dispRity demo

Thomas Guillerme

October 8, 2015

This is a quick demo to go through the beta version of the `dispRity` package (v.0.1.1). Many parts are still missing (see section 4) but this version should be running all right and give an idea of what's implemented in the package.

1 Before starting

`dispRity` is a package for calculating disparity in `R`. To keep it short, this package allows to summarise ordinated matrices (e.g. MDS, PCA, PCO, PCoA) into single values. These ordinated matrices can be called the morphospace (when using morphometric data) or the cladisto-space (for cladistic data) and will represent the total spread of the forms available in the dataset. However, keep in mind that if you are using other type of data, say ecological data for example, then the ordinated matrix will represent the eco-space which represents the total spread of ecological traits available in the dataset. In the rest of this demo, I will sometimes refer to the ordinated matrix as the any-o-space where you can replace *any* by any prefix you want your space to be based on!

1.1 Installation

You can install this package easily if you use the latest version of `R` and `devtools`.

```
install.packages("devtools")
library(devtools)
install_github("TGuillerme/dispRity", ref = "release")
library(dispRity)
```

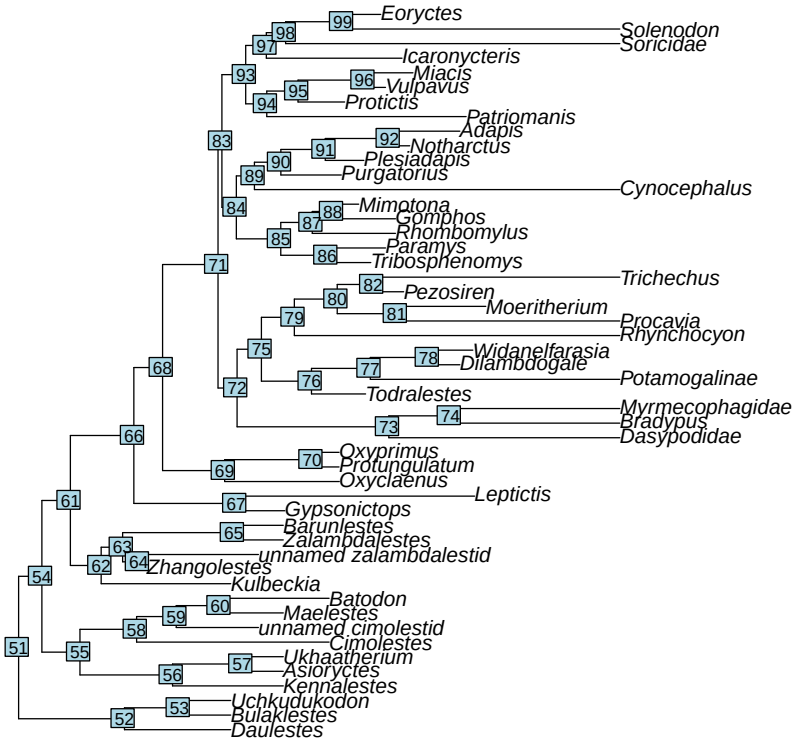
Note that we use the `release` branch here which is version 0.1.1. If you want the piping-hot (and full of bugs) version, change the option `ref = "release"` to `ref = "master"`. This package depends mainly on the `ape` package and the `timeSliceTree::paleotree` function. Future version will just require the specific function but this version (0.1.1) will install `paleotree` as a dependency (which is not that bad since it is a super useful package anyway!). Additionally, I warmly recommend to install the packages `geomorph` or/and `Claddis` if you want to try this demo with your very own ordinated morphometric or cladistic data.

1.2 Which data?

In this demo we are going to use a subset of the ordinated cladistic data from Beck and Lee 2014 [1]:

```
## Loading demo and the package data
library(dispRity)

## Loading required package:  paleotree
```

```
## An ordinated matrix with the tips only
data(BeckLee_mat50)
dim(BeckLee_mat50)

## [1] 50 48

head(BeckLee_mat50[,1:5])

##                    [,1]          [,2]        [,3]       [,4]      [,5]
## Cimolestes   -0.5319679  0.1117759259  0.09865194 -0.1933148 0.2035833
## Maelestes    -0.4087147  0.0139690317  0.26268300  0.2297096 0.1310953
## Batodon      -0.6923194  0.3308625215 -0.10175223 -0.1899656 0.1003108
## Bulaklestes  -0.6802291 -0.0134872777  0.11018009 -0.4103588 0.4326298
## Daulestes    -0.7386111  0.0009001369  0.12006449 -0.4978191 0.4741342
## Uchkudukodon -0.5105254 -0.2420633915  0.44170317 -0.1172972 0.3602273

## An ordinated matrix with the tips and the nodes (named nX)
data(BeckLee_mat99)
dim(BeckLee_mat99)

## [1] 99 97

head(BeckLee_mat99[,1:5], 2)

##                  [,1]       [,2]       [,3]       [,4]        [,5]
## Cimolestes -0.6082437 -0.0323683 0.08458885 -0.4338448 -0.30536875
## Maelestes  -0.5730206 -0.2840361 0.01308847 -0.1258848  0.06123611

tail(BeckLee_mat99[,1:5], 2)

##            [,1]      [,2]       [,3]       [,4]       [,5]
## n48 -0.05529018 0.4799330 0.04118477 0.04944912 -0.3558830
## n49 -0.13067785 0.4478168 0.11956268 0.13800340 -0.3222785

## A list of first and last occurrences data for some fossils
data(BeckLee_ages)
head(BeckLee_ages)

##             FAD  LAD
## Adapis     37.2 36.8
## Asioryctes 83.6 72.1
## Leptictis  33.9 33.3
## Miacis     49.0 46.7
## Mimotona   61.6 59.2
## Notharctus 50.2 47.0

## And a phylogeny:
data(BeckLee_tree)
plot(BeckLee_tree) ; nodelabels(cex=0.8)
```



However, I greatly encourage you to use your own data (after all, that's the point of an `R` package). You will simply need (1) an ordinated distance matrix: make sure that the number of column is at maximum the number of rows minus one. And secondly (2) a phylogenetic tree with the tips matching the ordinated matrix and a `$root.time` element for determining the age of the tree. You can use the function `tree.age` provided in this package but not described in this demo for generating the `$root.time` automatically (see `?tree.age`). In the following examples, just replace all the `BeckLee` mentions in this demo by your very own data:

```
## My ordinated matrix with tips only
MyData_mat50 <- BeckLee_mat50
## My ordinated matrix with tips and nodes
MyData_mat99 <- BeckLee_mat99
## My first and last occurence data
MyData_ages <- BeckLee_ages
## My phylogenetic tree
MyData_tree <- BeckLee_tree
```

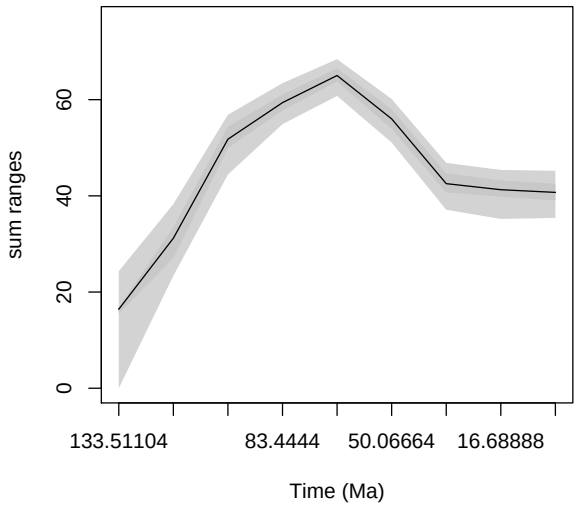
2 A super quick go through

Without much explanation, here's what this package allows to run:

1. Splitting the data

2. Bootstrapping the data
3. Calculating disparity
4. Summarising the results

```
## Splitting the data
sliced_data <- time.series(MyData_mat99, MyData_tree, method = "continuous",
    model = "acctran", time = 9, FADLAD = MyData_ages)

## Some tips have no FAD/LAD and are assumed to be single points in time.

## Bootstrapping the data
bootstrapped_data <- boot.matrix(sliced_data, 100)
## Calculating disparity
sum_of_ranges <- dispRity(bootstrapped_data, metric = c(sum, ranges))
## Summarising the results
summary(sum_of_ranges)

##      series  n observed  mean  2.5%   25%   75% 97.5%
## 1 133.51104  3    24.35 16.43  0.00 15.51 16.78 24.35
## 2 116.82216  6    40.85 31.19 23.38 27.09 33.51 38.29
## 3 100.13328 14    61.74 51.78 44.43 49.94 54.36 56.86
## 4   83.4444 19    69.86 59.38 54.91 57.76 61.06 63.43
## 5  66.75552 23    74.79 65.02 60.78 63.77 66.51 68.41
## 6  50.06664 17    65.09 56.01 51.08 53.95 58.02 60.15
## 7  33.37776 11    49.82 42.54 37.11 40.72 44.68 46.86
## 8  16.68888 10    47.62 41.28 35.20 39.85 43.20 45.38
## 9         0 10    47.62 40.72 35.42 39.05 42.53 45.21

plot(sum_of_ranges)
```



Et voilà, that's pretty much it...

3 In more details

In this section we are going to look at each functionality in more details. Of course, at this stage of the development of the package, any suggestion is most welcome. Please create an issue on the GitHub page: github.com/TGuillerme/dispRity or even just drop me an email at guillert@tcd.ie.

3.1 The `dispRity` objects

Disparity analysis involve a lot of shuffling around with many matrices (especially when bootstrapping the data) which can be a bit impractical to visualise and quickly jam your `R` console. For example, try to use `str` to show the structure of the object created above:

```
str(sum_of_ranges)
## That's a more than 4000 lines of output
```

Therefore this package proposes a new class of object called `dispRity` (how unexpected!) that allow to use a S3 `print.dispRity` method. In other words, when you are creating objects using the `dispRity` functions, you can quickly and easily have a look at them by just calling the object (it will automatically print it via `print.dispRity`). For example:

```
## Which class is the sum_of_ranges object?
class(sum_of_ranges)

## [1] "dispRity"

## We can summarise it using the S3 method print.dispRity
sum_of_ranges

## Disparity measurements across 9 series for 99 elements
## Series:
## 133.51104, 116.82216, 100.13328, 83.4444, 66.75552, 50.06664 ...
## Disparity calculated as: sum ranges for 97 dimensions.
## Data was split using continuous method.
## Data was bootstrapped 100 times, using the full bootstrap method.

## Some attentive ape package users will note the similarities with a phylo object...
```

Note that each object output from the different functions detailed below will display slightly different summarised informations (depending on the object content).

3.2 Splitting the data

One of the first functionality of this package is to facilitate splitting the ordinated matrix in subsamples. In fact, in many disparity analysis, one will analysis differences in subsamples of the any-o-space. For example, the subsamples can be composed of elements at a certain point in time (disparity-through-time analysis), or elements with some traits in common (e.g. habitat, phylogeny, etc.).

Custom splitting

`cust.series` is a function that allows to split the ordinated matrix according to a factor determined by the user. For example, here we can split the matrix based on phylogeny: let's classify in two different groups the crown and stem mammals present in the tree.

```
## We want to separate the species around the node 71 (see phylogeny above).
## All the descendant of these node are crown and all the ancestors are stem
crown <- extract.clade(MyData_tree, node = 71)$tip.label
stem <- drop.tip(MyData_tree, tip = crown)$tip.label
## We then have to feed this information in a data frame with one column
factors <- as.data.frame(
    matrix(data = c(rep("crown", length(crown)), rep("stem", length(stem))),
    ncol = 1, dimnames = list(c(crown, stem), "Group")))
## We then can use the customised series function
crown_stem <- cust.series(MyData_mat50, factors)
## This created a dispRity object containing two series: crown and stem
class(crown_stem)

## [1] "dispRity"

crown_stem

## 2 custom series for 50 elements
## Series:
## Group.crown, Group.stem.

## This object contains three elements
names(crown_stem)

## [1] "data"     "elements" "series"

## With "data" being the list sub-matrices
str(crown_stem$data)

## List of 2
##  $ Group.crown: num [1:30, 1:48] 0.3079 0.6531 0.5089 -0.1652 -0.0419 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:30] "Dasypodidae" "Bradypus" "Myrmecophagidae" "Todralestes" ...
##   .. ..$ : NULL
##  $ Group.stem : num [1:20, 1:48] -0.739 -0.68 -0.511 -0.477 -0.473 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:20] "Daulestes" "Bulaklestes" "Uchkudukodon" "Kennalestes" ...
##   .. ..$ : NULL

## "taxa" being the list of taxa in the original ordinated matrix
str(crown_stem$taxa)

##  NULL

## and finally, "series" containing information on the series type (custom) and names
## (crown and stem)
crown_stem$series

## [1] "custom"      "Group.crown" "Group.stem"
```

Of course, using phylogeny as a factor is just an example. Any type of factors can be used in this example (e.g. body mass categories, habitat, diet, etc...). See the other dispRity demos for other specific examples.

Time splitting

`time.series` is a function that allows to split the matrix in different ways using time as a category. This functions needs as input an ordinated matrix and a matching phylogenetic tree. Two types of time series splitting can be performed by using the `method` option:

1. the classical discrete time splitting (or time-binning) using `method = "discrete"`;
2. or a new continuous time splitting (or time-slicing) using `method = "continuous"`.

For the time-slicing method details, see Guillerme and Cooper *in prep.* [2]. Once the method is chosen, the function intakes the `time` argument which can be a vector of `numeric` values for:

1. defining the boundaries of the time bins (when `method = "discrete"`);
2. defining the time-slicing places (when using `method = "continuous"`).

Otherwise, the `time` argument can be set as a single `numeric` value for automatically generating a fix number of equidistant time-bins/slices. Additionally, we can also input a data frame containing the First/Last Occurrence Data (`FADLAD`) for taxa that span over a longer time than the tips/nodes age.

Let's see and example for time-binning (`method = "discrete"`):

```
## Let's generate three time bins containing the data present every 40 Ma
time_bins <- time.series(data = MyData_mat50, tree = MyData_tree,
    method = "discrete", time = c(120, 80, 40, 0))

## No FAD/LAD table has been provided.
## Every tips are assumed to be single points in time.

## Note that the function provides a warning saying that tips where single
## points in time (no FAD/LAD information). We can fix that by adding the
## age data for the taxa that have some longer occurrence spans.
time_bins <- time.series(data = MyData_mat50, tree = MyData_tree,
    method = "discrete", time = c(120, 80, 40, 0), FADLAD = MyData_ages)

## Some tips have no FAD/LAD and are assumed to be single points in time.

## To entirely avoid the warning we could collect the occurrence span data
## for all the taxa but that's not necessary. The function automatically
## assumes no occurrence span time (i.e. single points in time) for all taxa
## by default.
time_bins

## 3 discrete series for 50 elements
## Series:
## 120-80, 80-40, 40-0.
```

This generated indeed a list of 3 sub matrices. Note that we can also generate equivalent results by just telling the function that we want three time-bins (series) as follow:

```
time.series(data = MyData_mat50, tree = MyData_tree, method = "discrete", time = 3)

## No FAD/LAD table has been provided.
## Every tips are assumed to be single points in time.

## 3 discrete series for 50 elements
## Series:
## 133.51104-89.00736, 89.00736-44.50368, 44.50368-0.
```

We now have three time bins of 44.50368 Ma each. But hey, that's a bit silly, let's stick to bins of 40 Ma. When using this method, the oldest boundary of the first bin (or the first slice, see below) is automatically generated as being the root age + 1% of the the tree length as long as at least three elements are present at that point in time (an extra 1% is added until reaching the required minimum of three elements). We can also ask to include nodes in each bin by using `inc.nodes = TRUE` and providing a matrix that contains the ordinated distance between tips *AND* nodes.

For the time-slicing method (`method = "continuous"`), the idea is pretty similar. We need to provide a matrix that contains the ordinated distance between taxa AND nodes and an evolutionary model via the `model` argument. For now they are four evolutionary models implemented [2] (but see section 4):

1. `"acctran"` where the data chosen on each time slice is always the one of the offspring
2. `"deltran"` where the data chosen on each time slice is always the one of the descendant
3. `"punctuated"` where the data chosen on each time slice is randomly chosen between the offspring or the descendant
4. `"gradual"` where the data chosen on each time slice is either the offspring or the descendant depending on branch length

```
## Let's generate four time slices every 40 Ma assuming a gradual evolution model
time_slices <- time.series(data = MyData_mat99, tree = MyData_tree,
    method = "continuous", model = "gradual", time = c(120, 80, 40, 0),
    FADLAD = MyData_ages)

## Some tips have no FAD/LAD and are assumed to be single points in time.

time_slices

## 4 continuous series for 99 elements
## Series:
## 120, 80, 40, 0.

## Note that in the same way as for the discrete method, we can also
## automatically generate the slices
time.series(data = MyData_mat99, tree = MyData_tree, method = "continuous",
    model = "gradual", time = 4)

## No FAD/LAD table has been provided.
## Every tips are assumed to be single points in time.

## 4 continuous series for 99 elements
## Series:
## 133.51104, 89.00736, 44.50368, 0.

## But that gives the same silly results (slicing every 44.50368 Ma)
```

3.3 Bootstrapping the data

After separating the any-o-space into submatrices, one can directly calculate disparity within each subsample or first test the robustness of the data. To measure how robust is the data, we can perform bootstrap pseudo-replications or rarefaction analysis to make sure the results are not dependant on weird outliers. The `boot.matrix` allows to bootstraps and rarefy ordinated matrices in a fast and easy way. The default options will bootstrap the matrix 1000 times with no rarefaction.

```
boot.matrix(data = MyData_mat50)

## Bootstrapped ordinated matrix with 50 elements
## 1 unnamed series.
## Data was bootstrapped 1000 times, using the full bootstrap method.

## As we can wee, the output is also a dispRity object that is summarised
## automatically, and gives information on the data as well as the number of
## bootstraps and the bootstraps methods.
```

This function proposes the user to specify the bootstrap algorithm with the `boot.type` method. Currently two algorithms are implemented (but see section 4):

1. `"full"` where the bootstrapping is entirely stochastic (all the data is bootstrapped)
2. `"single"` where only one random taxa is replaced by one other random taxa each pseudo-replication

This function also has a `rarefaction` option to rarefy the data (how unexpected!). The default status is `FALSE` but it can be set to `TRUE` to fully rarefy the data (i.e. remove n taxa for the number of pseudo-replicates, where n varies from the maximum number of taxa present in the dataset to a minimum of 3 elements). It can also be set to a fix `numeric` value (or a set of numeric values). Finally, the last option from this function, `rm.last.axis`, is whether to remove a certain amount of axis of the ordinated data. This option can be logical where `FALSE` (default) will not remove any axis and `TRUE` will remove the 5% last axis or it can be set to a user defined proportion. This is a common practice in disparity-through-time analysis but it is hard to justify statistically. Here is a series of examples:

```
## Bootstrapping with the single bootstrap method
boot.matrix(MyData_mat50, boot.type = "single")

## Bootstrapped ordinated matrix with 50 elements
## 1 unnamed series.
## Data was bootstrapped 1000 times, using the single bootstrap method.

## Bootstrapping with the full rarefaction
boot.matrix(MyData_mat50, bootstraps = 20, rarefaction = TRUE)

## Bootstrapped ordinated matrix with 50 elements
## 1 unnamed series.
## Data was bootstrapped 20 times, using the full bootstrap method.
## Data was fully rarefied (down to 3 elements).

## Or with a set number of rarefaction levels
boot.matrix(MyData_mat50, bootstraps = 20, rarefaction = c(6:8,3))

## Bootstrapped ordinated matrix with 50 elements
## 1 unnamed series.
## Data was bootstrapped 20 times, using the full bootstrap method.
## Data was rarefied with a maximum of 6, 7, 8 and 3 elements

## And removing the last axis (default)
boot.matrix(MyData_mat50, rm.last.axis = TRUE)

## Bootstrapped ordinated matrix with 50 elements
## 1 unnamed series.
## Data was bootstrapped 1000 times, using the full bootstrap method.
## The 6 last axis were removed from the original ordinated data.
```

```
## Or with a fix value (50%)
boot.matrix(MyData_mat50, rm.last.axis = 0.5)

## Bootstrapped ordinated matrix with 50 elements
## 1 unnamed series.
## Data was bootstrapped 1000 times, using the full bootstrap method.
## The 35 last axis were removed from the original ordinated data.
```

Of course, one could be interested in directly supplying the subsampled matrices generated above directly to this function. In fact, it can also deal with a list of matrices or with a `dispRity` object output from the `cust.series` or `time.series` functions. Like this:

```
## Let's do a full rarefaction on the crown/stem data for later
crown_stemBS <- boot.matrix(crown_stem, rarefaction = TRUE)
## And just bootstrapping for the time binning/slicing
time_binsBS <- boot.matrix(time_bins)
time_slicesBS <- boot.matrix(time_slices)
## And just to remind us what's in each objects
crown_stemBS

## Bootstrapped ordinated matrix with 50 elements
## Series:
## Group.crown, Group.stem.
## Data was split using custom method.
## Data was bootstrapped 1000 times, using the full bootstrap method.
## Data was fully rarefied (down to 3 elements).

time_binsBS

## Bootstrapped ordinated matrix with 50 elements
## Series:
## 120-80, 80-40, 40-0.
## Data was split using discrete method.
## Data was bootstrapped 1000 times, using the full bootstrap method.

time_slicesBS

## Bootstrapped ordinated matrix with 99 elements
## Series:
## 120, 80, 40, 0.
## Data was split using continuous method.
## Data was bootstrapped 1000 times, using the full bootstrap method.
```

3.4 Calculating disparity

Now we're going to see the functionalities of the core function of this package: the `dispRity` function. This function is a modulable function that allow to simply (and quickly!) calculate disparity from a matrix.

Because disparity can be measured in many ways, this function is a tool to measure disparity *as defined by the user* (and here's where the modulable part comes in). In fact, the `dispRity` function intakes two main arguments: the data and the disparity metric. The disparity metric boils down to a single value that describes the matrix. Some are classic like the sum or the product of the ranges or the variances of the matrix columns (i.e. the any-o-space axis/dimensions) [3]. However, many more exist like the median distances between the taxa and the any-o-space centroid [2].

One can usually decompose the disparity metrics into two elements:

1. the **class metric** that is a descriptor of the matrix. For example describing the ranges of each column in the matrix or the euclidean distances between each row and the centroid of the matrix.

2. the **summary metric** that is a summary of the class metric values. For example, the sum of the ranges or the median of the euclidean distances.

Basically the combination can be infinite between the class and summary metrics. For example, people might want to measure the median variances of the axis or the product of the distances from the centroid. However, it is probable that some metrics are better to reflect some biological aspects of the any-o-space than others (see section 4)...

In practice, the `dispRity` function intakes a pair of class and summary metrics as a definition of disparity. Several of these metrics are implemented in other packages (like `stats::median`, `base::sum`, etc.) and this package proposes several metrics listed in `dispRity.metric` (see `?dispRity.metric`). It is even possible to use your very own class and summary metrics! This will be actually heavily encouraged and facilitate with the `make.metric` function in a future version (see 4).

To use these metrics pairs in the `dispRity` function, it's pretty easy:

```
dispRity(MyData_mat50, metric = c(sum, ranges))

## Disparity measurements across 1 series for 50 elements
## 1 unnamed series.
## Disparity calculated as: sum ranges for 48 dimensions.

## Yep, that's it. The order of the metrics does not matter (the function detects
## automatically which one is the class and the summary metric) so the combinations
## are numerous!
dispRity(MyData_mat50, metric = c(sum, variances))

## Disparity measurements across 1 series for 50 elements
## 1 unnamed series.
## Disparity calculated as: sum variances for 48 dimensions.

dispRity(MyData_mat50, metric = c(prod, centroids))

## Disparity measurements across 1 series for 50 elements
## 1 unnamed series.
## Disparity calculated as: prod centroids for 48 dimensions.

dispRity(MyData_mat50, metric = c(mode.val, ranges))

## Disparity measurements across 1 series for 50 elements
## 1 unnamed series.
## Disparity calculated as: mode.val ranges for 48 dimensions.

## Or funny user defined ones!
total.range <- function(X) abs(range(X)[1]-range(X)[2])
dispRity(MyData_mat50, metric = c(total.range, ranges))

## Disparity measurements across 1 series for 50 elements
## 1 unnamed series.
## Disparity calculated as: total.range ranges for 48 dimensions.

## That would be disparity as the range of the axis ranges!
```

Of course, this function can take a simple ordinated matrix but, more interesting it can also deal with a list of matrices (from the `cust.series` or `time.series` functions for example) or a bootstrapped or/and rarefied matrix (from the `boot.matrix` function for example).

```
## For example, let's calculate the sum of ranges from our crown/stem mammals
dispRity(crown_stem, metric = c(sum, ranges))

## Disparity measurements across 2 series for 50 elements
## Series:
## Group.crown, Group.stem.
## Disparity calculated as: sum ranges for 48 dimensions.
## Data was split using custom method.

## Or even better, from the bootstrapped data
disp_crown_stemBS <- dispRity(crown_stemBS, metric = c(sum, ranges))
## Note that the bootstrapped + rarefied data takes some time to run (it's
## calculating disparity from 46 matrices!)

## Let's calculate a different metric for the disparity-through-time analysis
disp_time_binsBS <- dispRity(time_binsBS, metric = c(median, centroids))
disp_time_slicesBS <- dispRity(time_slicesBS, metric = c(median, centroids))
```

Again, note that the results are `dispRity` objects and can displayed through the S3 `print.dispRity` method to remind users what are in these objects. Also note that an even faster parallel version will be developed in the future (see section 4).

3.5 Summarising the results

As shown in these examples above however, even though the disparity has been calculated in many way, the function doesn't directly outputs the results, rather just a summary of the analysis. To really display the results we can use the S3 `summary.dispRity` and `plot.dispRity` functions.

`summary` function

This function intakes a `dispRity` object plus various options namely the `quantile` values for the confidence intervals levels; the `cent.tend` for the central tendency to use for summarising the results and two *visual* options which are whether to recall the `dispRity` options and how much digits are wanted in the results.

```
## Let's have a look at the disparity-through-time using in the time-binned data
summary(disp_time_binsBS)

##   series  n observed  mean  2.5%   25%   75% 97.5%
## 1 120-80  8    1.203 1.116 0.933 1.081 1.172 1.237
## 2  80-40 27    1.344 1.323 1.275 1.309 1.341 1.359
## 3   40-0 16    1.353 1.323 1.250 1.303 1.345 1.372

## The 50 and 95 quantiles are calculated and the central tendency is the mean by default.
## Note that the function also displays the observed disparity (non-bootstrapped).
## We can change that easily by specifying different values in the options
summary(disp_time_binsBS, quantile = 88, cent.tend = sd, rounding = 1)

##   series  n observed  sd  6% 94%
## 1 120-80  8      1.2 0.1 1.0 1.2
## 2  80-40 27      1.3 0.0 1.3 1.4
## 3   40-0 16      1.4 0.0 1.3 1.4

## Also, if we happen to have forgotten what was in the disp_time_binsBS object (as in
## which options where used to modify the original matrix) we can use the recall option:
summary(disp_time_binsBS, recall = TRUE)
```

```
## Disparity calculated as: median centroids for 48 dimensions.
## Data was split using discrete method.
## Data was bootstrapped 1000 times, using the full bootstrap method.
##   series  n observed  mean  2.5%   25%   75% 97.5%
## 1 120-80  8    1.203 1.116 0.933 1.081 1.172 1.237
## 2  80-40 27    1.344 1.323 1.275 1.309 1.341 1.359
## 3   40-0 16    1.353 1.323 1.250 1.303 1.345 1.372

## Note that the recall just prints text. The output of summary.dispRity is always a
## data.frame
summary_table <- summary(disp_time_binsBS, recall = TRUE)

## Disparity calculated as: median centroids for 48 dimensions.
## Data was split using discrete method.
## Data was bootstrapped 1000 times, using the full bootstrap method.

class(summary_table)

## [1] "data.frame"

## Finally we can see the results of the rarefaction analysis for the crown/stem data
head(summary(disp_crown_stemBS))

##        series n observed  mean  2.5%   25%   75% 97.5%
## 1 Group.crown 3       NA 15.13 10.27 15.12 15.99 16.60
## 2 Group.crown 4       NA 18.41 14.79 18.31 19.36 19.91
## 3 Group.crown 5       NA 20.73 16.15 19.48 21.84 22.46
## 4 Group.crown 6       NA 22.65 18.88 21.76 23.77 24.46
## 5 Group.crown 7       NA 24.18 21.06 23.41 25.27 26.07
## 6 Group.crown 8       NA 25.52 22.12 24.88 26.63 27.46

## This outputs a longer table with all the variations of the number of crown/stem taxa
## from the maximum (30 and 20) to the minimum (3). Note that the observed disparity
## is NA for rarefied data.
```

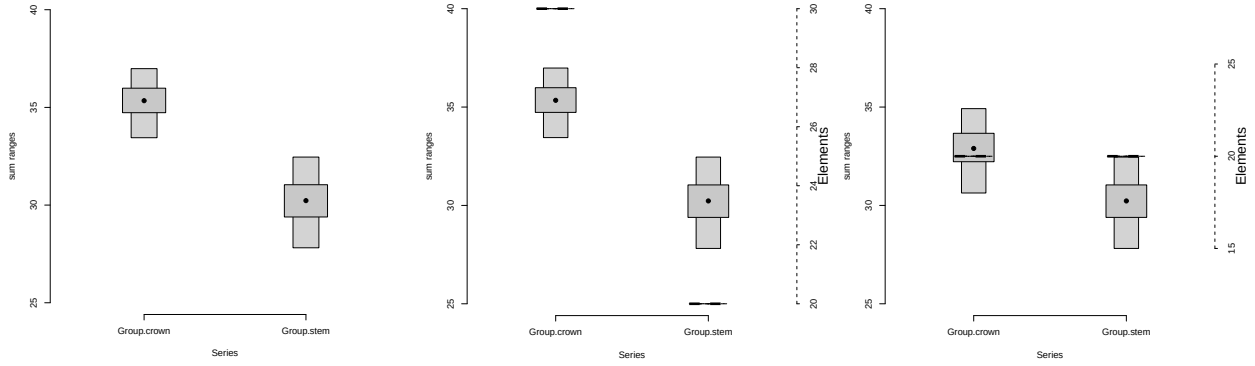
`plot` function

This final function allows to visualise the disparity results in a often nicer fashion than just a table (even though the exact same data is displayed). The `plot.dispRity` option intakes the same options as `summary.dispRity` along side with various graphical options described in the function manual. Here, let's just have a look a few of these options.

Let's start with plotting the difference in disparity between the crown and the stem mammals. For this we are going to simply specify the type of plots using the `type` option that allows to choose between the `"continuous"` or `"discrete"` method. In our case, we are interested in looking at the results in a discrete way. Note that the plot `type` can be left missing. The function will then use `"discrete"` as default or `"continuous"` if the input data came from the `time.series` function with `method = "continuous"`.

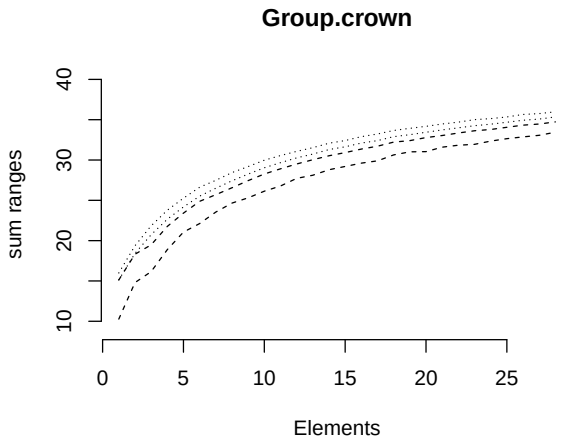
The grey squares represent the confidence intervals (50 in dark grey and 90 in light grey) and the dot represents the mean. However, it might be interesting to see how many taxa are present in each group. This can be done via the `elements` option. A third interesting option is to plot the rarefied data if a rarefaction was applied on the data with the `boot.matrix` function. For displaying the rarefied data we can use the `rarefaction` option. This option can be either `FALSE` (default) for always plotting the maximum number of taxa per series or `TRUE` to always plot the minimum. Additionally, it can be a specific number to plot a specific number of taxa. Let say we want both groups to have 20 taxa maximum. We can see now that the confidence intervals overlap more than previously

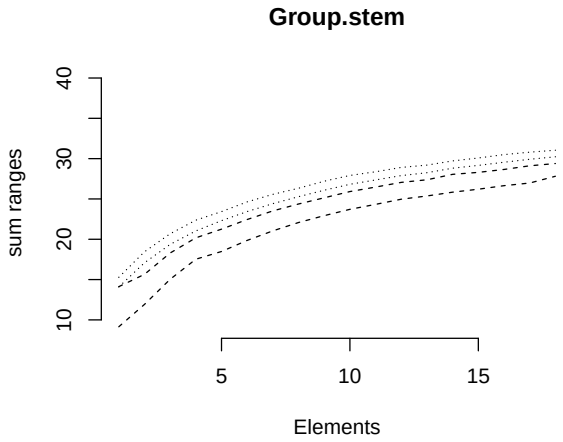
```
## Graphical options
quartz(width = 15, height = 5) ; par(mfrow = (c(1,3)), bty = "n")
## Plotting the score for both groups
plot(disp_crown_stemBS, ylim = c(25,40))
## Same one with the number of taxa for each data set
plot(disp_crown_stemBS, elements = TRUE, ylim = c(25,40))
## Same one with rarefied data (note how that affects the elements data as well!)
plot(disp_crown_stemBS, elements = TRUE, rarefaction = 20, ylim = c(25,40))
```



To understand how the number of taxa affects each series, we can use the `rarefaction = "plot"` function to plot the rarefaction curves.

```
## Graphical options
par(bty = "n")
## The rarefaction curves
plot(disp_crown_stemBS, elements = TRUE, rarefaction = "plot")
## Note that the function automatically draws the curves for each series and splits
## the plot screen accordingly.
```

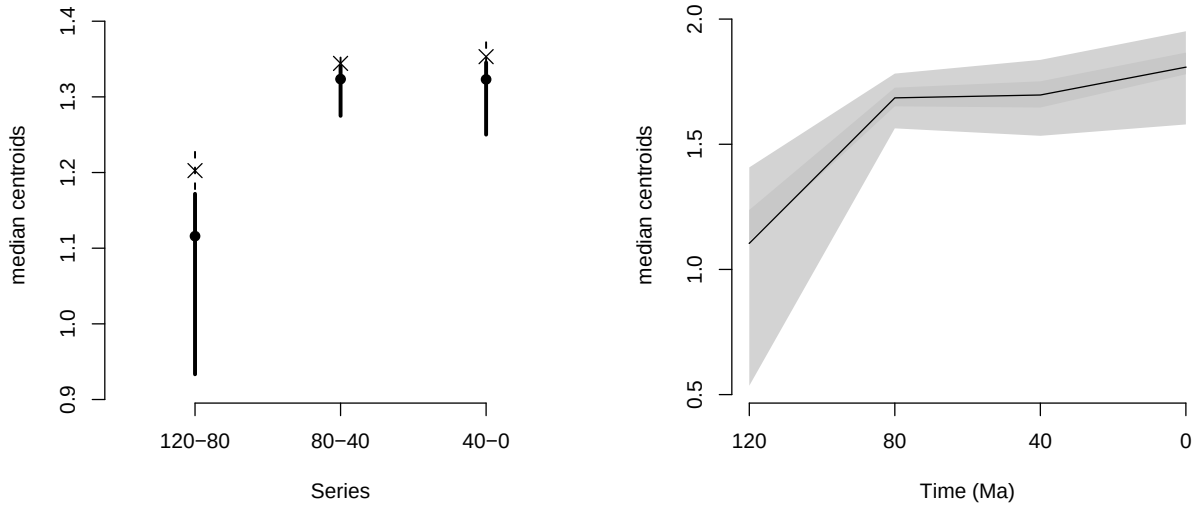




The different dashed lines represent the different confidence intervals around each rarefaction curve.

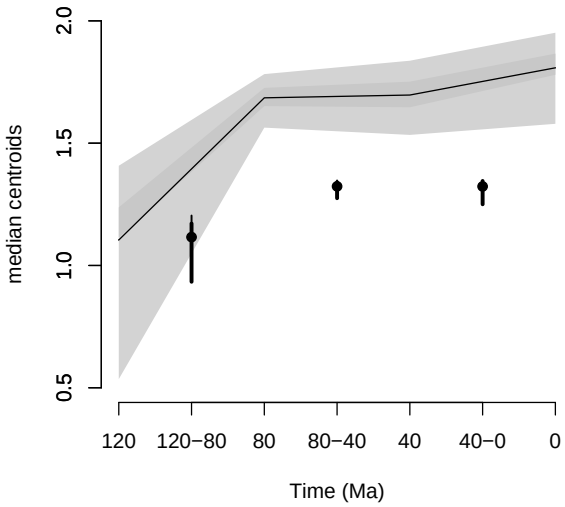
Additionally, for the `type = "discrete"` option, we might also decide to plot many boxplots (which can be a bit messy). The `discrete_type` option allows to switch between `"box"` or `"line"` for showing the results in one or the other format. We can also plot the observed data when existing (i.e. not with some rarefaction levels) by using `observed = TRUE`. Let's look at that with the time binned data (even though there is only three time bins). Finally, we can use the `type = "continuous"` option that comes in handy for plotting continuous data like in the time sliced analysis (obviously...).

```
## Graphical options
quartz(width = 10, height = 5) ; par(mfrow = (c(1,2)), bty = "n")
## The disparity-through-time data in a time-binned way (with lines rather than
## boxes) with the observed data as crosses
plot(disp_time_binsBS, discrete_type = "line", observed = TRUE)
## The disparity-through-time data in a time-sliced ways
plot(disp_time_slicesBS)
```



We can even plot both results (discrete and continuous) on the same graph:

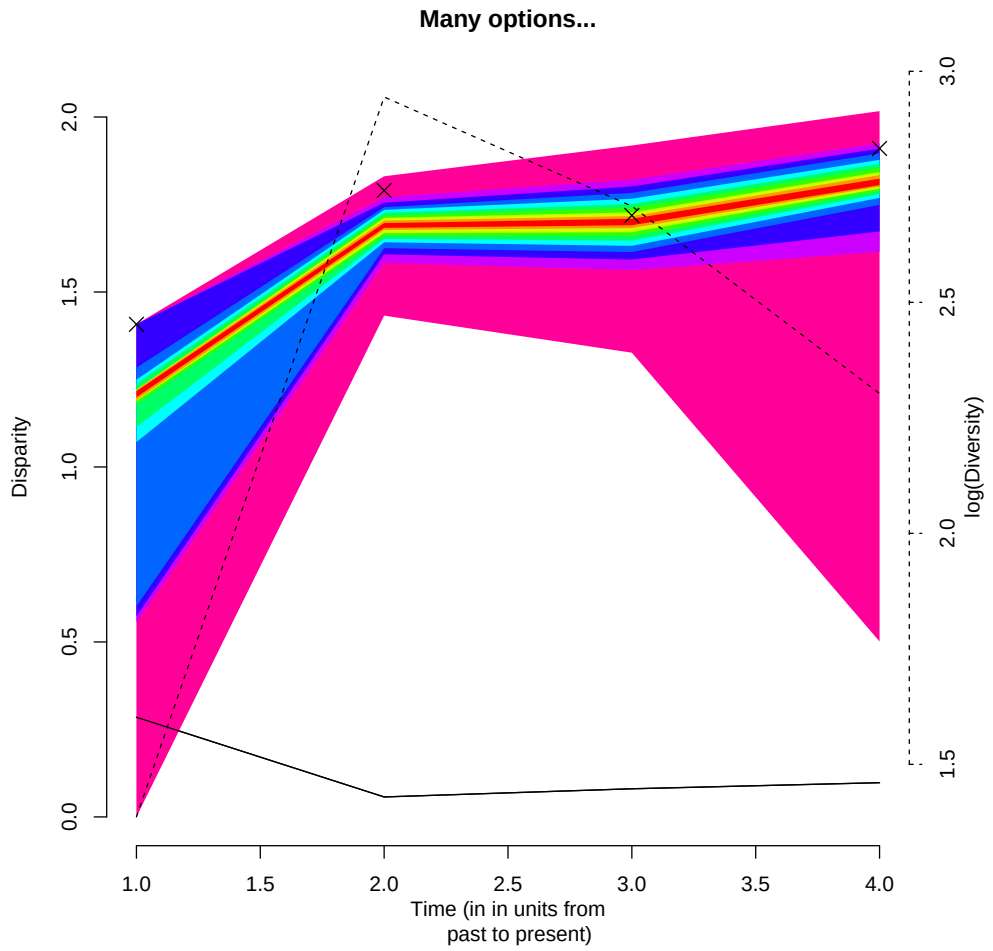
```
## Graphical options
par(bty = "n")
## First, plotting the continuous data
plot(disp_time_slicesBS, ylim=c(0.5,2))
## Second, adding the discrete data
par(new = TRUE)
plot(disp_time_binsBS, discrete_type = "line", ylim=c(0.5,2), xlab="", ylab="")
```



Note that the results vary in values but not in pattern. The change in values might be due to the fact that nodes data are not included in the discrete analysis.

I encourage you to play with the graphical options to have some prettier results. Note that most of the options from `plot` can be passed to `plot.dispRity` via `...`.

```r
## Graphical options
par(bty = "n")
## A plot with many options!
plot(disp_time_slicesBS, quantile = c(seq(from=10, to=100, by=10)), cent.tend = sd,
    type = "continuous", elements = "log", col = c("black", rainbow(10)),
    ylab = c("Disparity", "log(Diversity)"), xlab = "Time (in in units from
    past to present)", time.series = FALSE, observed = TRUE, main = "Many options...")
```



4 What's left?

As stated at the start of the demo, this version 0.1.1 is still in development and many parts are missing. Here are the new functionalities that will be implemented before the proper release version (v.1).

4.1 Some statistics

Some people might get interested in p-values and other statsy numbers. Because the format of the `dispRity` objects is a bit special, I am currently developing a `dispRity.test` function that will allow to intake your favourite statistical test and a `dispRity` object. I will also propose several inbuilt test such as the time lag t-test [2]. Any other suggestions are most welcome!

4.2 Users defined metrics

To help people develop their own disparity metrics (both class and summary) and to make sure they will be usable in the `dispRity` function, I am developing a `make.metric` function that will test the metric in two ways: (1) is the output sensible (defined by the user)? (2) does it compute properly in the `dispRity` function?

Along side with developing this package, I also aim to test the efficiency of some metrics in describing some aspects of the any-o-space so that it could be used as guidelines for disparity analysis

4.3 More user defined stuff

I intend also develop functions to help users to develop their own algorithms for the bootstrap method (via `make.boot`) or the evolutionary models (via `make.model`). Both functions will provide similar testing as the `make.metric` function.

4.4 Faster!

Finally, for long analysis, I intend to develop a parallel running version of the package. In fact, most of the internal functions are base on `lapply` functions that can be easily passed to `snow::parLapply` or similar parallel functions.

References

[1] Beck RM, Lee MS. Ancient dates or accelerated rates? Morphological clocks and the antiquity of placental mammals. Proceedings of the Royal Society B: Biological Sciences. 2014;281(20141278):1–10. Available from: `http://dx.doi.org/10.1098/rspb.2014.1278`.

[2] Guillerme T, Cooper N. Mammalian morphological diversity does not increase in response to the Cretaceous-Paleogene mass extinction and the extinction of the (non-avian) dinosaurs. in prep;Available from: `https://github.com/TGuillerme/SpatioTemporal_Disparity/raw/master/Writing/STD_draft.pdf`.

[3] Wills MA, Briggs DEG, Fortey RA. Disparity as an Evolutionary Index: A Comparison of Cambrian and Recent Arthropods. Paleobiology. 1994 04;20(2):93–130. Available from: `http://www.jstor.org/stable/2401014`.