output varying depending on the `phylofactor.fcn` input. For example, the `PhyloFactor` summary will provide the raw, contrast-projected data along with the meta-data.

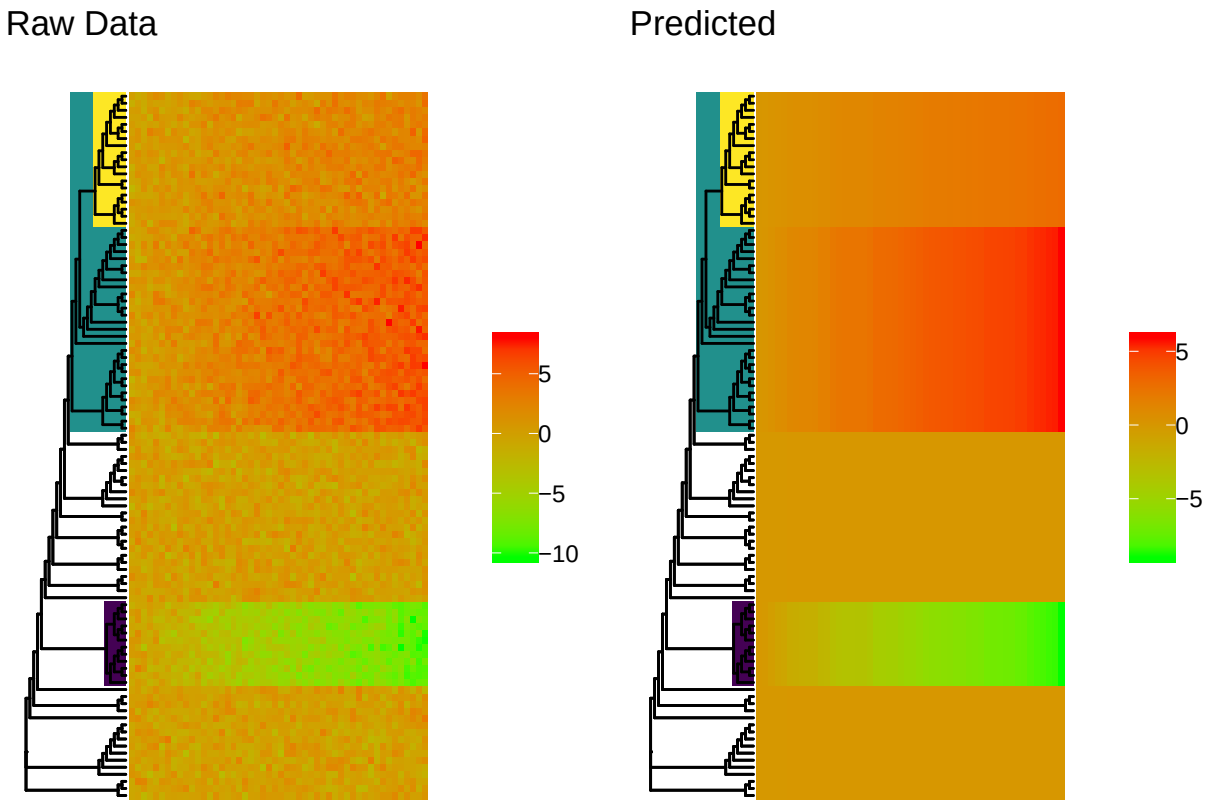
```
s_pf <- summary(pf_PhyloFactor,factor=1)

plot(s_pf$data$latitude,s_pf$data$Data,pch=16,xlab='latitude',ylab='ILR-size',main='Factor 1')
lines(s_pf$data$latitude,s_pf$data$fitted.values,col='blue',lwd=2)
legend('topright',legend=c('Observed','Predicted'),lwd=c(NA,2),pch=c(16,NA),col=c('black','blue'))
```




### predict

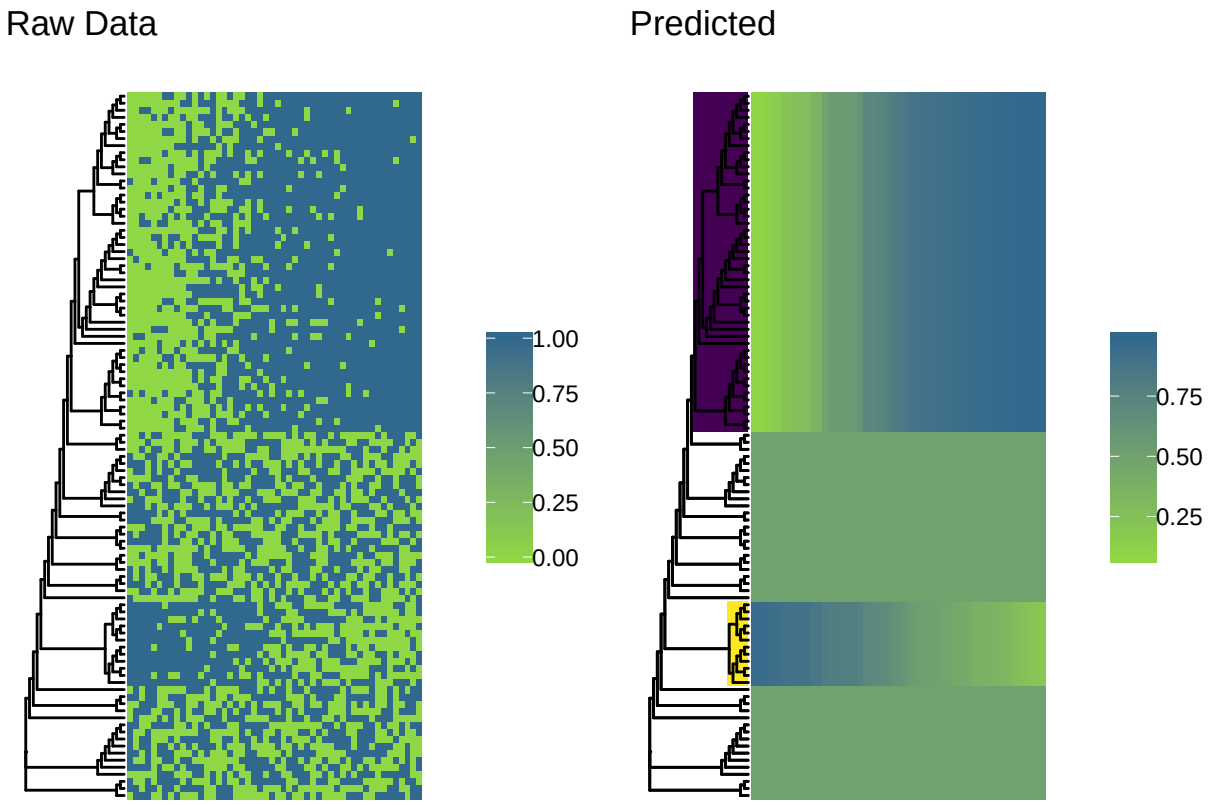
The function `predict` can predict entire data matrices for `PhyloFactor` output and `gpf` output for `algorithm='mStable'`. For other `gpf` algorithms, `predict()` will output predictions for the original phylo-factored `Data` or optional input `newdata` (`newdata` must be a data frame with a column `Species` and all the relevant data for `predict.glm` or `predict.gam`).

```
preds <- predict(pf_PhyloFactor)
g.orig <- pf.heatmap(pf_PhyloFactor,factors=1:3,
                     column.order=order(pf_PhyloFactor$X$latitude),
                     width=3)+
  ggplot2::ggtitle('Raw Data')
g.pred <- pf.heatmap(pf_PhyloFactor,factors=1:3,Data=preds,
                     column.order=order(pf_PhyloFactor$X$latitude),
                     width=3)+
  ggplot2::ggtitle('Predicted')

ggpubr::ggarrange(g.orig,g.pred,ncol=2)
```




Likewise, it will predict values for the `gpf algorithm='mStable'` phylofactor object

```
preds <- predict(pf_gpf_mStable,type='response')
cols=viridis::viridis(7)[c(3,6)] #colorblind-friendly colors

g.orig <- pf.heatmap(pf_gpf_mStable,
                     column.order=order(pf_gpf_mStable$MetaData$latitude),
                     width=3,low=cols[2],high=cols[1])+ggplot2::ggtitle('Raw Data')
g.pred <- pf.heatmap(pf_gpf_mStable,factors=1:2,Data=preds,
                     column.order=order(pf_gpf_mStable$MetaData$latitude),
                     width=3,low=cols[2],high=cols[1])+ggplot2::ggtitle('Predicted')

ggpubr::ggarrange(g.orig,g.pred,ncol=2)
```




# Visualization

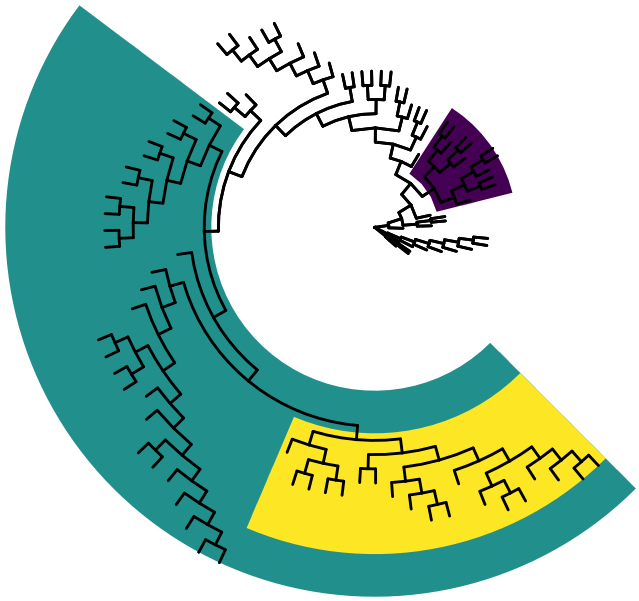
As shown above, phylofactor objects can be visualized to produce plots that combine phylogenies and meta-data associations.

## pf.tree

`pf.tree` highlights factored clades on a plotted phylogeny. It is a wrapper for `ggtree`, and outputs a `ggtree` object that can be annotated and a legend indicating the colors used for the factors in the `ggtree` object. Be sure to cite the ggtree package if you use these figures in a paper.

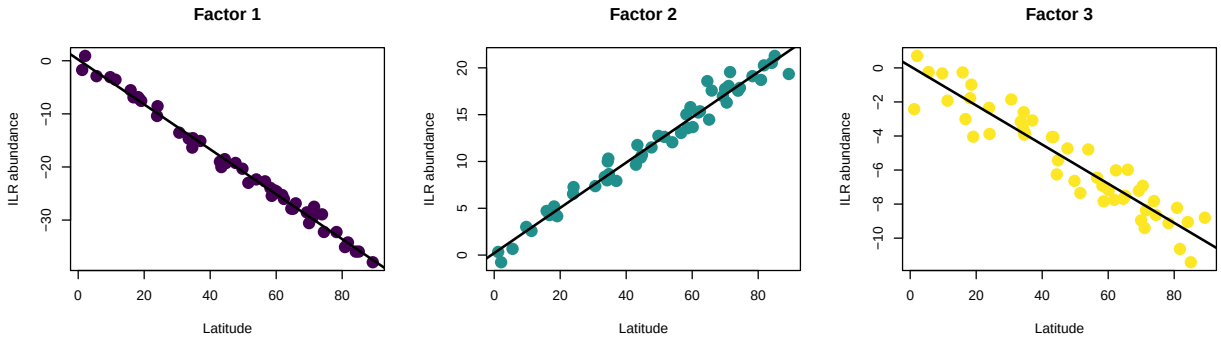
```
gtree <- pf.tree(pf_PhyloFactor)
ggtree::rotate_tree(gtree$ggplot,-45)
```

```
## Coordinate system already present. Adding new coordinate system, which will replace the existing one
```



The legend can be used to make complimentary plots that match the information in the tree.

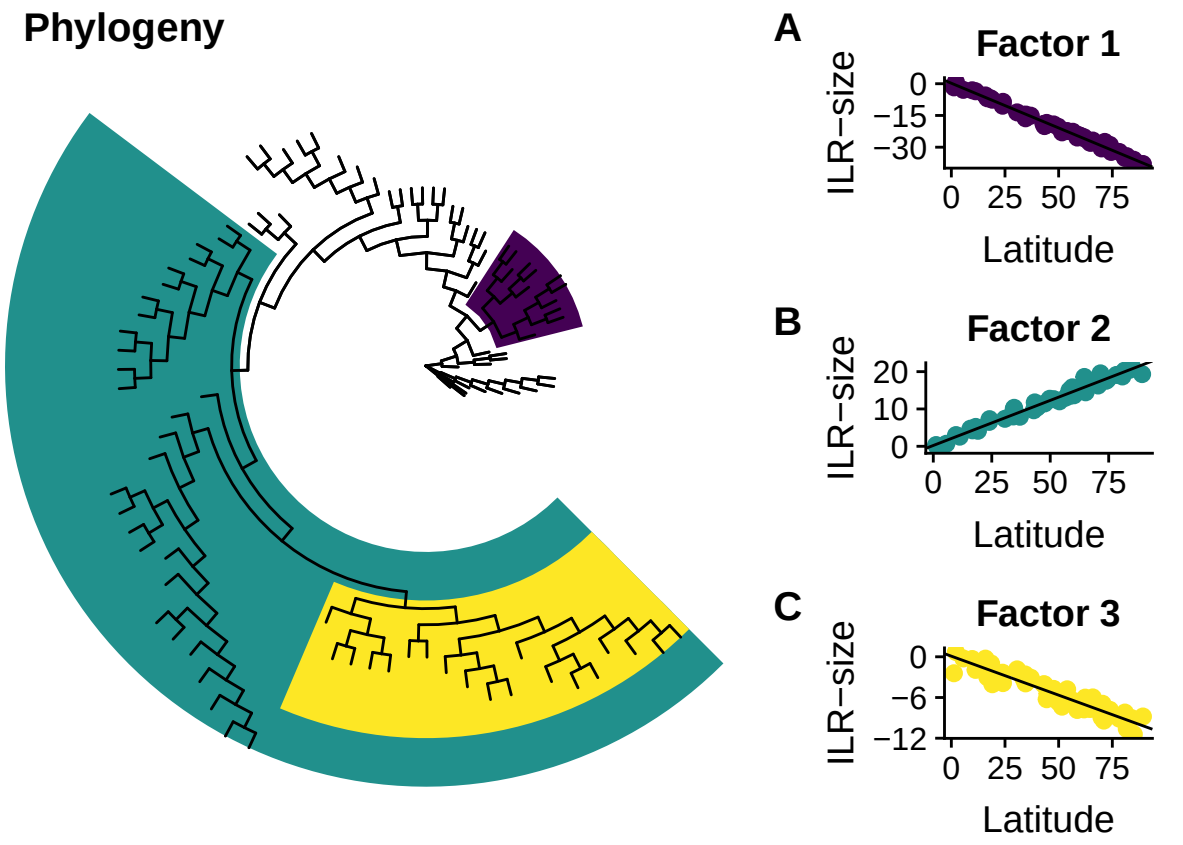
```
par(mfrow=c(1,3))
x <- pf_PhyloFactor$X$latitude
for (i in 1:3){
  y <- t(pf_PhyloFactor$basis[,i]) %*% log(pf_PhyloFactor$Data)
  plot(x,y,col=gtree$legend$colors[i],pch=16,cex=2,
       xlab='Latitude',ylab='ILR abundance',main=paste('Factor',i))
  abline(coef(pf_PhyloFactor$models[[i]]),lwd=2,col='black')
}
```




All of these plots, and information in the models, can be combined for informative, publication-quality figures.

The libraries `ggplot2`, `ggtree`, and `cowplot` can be used to make a figure showing the phylogenetic partitions (highlighting group 1 for each factor) along with the ILR-BodySizes for those factors.

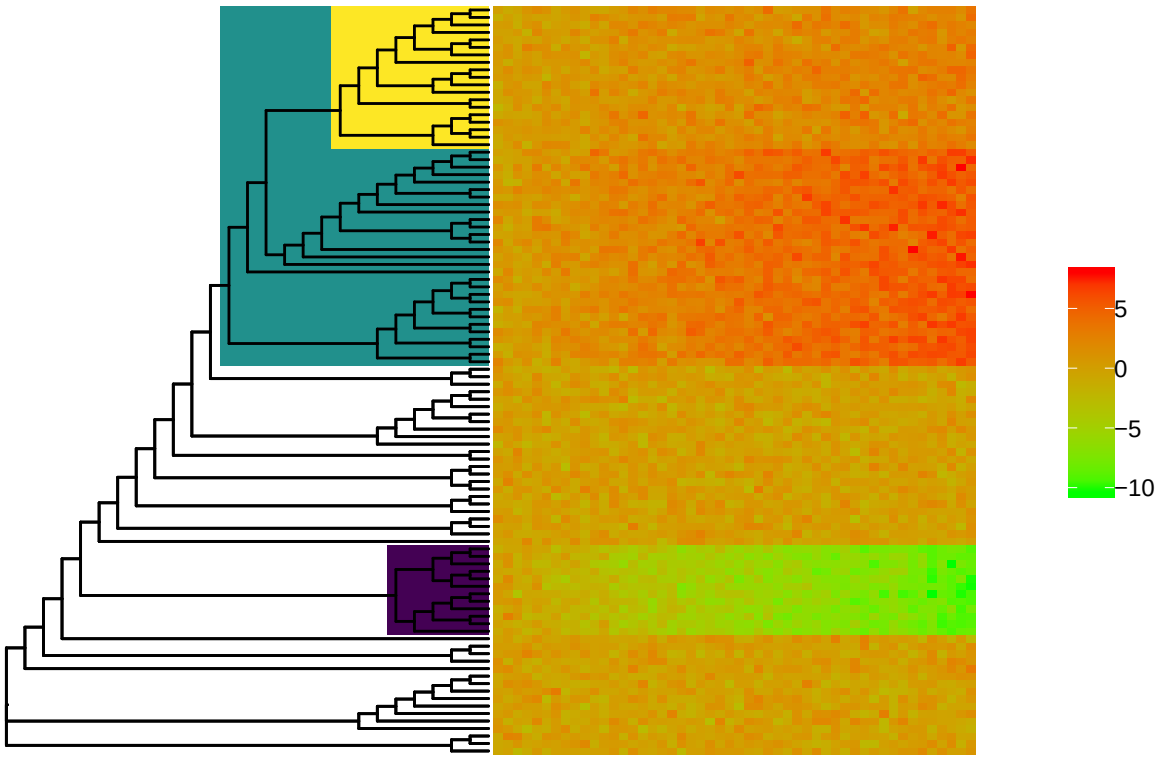
```
library(ggplot2)
library(ggtree)
library(cowplot)

i=1
y <- t(t(pf_PhyloFactor$basis[,i]) %*% log(pf_PhyloFactor$Data))
b <- coef(pf_PhyloFactor$models[[i]])[1]
a <- coef(pf_PhyloFactor$models[[i]])[2]
D <- data.frame('ILR_Abundance'=y,'Latitude'=x)
g1 <- ggplot(D,aes(Latitude,ILR_Abundance))+
  geom_point(size=2.5,color=gtree$legend$colors[i])+
  geom_abline(slope=a,intercept = b)+
  ggtitle(paste('Factor',i))+
  scale_y_continuous('ILR-size',breaks=c(0,-15,-30))

i=2
y <- t(t(pf_PhyloFactor$basis[,i]) %*% log(pf_PhyloFactor$Data))
b <- coef(pf_PhyloFactor$models[[i]])[1]
a <- coef(pf_PhyloFactor$models[[i]])[2]
D <- data.frame('ILR_Abundance'=y,'Latitude'=x)
g2 <- ggplot(D,aes(Latitude,ILR_Abundance))+
  geom_point(size=2.5,color=gtree$legend$colors[i])+
  geom_abline(slope=a,intercept = b)+
  ggtitle(paste('Factor',i))+
  scale_y_continuous('ILR-size',breaks=c(0,10,20))

i=3
y <- t(t(pf_PhyloFactor$basis[,i]) %*% log(pf_PhyloFactor$Data))
b <- coef(pf_PhyloFactor$models[[i]])[1]
a <- coef(pf_PhyloFactor$models[[i]])[2]
D <- data.frame('ILR_Abundance'=y,'Latitude'=x)
g3 <- ggplot(D,aes(Latitude,ILR_Abundance))+
  geom_point(size=2.5,color=gtree$legend$colors[i])+
  geom_abline(slope=a,intercept = b)+
  ggtitle(paste('Factor',i))+
  scale_y_continuous('ILR-size',breaks = c(0,-6,-12))

tr <- gtree$ggplot %>%
        rotate_tree(angle=-45)

ggdraw() +
  draw_plot(tr,
            x = 0.05, y = 0.22, width = .7, height = .7,scale = 1.8) +
  draw_plot(g1, x = .7, y = .66, width = .3, height = .33) +
  draw_plot(g2, x = .7, y = .33, width = .3, height = .33) +
  draw_plot(g3, x = .7, y = 0, width = .3, height = .33) +
  draw_plot_label(label = c("Phylogeny", "A", "B","C"), size = 15,
                  x = c(0, 0.67, .67,.67), y = c(1, 1, 0.66,0.33))
```




### pf.heatmap

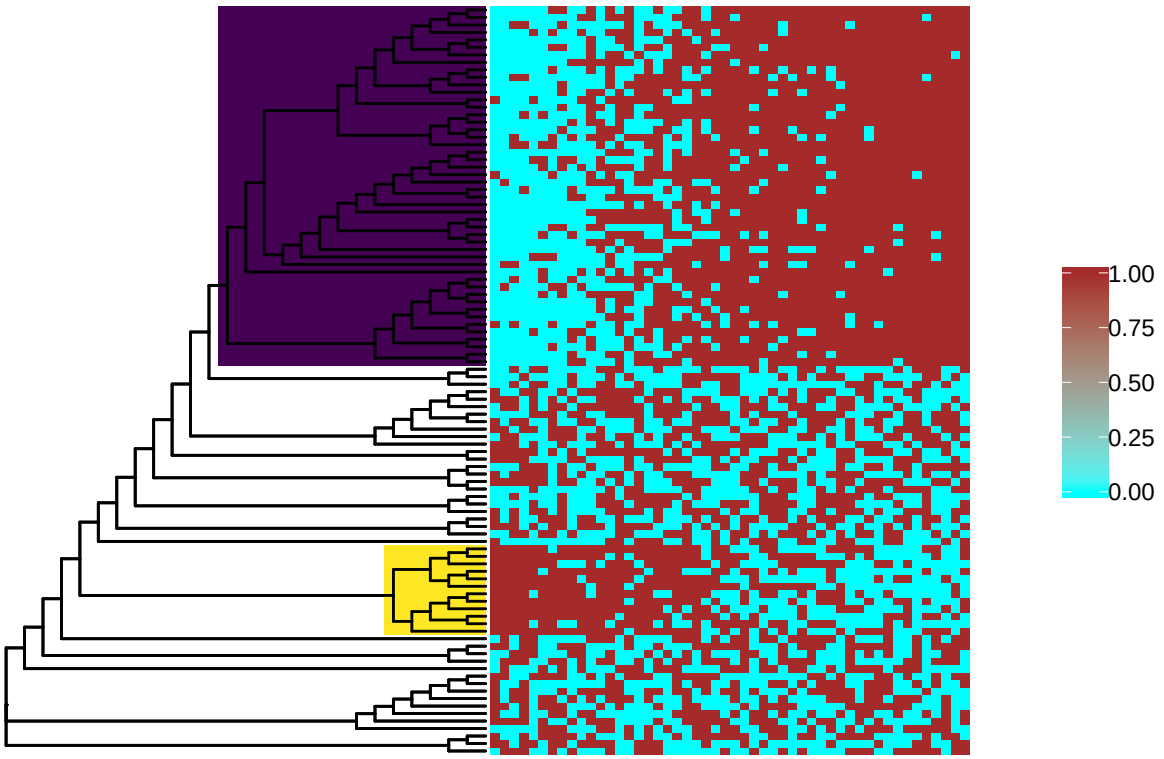
Another useful function is `pf.heatmap`, which can flexibly produce heatmaps illustrating phylogenetic factors along with phylogenetic patterns in the data. Heatmaps can be generated directly from phylofactor objects output from `PhyloFactor` or `gpf` with `algorithm='mStable'`. Alternatively, the `pf.heatmap` can take as input a new data (e.g. predicted data). The heatmap is a wrapper for the gheatmap from ggtree, so be sure to cite the ggtree package if you publish these figures..

```
pf.heatmap(pf_PhyloFactor,factors=1:3,column.order = order(MetaData$latitude))
```




And likewise for our `mStable` output:

```
cols=c('cyan','brown')
pf.heatmap(pf_gpf_mStable,factors=1:2,
           column.order=order(pf_gpf_mStable$MetaData$latitude),
           low=cols[1],high=cols[2])
```




## Phylofactor Objects

The common algorithm in phylofactorization warrants common, intuitive output from the various phylofactor functions, `twoSampleFactor`, `PhyloFactor`, `PhyCA` and `gpf`. These functions all returns `phylofactor` class objects that can allow the familiar user greater freedom in summary, visualization and downstream analysis.

In this section, we'll explain the common structure of phylofactor objects and the unique features output from the different methods. We'll start by looking at the common names:

```
names_twoSample <- names(pf_twoSample)
names_pf <- names(pf_PhyloFactor)
names_phyca <- names(phyca)
names_gpf <- names(pf_gpf_mStable)
common_names <- Reduce(intersect,list(names_twoSample,names_pf,names_phyca,names_gpf))
common_names
```

```
## [1] "groups"          "factors"         "basis"           "bins"
## [5] "tree"            "nfactors"        "Data"            "method"
## [9] "phylofactor.fcn"
```

The table of `factors` gives a summary of the object and groups vs. bins are covered in this table. The `bins` of the phylofactor object are the resultant bins from the entire phylofactorization. Intermediate bins can be obtained with the functions `bins` or the data can be projected onto bins via the function `pf.BINprojection`. The `basis` is a matrix whose columns are basis elements made with `ilrvec`. The rest should be self-explanatory.

Below, we discuss some unique features of the `twoSampleFactor` and `PhyloFactor` output. Objects from `PhyCA` and `gpf` have similar elements to `PhyloFactor`.

### twoSampleFactor objects

```
names(pf_twoSample)
```

```
##  [1] "pvals"               "objective"           "groups"
##  [4] "factors"             "basis"               "bins"
##  [7] "tree"                "nfactors"            "Data"
## [10] "model.fcn"           "method"              "additional.arguments"
## [13] "phylofactor.fcn"
```

1) "pvals": uncorrected P-values from the winning edge's two-sample test.
2) "objective": the raw objective statistic (in this case, the t-statistic for equal variances)
3) "model.fcn": the function used for contrasting groups
4) "method": This will always be "twoSample"
5) "choice": The particular choice function

### PhyloFactor objects

```
names(pf_PhyloFactor)
```

```
##  [1] "Data"               "X"                  "tree"
##  [4] "models"             "choice"             "choice.fcn"
##  [7] "cluster.depends"    "method"             "transform.fcn"
## [10] "additional.args"    "contrast.fcn"       "terminated"
## [13] "groups"             "factors"            "basis"
## [16] "nfactors"           "bins"               "bin.sizes"
## [19] "Monophyletic.clades" "algorithm"         "phylofactor.fcn"
```

1) "Data": Data matrix used for phylofactorization. This matrix may differ from the input OTU table, as `PhyloFactor` trims rows that are not in the tree and, if `transform.fcn=log`, it will replace zeros with 0.65, and converts these sequence counts to relative abundances. If you have a preferred method of dealing with zeros it's necessary to do that before inputting the data into `PhyloFactor`.
2) "X": the independent variable or data frame of independent variables.
3) "models": For the default regression-phylofactorization, `PhyloFactor` outputs the `"glm"` objects from the factored edges. For customized phylofactorization in `PhyloFactor`, the element "models" will be null and a new element `customized.output` can be returned for the factored edges.
4) "method": Method - glm, gam or custom.
5) "transform.fcn" - the function used to transform the data prior to projection onto contrast basis elements. The default is `log`, hence our inputting the raw, log-normally distributed body sizes.
6) "terminated": logical, indicating whether or not the iterations were terminated due to stopping criteria.
7) "basis": The `ilrvec` balancing elements constructed from phylofactorization.
8) "nfactors": the number of factors.
9) "bins": the resultant bins at the end of the phylofactorization.

10) "bin.sizes": breakdown of bin sizes remaining at the end of phylofactorization. Includes the number of bins with that size.

11) "Monophyletic.clades": indexes of which bins are monophyletic.

**P-values: Currently not well-calibrated.**

One point that is essential to state and reiterate is that the statistics summarized in the `$factors` element are the raw statistics from the models. For `PhyloFactor` output, for example, the F statistic drawn is the best of many.

```
pf_PhyloFactor$factors[,c('F','Pr(>F)')]
```

```
##                   F Pr(>F)
## Factor 1 5615.7960      0
## Factor 2 1716.3536      0
## Factor 3  371.9624      0
```

In this examle, the choice function `choice='F'` maximized the F-statistic. The null distribution for the first factor's F-statistic is approximately the distribution of the maximum of E independent, identically distributed F-statistics where E is the number of edges in the phylogeny (for m species, a fully resolved phylogeny has 2m-3 edges). The order statistics from the F distribution do not appear to be well-calibrated null distributions for downstream F-statistics. The `Pr(>F)` is the p-value of the raw F-statistic, not the p-value for the phylofactor object. It's reported here so one can use bonferroni and other `p.adjust` style cutoffs to estimate the significance of factors. To implement Holm's sequentially rejective p-value cutoffs, the number of distinct contrasted groups, E(k), considered at factor k is approximately equal to `seq(2*m-3,1,by=-2)[k]`.

# Advanced Controls

A few advanced features allow the user greater control over phylofactorization.

## stop.fcn

One important question shared among phylofactorization and other reduced-rank approximation methods (e.g. factor analysis, PCA, reduced-rank regression, etc.) is how factors to iterate. While null simulations can provide definitive statements on the null distributions of the objective function, null simulations can be computationally infeasible.

Another alternative is to stop phylofactorization when the set of objective functions appears to have little signal, either by matching a null distribution or by having P-values which are uniformly distributed or satisfiy a desired family-wise error rate or false-discovery rate threshold. These stopping functions can be employed with the input `stop.fcn`.

```
pf_stop <- PhyloFactor(BodySize,tree,MetaData,frmla=Data~latitude,choice='F',stop.early=T)
pf_stop$factors
```

Notice we did not have to input the number of factors, yet it correctly chose three factors. The built-in stopping function is based on a one-sided Kolmogorov-Smirnov test of the uniformity of the distribution of P-values from F-tests of the edges' regression models at each iteration. If the distribution of P-values across edges is uniform, there is little reason to believe that the "best" edge is significant.

Other stopping functions can be created and even customized. For example, instead of doing a KS-test, we could use a Holm, sequentially-rejective test. This can be implemented by passing a `stop.fcn` function to

PhyloFactor (stopping functions will come soon to gpf and twoSampleFactor, and have the same format as PhyloFactor).

The stop.fcn takes as input arguments a list of stopStatistics and outputs a logical - TRUE will stop phylofactorization and FALSE will continue iteration. In PhyloFactor, the default for stop statistics are P-values whose uniformity can be checked. Below, instead of testing the uniformity, we'll check whether the minimum P-value falls below Holm's sequentially-rejective cutoff for a 5% family-wise error rate.

```
holm <- function(P){
  P <- unlist(P)    ### must unlist the input;
  # The input is a list to permit more flexible analyses requiring multiple statistics per edge
  P <- stats::p.adjust(P,method='holm')
  if (min(P)>0.05){
    return(TRUE)
  } else{
    return(FALSE)
  }
}

pf_holm <- PhyloFactor(BodySize,tree,MetaData,
                       frmla=Data~latitude,
                       choice='F',
                       stop.early=T,
                       stop.fcn=holm)
```

```
## Warning in any(Data): coercing argument of type 'double' to logical
```

Notice our Holm stopping function also produced the right number of factors:

```
pf_holm
```

```
##        phylofactor object from function PhyloFactor
##        --------------------------------------------
## Method                    : glm
## Choice                    : F
## Formula                   : Data ~ latitude
## Number of species         : 100
## Number of factors         : 3
## Largest non-remainder bin : 48
## Number of singletons      : 0
## Paraphyletic Remainder    : 40 species
##
## ----------------------------------------------------------------
## Factor Table:
##                                Group1                         Group2       F
## Factor 1 12 member Monophyletic clade 88 member Monophyletic clade 5615.80
## Factor 2 48 member Monophyletic clade 40 member Paraphyletic clade 1716.40
## Factor 3 19 member Monophyletic clade 29 member Paraphyletic clade  371.96
##          Pr(>F)
## Factor 1      0
## Factor 2      0
## Factor 3      0
```

## Objective Functions

Objective functions can also be customized for the functions `PhyloFactor`, `twoSampleFactor`, and `gpf`. In `? PhyloFactor`, see the information about the input `choice.fcn` and the example for a customized, generalized additive model. For `twoSampleFactor`, you can input your own `TestFunction` (details also provided in `? twoSampleFactor`). Likewise, for `gpf` algorithms `phylo` and `mStable`, you can use your own `model.fcn` and `objective.fcn`. Data frames containing the `phylo` factor will be input as the argument `"data"` to your `model.fcn`, along with `formula=frmla.phylo` and the additional arguments `....` This topic can be exhaustive, and the help files of each function indicate precisely how to customize the objectives.

Here, we'll make one customized objective function for each phylofactor function, illustrating the general input/output requirements of customized functions.

### Customized twoSampleFactor

The function `twoSampleFactor` allows an input `TestFunction`. Below, we'll partition body masses usin ga Mood test of the equality of the medians of two groups. It will depend on the function `Mood.test` which we pass to the clusters as a character

```
mtest <- 'Mood.test <- function(x, y){
      z <- c(x, y)
      g <- rep(1:2, c(length(x), length(y)))
      m <- median(z)
      return(fisher.test(z < m, g)$p.value)
    }'
```

The clusters will all have the expression from `mtest` evaluated, defining the `Mood.test` variable locally for use in our input `TestFunction`. A customized `TestFunction` takes as input

1) `grps` a two-member list containing the indexes for Group1 and Group2
2) `Z` our input data
3) `PF.output` logical. When true, the output will be stored in the `pvals` element of the phylofactor object.

The output when `PF.output=FALSE` should be the objective function whose maximum value determines the edge to partition.

```
my.test.fcn <- function(grps,Z,PF.output=F,...){
  if (!PF.output){
    return(1/Mood.test(Z[grps[[1]]],Z[grps[[2]]]))
  } else {
    return(median(Z[grps[[1]]])-median(Z[grps[[2]]]))
  }
}
```

Our `mood.test` and customized `TestFunction` can be used to factor `logBodySize`.

```
pf_twoSample_Custom <- twoSampleFactor(Z=logBodySize,
                                       tree=pf_twoSample$tree,
                                       nfactors = 2,
                                       TestFunction = my.test.fcn,
                                       cluster.depends=mtest,
                                       ncores=2)
```

We can display the factors along with the difference in medians:

```
cbind(pf_twoSample_Custom$factors,'median_difference'=pf_twoSample_Custom$pvals)
```

```
##                                  Group1                        Group2
```

```
## Factor 1 48 member Monophyletic clade 52 member Monophyletic clade
## Factor 2 12 member Monophyletic clade 40 member Paraphyletic clade
##          median_difference
## Factor 1          1.521141
## Factor 2         -2.199332
```

### Customized PhyloFactor

To customize `PhyloFactor`, one needs to input a `choice.fcn` and `cluster.depends`. We'll define a generalized additive model fitting a smoothing spline of BodySize~Latitude. The objective function will be the F-statistic from an anova. Using gam from the package `mgcv` will require specifying `cluster.depends` to load `library(mgcv)`. The customized gam will require defining a `choice.fcn` with required input:

1) `y` the projection of the transformed data through `contrast.fcn`.
2) `X` the meta-data
3) `PF.output` sames as above

The customized `choice.fcn` must output a list containing the element `objective` and can optionally return a `stopStatistics` for input into the `stop.fcn`. We'll use our `holm` stopping function defined above.

```
load.mgcv <- 'library(mgcv)'
my.gam <- function(y,X,PF.output=F,...){
  dataset <- cbind('BodySize'=y,X)

  fit <- gam(formula=BodySize~s(latitude),
             data=dataset,...)
  output <- NULL
  if (PF.output){
    ## output the model
    output <- fit
  } else {
    a <- anova(fit)
    output$objective <- a$s.table[1,'F']
    output$stopStatistics <- a$s.table[1,'p-value']
  }
  return(output)
}
```

All of this can be input into `PhyloFactor` and the `stop.fcn` can be used. The use of `...` allows us to input a smoothing penalty into our gam.

```
pf_PhyloFactor_Custom <- PhyloFactor(BodySize,tree,MetaData,
                                     choice.fcn=my.gam,
                                     cluster.depends = load.mgcv,
                                     stop.fcn=holm,
                                     stop.early=T,
                                     sp=10,
                                     ncores=2)
```

```
## Loading required package: nlme
```

```
##
## Attaching package: 'nlme'
```

```
## The following object is masked from 'package:ggtree':
##
##     collapse
```

```
## This is mgcv 1.8-26. For overview type 'help("mgcv-package")'.
## Warning in any(Data): coercing argument of type 'double' to logical
```

### Customized gpf

The function `gpf` is customized through `model.fcn` and `objective.fcn`. You can see how these two functions are called in the function `getObjective`.

One topic of interest is the use a generalized additive model in `gpf`. For smoothing functions, the syntax `s(x,by=phylo)` can be used to create a smoothing spline that is different for each group. `model.fcn` must take as input `formula`, `data`, and optional input arguments in `....`

```
pf_gam <- gpf(PA_Data,tree,frmla.phylo=cbind(Successes,Failures)~s(latitude,by=phylo),
              MetaData=MetaData,model.fcn=mgcv::gam,nfactors=2,
              algorithm='mStable',family=binomial)
```

The above `gpf` call uses the default objective function, `pvDeviance`, which attempts to parse out the deviance for a model with vs. a model without the `phylo` term. To be more explicit, one can define their own `objective.fcn`. The `objective.fcn` must take as input

1) `fit` - the output from `model.fcn`
2) `grp` - the two-member list containing Group1 (`phylo==R`) and Group2 (`phylo==S`)
3) `tree` - input to `gpf`
4) `PartitioningVariables` - input to `gpf`
5) `model.fcn` - input to `gpf`
6) `phyloData` - data frame with a column, `phylo`, indicating which group a species is found in.
7) `...` - optional input arguments.

The objective function should output a numeric that increases monotonically with edge quality (i.e. the maximum objective will be chosen for edge-partitioning).

```
my.obj <- function(fit,grp,tree,PartitioningVariables,model.fcn,phyloData,...){
  fit2 <- model.fcn(cbind(Successes,Failures)~s(latitude),data=phyloData,family=binomial)
  return(deviance(fit2)-deviance(fit))
}
```

All of these can be input to a customized gam:

```
pf_gam2 <- gpf(PA_Data,tree,frmla.phylo=cbind(Successes,Failures)~s(latitude,by=phylo),
              MetaData=MetaData,model.fcn=mgcv::gam,nfactors=2,ncores=2,
              algorithm='mStable',family=binomial,objective.fcn=my.obj)
```

Customizing `gpf` allows the broadest variety of phylofactorizations. By setting `algorithm='phylo'`, the `objective.fcn` will have access to the object `phyloData` that contains the entire dataset as a `data.table` with species labeled by which side of the edge they're found. With such a dataset, one could implement `twoSampleFactor` by defining a two-sample test between `phylo==R` and `phylo==S` elements of `phyloData`, one could implement `PhyloFactor` by implementing the appropriate aggregation of data within `phylo` groups and contrast between `phylo` groups (e.g. the `BalanceContrast` function applied by sample x phylo), and one could perform more complicated phylogenetic comparative-style analyses with the input tree - hence, this algorithm `gpf` is "generalized" phylofactorization. Those `gpf` outputs will have access to many of the other machinery for visualizing and cross-validating phylofactor objects.

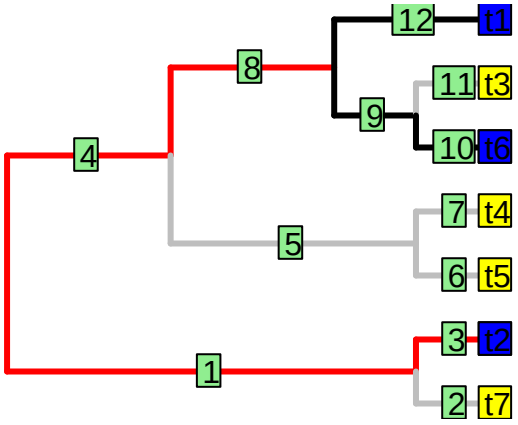
## Cross-Validation

Still in development are functions which will allow one to map phylogenetic factors from one dataset to another, even with disjoint sets of species, provided all the species can be mapped onto a common tree. One

function, `crossVmap`, accepts a single contrasted groups list, as in `pf$groups[[1]]`, and will indicate which species of the new community are in which group. Optionally, you can obtain the interruptions.

Below, we illustrate how `crossVmap` can be used

```
set.seed(1)
tree <- ape::rtree(7)
c1 <- c('t2','t1','t6')
c2 <- setdiff(tree$tip.label,c('t1'))

grp1 <- c('t1','t6')
grp2 <- c('t2')
Grps <- list(grp1,grp2)

factored.edges <- getFactoredEdges(grp1=match(grp1,tree$tip.label),
                                   grp2=match(grp2,tree$tip.label),tree=tree)

ecols <- rep('black',ape::Nedge(tree))
ecols[factored.edges] <- 'red'
ecols[c(2,5:7,11)] <- 'grey'
plot(tree,use.edge.length = FALSE,
            edge.width = 3,edge.color = ecols,show.tip.label = F)
tiplabels(c2,match(c2,tree$tip.label))
tiplabels(c1,match(c1,tree$tip.label),bg='blue')
edgelabels()
```




In the scenario above, the blue boxes indicate the original community and the new community consists of all species except `t1`. A factor is found separating species `{'t1','t6'}` from `{'t2'}` in community 1. The factored edges are in red. This contains one inclusion - species `t3` should be placed within the first factor - and two interruptions.

```
crossVmap(Grps,tree,original_community = c1,new_community = c2,ignore.interruptions = F)
```

```
## $`12`
## [1] "t6" "t3"
##
## $`10`
## [1] "t5" "t4"
##
## $`9`
## [1] "t7"
##
## $`2`
## [1] "t2"
```

The first and last elements of the output above are the unambiguous factors - notice that `t6` was correctly placed in the first factor and `t1` is no longer there as it is not in the new community. The interruptions are found in the intermediate elements in the order in which they are found when traversing from the two extreme groups. The interrupting nodes are the names of the list.

When one has an entire phylofactorization, it's possible to map factors from one community to another with the function `pf.crossValidateMap`. We won't simulate a phylofactorization, but instead construct a minimal phylofactor object to demonstrate this. In addition to the groups partitioned above, our phylofactor object will partition `t1` from `t6` in the second factor.

For this, we want the first factor to be mapped as it was with crossVmap. For the second factor in the new community, the tip `t3` will be an interruption and `t1` not exist.

```
p <- NULL
p$nfactors <- 2
p$tree <- drop.tip(tree,setdiff(tree$tip.label,c1))
p$groups <- list(list(c(1),c(2,3)),list(2,3))
pf.crossValidateMap(p,tree,c2)
```

```
## [[1]]
## [[1]]$`2`
## [1] "t2"
##
## [[1]]$`12`
## [1] "t6" "t3"
##
##
## [[2]]
## [[2]]$`1`
## [1] "t6"
##
## [[2]]$`3`
## character(0)
```

The second group of the second factor is empty. For large phylofactor objects, this can be problematic, and so the function can optionally `fill.empty` groups

```
cv <- pf.crossValidateMap(p,tree,c2,fill.empty=T)
cv$groups[[2]]
```

```
## $`1`
## [1] "t6"
##
```

```
## $`3`
## [1] "t1"
```

Now, the second group of the second factor has a species - `t1`, which is not present in the new community. These additions are noted in output object

```
cv$filler.species
```

```
##   factor group species
## 1      2     2      t1
```

indicating which factor-group combos received which filler species. This can be useful for tacking on empty observations to a data frame or data matrix.