A jrich worked example

D.R. Miranda Esquivel

06/03/2016

These functions calculate the taxonomic measures presented in Miranda-Esquivel (2016). The package introduces Jack-knife resampling in evolutionary distinctiveness prioritization analysis, as a way to evaluate the support of the ranking in area prioritization, and the persistence of a given area in a conservation analysis.

For further information, you could read http://link.springer.com/chapter/10.1007/978-3-319-22461-9_11

1. An example with a single topology and distribution to reproduce Figure 1 in Miranda-Esquivel (2016)

First of all, we remove everything from the R environment.

```
#
rm(list=ls())
```

And close the graphic devices, if there is any.

```
#
if (dev.cur()!=1){dev.off()}
```

```
## null device
##           1
```

Now, we cat get the stable version of the library from CRAN or the latest version from GitHub.

```
#library("devtools")

#install_github("Dmirandae/jrich")


install.packages('jrich')
```

We load the library before proceeding.

```
library(jrich)
```

```
## Loading required package: ape
```

and set the working directory to the R Data (make sure you have changed it to your own directory).

```
# setwd("./myData/")
```

Now, we can read a tree from `figure1.tre`, which is written in Newick format or we can use directly the tree from the data file.

To read the tree form a file use:

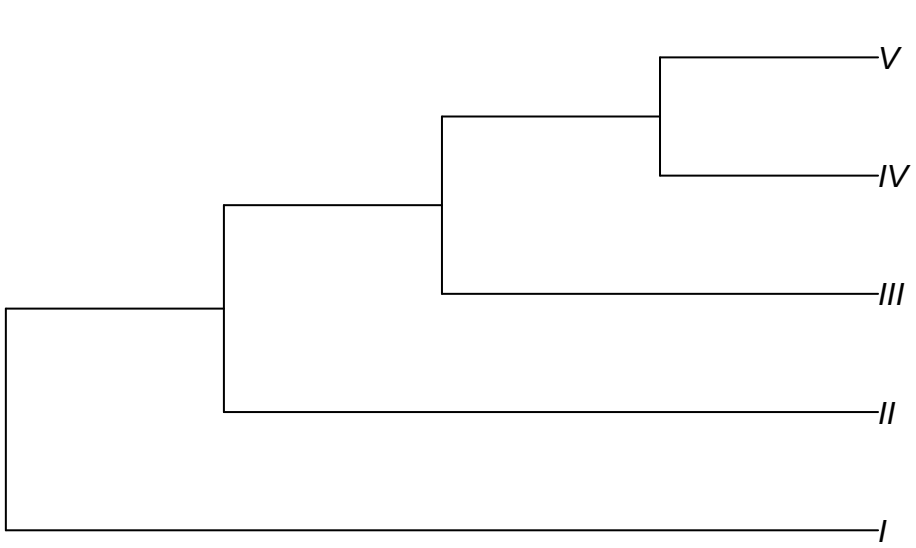
```
tree.figure1 <- read.tree ("yourTree.tre")
```

Or, upload the data set, using the `data` funtion.

```
 data(tree)
```

We can plot the tree.

```
plot(tree, main= "Figure 1. Area Cladogram")
```



We can also read distributions using the species per area format. The distributions could be a csv file with several features: Each line has a species name and an area, and multiple areas for the same species means a widespread taxon in multiple lines.

The following function creates a data frame for the species distributions.

```
distrib.figure1 <- Read.Data("figure1.csv")

head(distrib.figure1)
```

Also, we could load the example distribution.

```
data(distribution)
```

We run the initial Index calculation, with a verbose output.

```
library(jrich)

initial.Values <- Calculate.Index(tree=tree, distrib = distribution, verbose=T)
```

```
## [1] "Deleted  0  out of  5"
```

```
initial.Values
```

```
##   area I Ie    Is   Ise     W    We    Ws   Wse rich endem jtopol jtip
## 1    A 8  8 0.200 0.364 4.000 4.000 0.158 0.300    1     1      0    0
## 2    B 4  4 0.100 0.182 2.000 2.000 0.079 0.150    1     0      0    0
## 3    C 2  2 0.050 0.091 1.333 1.333 0.053 0.100    1     0      0    0
## 4    D 1  1 0.025 0.045 1.000 1.000 0.039 0.075    1     0      0    0
## 5    E 1  1 0.025 0.045 1.000 1.000 0.039 0.075    1     0      0    0
## 6    F 8  2 0.200 0.091 5.333 1.333 0.211 0.100    4     0      0    0
## 7    G 8  2 0.200 0.091 5.333 1.333 0.211 0.100    4     0      0    0
## 8    H 8  2 0.200 0.091 5.333 1.333 0.211 0.100    4     0      0    0
```

Note that the figures for **Is/Ws** indices here are different from Figure 1 in Miranda-Esquivel 2016, as here are re-scaled to sum 1, but the proportions are exactly the same.

To obtain the same figures for **Is/Ws** indices as in Figure 1, we must use this code:

```
figure1.Values <-  Calculate.Index(tree=tree, distrib = distribution, verbose=F, standard = "tree")

figure1.Values
```

```
##   area I Ie    Is   Ise     W    We    Ws   Wse rich endem jtopol jtip
## 1    A 8  8 0.500 0.500 4.000 4.000 0.429 0.429    1     1      0    0
## 2    B 4  4 0.250 0.250 2.000 2.000 0.214 0.214    1     0      0    0
## 3    C 2  2 0.125 0.125 1.333 1.333 0.143 0.143    1     0      0    0
## 4    D 1  1 0.062 0.062 1.000 1.000 0.107 0.107    1     0      0    0
## 5    E 1  1 0.062 0.062 1.000 1.000 0.107 0.107    1     0      0    0
## 6    F 8  2 0.500 0.125 5.333 1.333 0.571 0.143    4     0      0    0
## 7    G 8  2 0.500 0.125 5.333 1.333 0.571 0.143    4     0      0    0
## 8    H 8  2 0.500 0.125 5.333 1.333 0.571 0.143    4     0      0    0
```

```
all.equal(initial.Values,figure1.Values)
```

```
## [1] "Component \"Is\": Mean relative difference: 1.499"
## [2] "Component \"Ise\": Mean relative difference: 0.374"
## [3] "Component \"Ws\": Mean relative difference: 1.71029"
## [4] "Component \"Wse\": Mean relative difference: 0.429"
```

1. Correlations between values

Plot the initial Values for the index that "explains" the most with this line:

```
correlations <- cor(initial.Values[,2:10],initial.Values[,2:10])
```

Do not forget using the following code to avoid the "autocorrelation".

```
diag(correlations) <- 0.0
```

Now we can determine the 'Best' descriptor:

```
best.Index <- which.max(apply(correlations,2,sum))

best.Index
```

```
## I
## 1
```

With the following instruction we can get the name of index that displays the highest value, and the row of the column **rich** in which the object is found.

Keep in mind that richness in not a good predictor for all indices.

```
which(abs(correlations[,"rich"])==max(abs(correlations[,"rich"])))
```

```
## Ws
##  7
```

To plot aesthetic graphics, load the library `ggplot2`.

```
library(ggplot2)
```

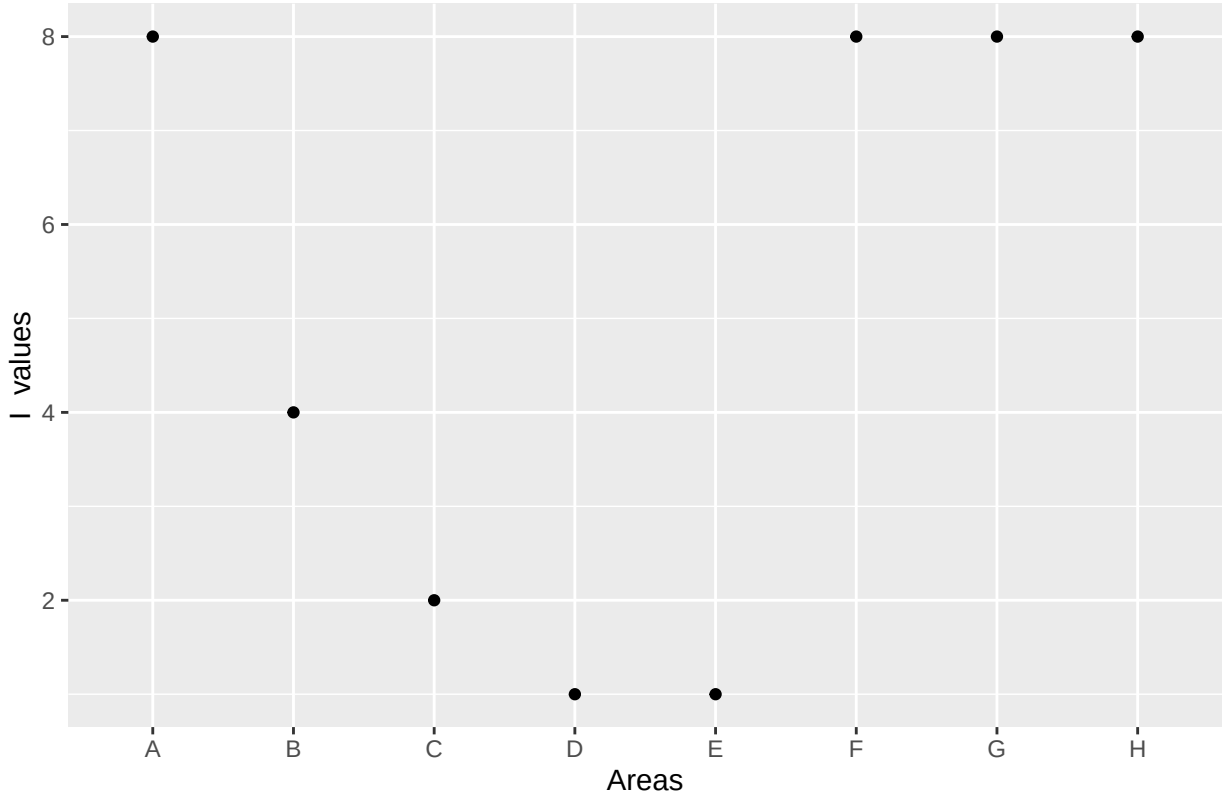
2. The index that "explains the most", without resampling is:

```
best.Index <-   which.max(apply(correlations,2,sum))

best.Index
```

```
## I
## 1
```

```
qplot(initial.Values$area,initial.Values[,names(best.Index)], xlab = "Areas",
      ylab =paste(names(best.Index)," values"), main = paste("Figure 2. Values of the most informative
```

Figure 2. Values of the most informative index: I Index



In this example, Areas **A / F / G / H** have the same ranking, as area **A** harbors species **I**, while areas **F / G / H** have the highest absolute richness.

We run the analysis with a single Jackknife replicate with a jtip value of 0.5.

```
jack.Values <-  Calculate.Index(tree=tree, distrib = distribution,jtip = 0.5)
```

```
## [1] "Deleted  4  out of  5"
```

The absolute difference between these two outputs can be computed with this instruction:

```
all.equal(initial.Values, jack.Values)
```

```
##  [1] "Component \"I\": Mean relative difference: 0.6666667"
##  [2] "Component \"Ie\": Mean relative difference: 1"
##  [3] "Component \"Is\": Mean relative difference: 0.6"
##  [4] "Component \"Ise\": Mean relative difference: 1"
##  [5] "Component \"W\": Mean relative difference: 0.7428424"
##  [6] "Component \"We\": Mean relative difference: 0.8236851"
##  [7] "Component \"Ws\": Mean relative difference: 0.5764236"
##  [8] "Component \"Wse\": Mean relative difference: 1"
##  [9] "Component \"rich\": Mean relative difference: 0.8125"
## [10] "Component \"endem\": Mean relative difference: 1"
## [11] "Component \"jtip\": Mean absolute difference: 0.5"
```

3. But a single replicate is not interesting, therefore we can repeat the process 100 times, using two approaches:

- **3.1** Jackknifing with a jtip value of 0.5, 100 replicates, using the `Calculate.Index` function.

```
jack.Ranking.100 <- list()
for (i in 1:100){
  ## if you want to get the output number replicate, uncomment this line:
  ##print(paste("replicate #",i), )

  jack.Ranking.100[[i]] <-  as.data.frame(Rank.Indices(Calculate.Index(tree=tree, distrib = distributio
}
```

- **3.2** Comparing the whole ranking for the best index, 100 times.

```
initial.Ranking <-  as.data.frame(Rank.Indices(initial.Values))

jack.Ranking.100.comparison <- NULL

for (i in 1:100){

  if(!all(jack.Ranking.100[[i]][,best.Index] == "XOX")){
    jack.Ranking.100.comparison[i] <- all.equal(initial.Ranking[,best.Index],
                                                jack.Ranking.100[[i]][,best.Index])
  }else jack.Ranking.100.comparison[i] <- 0
}
```

Get the number of hits or the extent of the Jackknife replicate to recover the initial ranking:

```
length(which(jack.Ranking.100.comparison==TRUE))
```

```
## [1] 7
```

To estimate the error, we use this code:

```
length(which(as.data.frame(jack.Ranking.100.comparison)!=TRUE))
```

```
## [1] 93
```

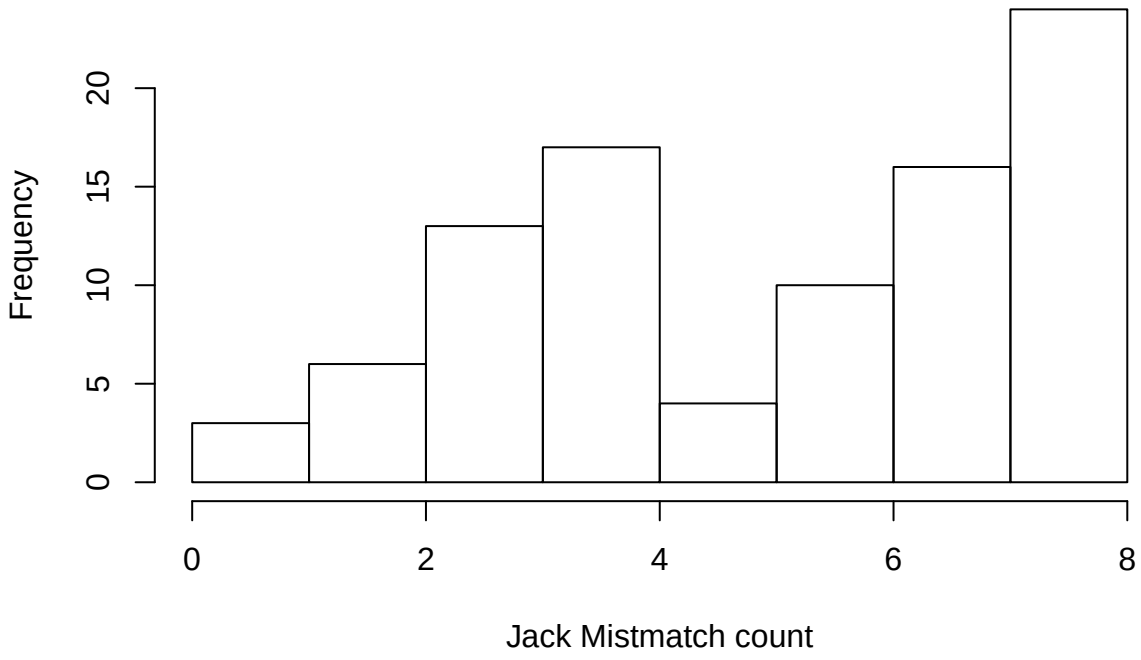
These two figures indicate that the best index is not **I**. As it is also important to estimate the error, we can use the following code to compute it.

```
jack.Mismatch <- jack.Ranking.100.comparison[jack.Ranking.100.comparison!=TRUE]

count.Jack.Mismatch <- gsub(" string mismatches","",jack.Mismatch)

count.Jack.Mismatch <- as.numeric(count.Jack.Mismatch)
```

Plot the distribution of the error, not so bell-shaped.

```
hist(sort(count.Jack.Mismatch,na.last = NA), main = "Figure 3. Histogram of Error Distribution", xlab=
```

Figure 3. Histogram of Error Distribution



4. A wrap to theprevious function, and evaluating the number of times we recovered 1/2/3 position in the ranking.

Note that `Calculate.Index` recovers the index values while `Best.Index` recovers the ranking comparison.

```
jack.figure1.jtip05.100replicates <- Best.Index(tree=tree, distrib = distribution, jtip = 0.5, replicat
```

```
## [1] "Deleted  0  out of  5"
## [1] "Deleted  4  out of  5"
## [1] "Deleted  3  out of  5"
## [1] "Deleted  2  out of  5"
## [1] "Deleted  2  out of  5"
## [1] "Deleted  1  out of  5"
## [1] "Deleted  2  out of  5"
## [1] "Deleted  1  out of  5"
## [1] "Deleted  3  out of  5"
## [1] "Deleted  3  out of  5"
## [1] "Deleted  2  out of  5"
## [1] "Deleted  3  out of  5"
## [1] "Deleted  1  out of  5"
## [1] "Deleted  1  out of  5"
## [1] "Deleted  2  out of  5"
## [1] "Deleted  2  out of  5"
## [1] "Deleted  2  out of  5"
## [1] "Deleted  4  out of  5"
## [1] "Deleted  2  out of  5"
## [1] "Deleted  1  out of  5"
## [1] "Deleted  2  out of  5"
## [1] "Deleted  1  out of  5"
## [1] "Deleted  3  out of  5"
```

```
## [1] "Deleted  2  out of  5"
## [1] "Deleted  3  out of  5"
## [1] "Deleted  2  out of  5"
## [1] "Deleted  3  out of  5"
## [1] "Deleted  4  out of  5"
## [1] "Deleted  3  out of  5"
## [1] "Deleted  4  out of  5"
## [1] "Deleted  0  out of  5"
## [1] "Deleted  4  out of  5"
## [1] "Deleted  3  out of  5"
## [1] "Deleted  2  out of  5"
## [1] "Deleted  3  out of  5"
## [1] "Deleted  1  out of  5"
## [1] "Deleted  1  out of  5"
## [1] "Deleted  2  out of  5"
## [1] "Deleted  3  out of  5"
## [1] "Deleted  3  out of  5"
## [1] "Deleted  3  out of  5"
## [1] "Deleted  2  out of  5"
## [1] "Deleted  4  out of  5"
## [1] "Deleted  1  out of  5"
## [1] "Deleted  3  out of  5"
## [1] "Deleted  1  out of  5"
## [1] "Deleted  5  out of  5"
## [1] "Deleted  3  out of  5"
## [1] "Deleted  4  out of  5"
## [1] "Deleted  2  out of  5"
## [1] "Deleted  2  out of  5"
```

```
jack.figure1.jtip05.100replicates
```

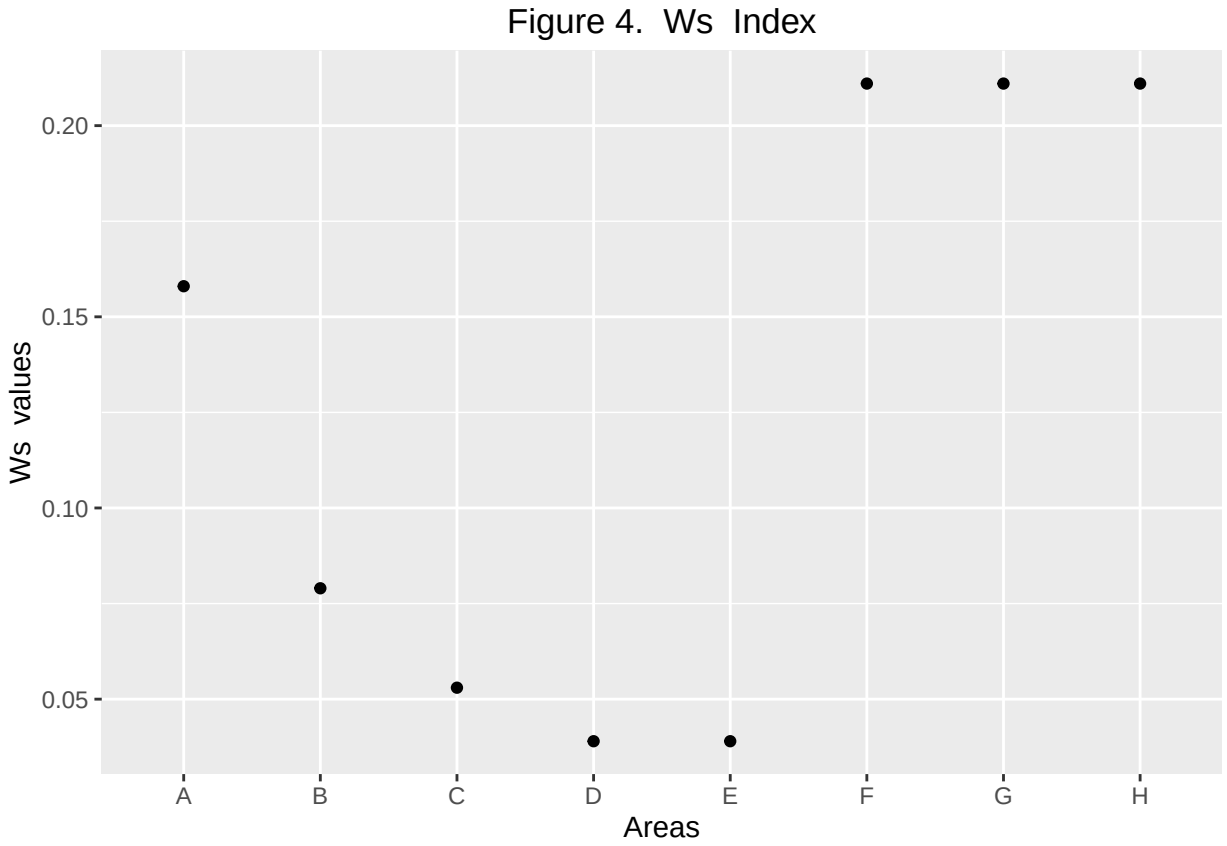
```
##   Wse Ws We  W Ise Is Ie  I
## 1   2 48 28 42   2 40 28 40
```

```
best.Index = names(jack.figure1.jtip05.100replicates)[c(which(jack.figure1.jtip05.100replicates == max(
```

W / Ws explains better than **I**, as they have a Jackknife value. In this context we can plot **Ws**.

```
for (i in 1:length(best.Index)){

print(best.Index[i])

print(qplot(initial.Values$area,initial.Values[,best.Index[i]], xlab = "Areas",
      ylab =paste(best.Index[i]," values"), main = paste("Figure 4. ",best.Index[i]," Index")))
}
```

```
## [1] "Ws"
```



As it is shown in **Figure 4.**, areas **F/G/H** have a higher value as they have higher richness, but even so, the support is relatively low.

2. An example with two topologies and distributions

For this example, We will work with a tree and a distribution for an real taxon: *Puranius* (These two files could be found in in the data directory).

```
tree.Puranius <- read.tree ("puranius.tre")

distrib.Puranius <- Read.Data ("puranius.csv.gz")
```

Now we will assign a list to the object `data.Puranius` in order to join the tree and the distribution, which could certainly be made with a function as well. However, I prefer this approach.

```
data.Puranius  <- list()

data.Puranius[[1]] <- tree.Puranius

data.Puranius[[2]] <- distrib.Puranius

head(data.Puranius)
```

We will proceed just as we did for the first Taxon, thence we got a tree and a distribution for another Taxon: *Janus*.

```
tree.Janus <- read.tree ("Janus.tre")

distrib.Janus <- Read.Data("Janus.csv.gz")

data.Janus  <- list()

data.Janus[[1]] <- tree.Janus

data.Janus[[2]] <- distrib.Janus

data.Janus
```

The following code creates a list to handle multiple datasets. In this case, the list `Multitaxon1` contains two objects.

```
Multitaxon1 <- list()

Multitaxon1[[1]] <- data.Janus

Multitaxon1[[2]] <- data.Puranius
```

The function `Multi.Index.Calc` calculates indices values for a MultiData list. For this case, it computes the Initial ranking for `Multitaxon1` with default values. This action is performed just once.

```
data(Multitaxon1)

initial.Values.Multi <-  Multi.Index.Calc(Multitaxon1)
```

```
## [1] "Deleted  0  out of  20"
## [1] "Deleted  0  out of  20"
```

Before plotting the initial Values for the Index that explains the most, make sure that the library `ggplot2` is loaded.

```
library(ggplot2)
```

1. Correlations between values

Plotting as example, the index value that explains the most is:

```
correlations.Multi <-  cor(initial.Values.Multi[,2:10], initial.Values.Multi[,2:10])
```

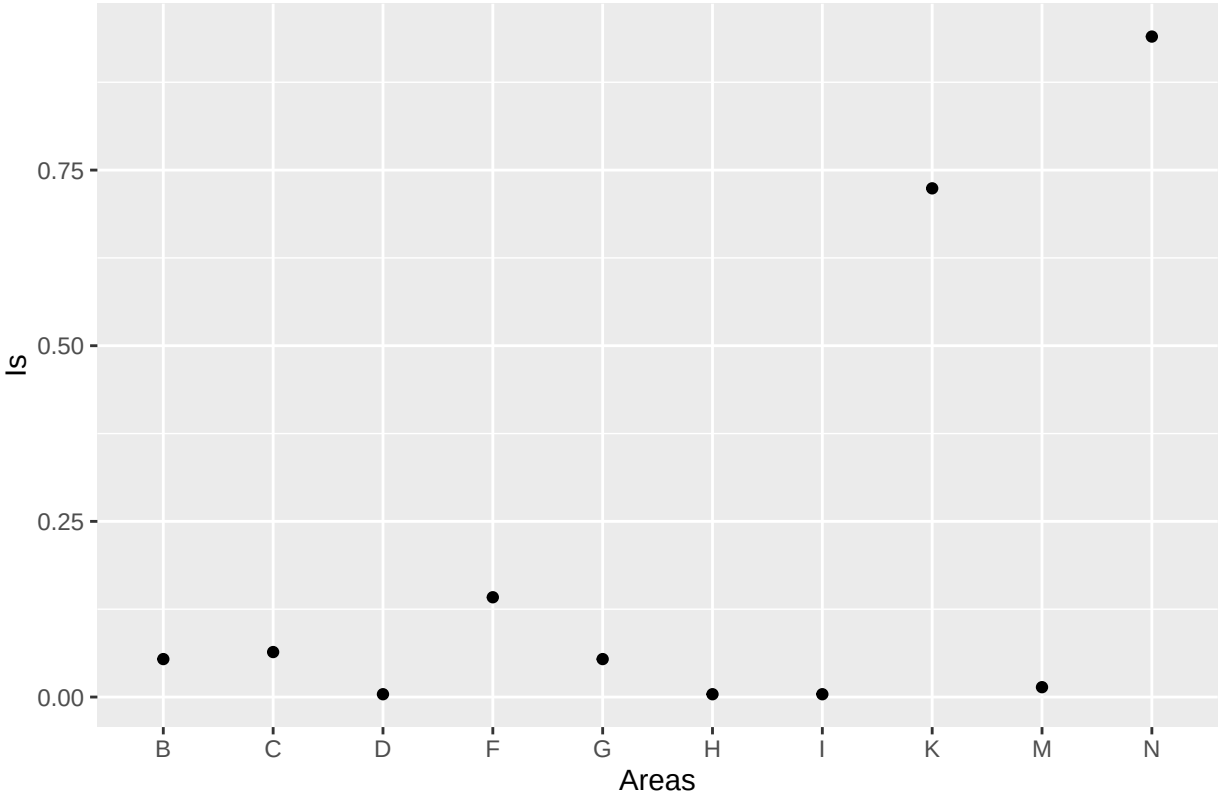
To avoid the highest "autocorrelation", use this line:

```
diag(correlations.Multi) <- 0.0
```

In case we want to know the index that explains the most, use this code:

```
best.Index.Multi <-   which.max(apply(correlations.Multi,2,sum))

qplot(initial.Values.Multi$area,initial.Values.Multi[,names(best.Index.Multi)], xlab = "Areas", ylab =na
```

Figure 5. Values of the Index that explains the most per area



2. A delete experiment: Prob jtip jtopol 0.5

```
jack.Multi <-  Multi.Index.Calc(Multitaxon1,jtip = 0.5,jtopol = 0.5)
```

```
## [1] "Deleted  0  out of  20"
## [1] "Deleted  6  out of  20"
```

See that the `jtopol` and `jtip` present the deleted proportions, and the two data frames are different.

```
all.equal(jack.Multi,initial.Values.Multi)
```

```
##  [1] "Component \"I\": Mean relative difference: 0.07621951"
##  [2] "Component \"Ie\": Mean relative difference: 0.3147732"
##  [3] "Component \"Is\": Mean relative difference: 0.08311822"
##  [4] "Component \"Ise\": Mean relative difference: 0.158"
##  [5] "Component \"W\": Mean relative difference: 0.2030981"
##  [6] "Component \"We\": Mean relative difference: 0.1651033"
##  [7] "Component \"Ws\": Mean relative difference: 0.1404298"
##  [8] "Component \"Wse\": Mean relative difference: 0.1215608"
##  [9] "Component \"rich\": Mean relative difference: 0.2571429"
## [10] "Component \"endem\": Mean relative difference: 0.3125"
## [11] "Component \"jtopol\": Mean relative difference: 1"
## [12] "Component \"jtip\": Mean relative difference: 1"
```

```
vector <-   which.max(apply(correlations,2,sum))
```

3. Jack Ranking Multitaxon 100 times

This operation computes the index values and returns a data matrix. Diverse from the first method, it allows to perform this operation 100 times. The number of replicates can be easily modified, by changing the rank in for `(i in 1:100)`.

```
jack.Ranking.100 <- list()

for (i in 1:100){
  print(paste("replicate #",i))
  jack.Ranking.100[[i]] <-  as.data.frame(Rank.Indices(Multi.Index.Calc(Multitaxon1,
  jtip = 0.5,jtopol = 0.5)))
}
```

```
## [1] "replicate # 1"
## [1] "Deleted  9  out of  20"
## [1] "Deleted  0  out of  20"
## [1] "replicate # 2"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  13  out of  20"
## [1] "replicate # 3"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  10  out of  20"
## [1] "replicate # 4"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  13  out of  20"
## [1] "replicate # 5"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  0  out of  20"
## [1] "replicate # 6"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  0  out of  20"
## [1] "replicate # 7"
## [1] "Deleted  12  out of  20"
## [1] "Deleted  0  out of  20"
## [1] "replicate # 8"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  9  out of  20"
## [1] "replicate # 9"
## [1] "Deleted  9  out of  20"
## [1] "Deleted  0  out of  20"
## [1] "replicate # 10"
## [1] "Deleted  12  out of  20"
## [1] "Deleted  12  out of  20"
## [1] "replicate # 11"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  0  out of  20"
## [1] "replicate # 12"
## [1] "Deleted  13  out of  20"
## [1] "Deleted  0  out of  20"
## [1] "replicate # 13"
```

```
## [1] "Deleted  12  out of  20"
## [1] "Deleted  7  out of  20"
## [1] "replicate # 14"
## [1] "Deleted  10  out of  20"
## [1] "Deleted  0  out of  20"
## [1] "replicate # 15"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  7  out of  20"
## [1] "replicate # 16"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  0  out of  20"
## [1] "replicate # 17"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  0  out of  20"
## [1] "replicate # 18"
## [1] "Deleted  9  out of  20"
## [1] "Deleted  0  out of  20"
## [1] "replicate # 19"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  10  out of  20"
## [1] "replicate # 20"
## [1] "Deleted  12  out of  20"
## [1] "Deleted  0  out of  20"
## [1] "replicate # 21"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  12  out of  20"
## [1] "replicate # 22"
## [1] "Deleted  8  out of  20"
## [1] "Deleted  0  out of  20"
## [1] "replicate # 23"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  0  out of  20"
## [1] "replicate # 24"
## [1] "Deleted  9  out of  20"
## [1] "Deleted  0  out of  20"
## [1] "replicate # 25"
## [1] "Deleted  12  out of  20"
## [1] "Deleted  11  out of  20"
## [1] "replicate # 26"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  8  out of  20"
## [1] "replicate # 27"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  14  out of  20"
## [1] "replicate # 28"
## [1] "Deleted  10  out of  20"
## [1] "Deleted  0  out of  20"
## [1] "replicate # 29"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  9  out of  20"
## [1] "replicate # 30"
## [1] "Deleted  15  out of  20"
## [1] "Deleted  0  out of  20"
## [1] "replicate # 31"
```

```
## [1] "Deleted  0  out of  20"
## [1] "Deleted  0  out of  20"
## [1] "replicate # 32"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  0  out of  20"
## [1] "replicate # 33"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  15  out of  20"
## [1] "replicate # 34"
## [1] "Deleted  7  out of  20"
## [1] "Deleted  0  out of  20"
## [1] "replicate # 35"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  12  out of  20"
## [1] "replicate # 36"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  11  out of  20"
## [1] "replicate # 37"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  11  out of  20"
## [1] "replicate # 38"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  0  out of  20"
## [1] "replicate # 39"
## [1] "Deleted  12  out of  20"
## [1] "Deleted  0  out of  20"
## [1] "replicate # 40"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  0  out of  20"
## [1] "replicate # 41"
## [1] "Deleted  10  out of  20"
## [1] "Deleted  0  out of  20"
## [1] "replicate # 42"
## [1] "Deleted  11  out of  20"
## [1] "Deleted  6  out of  20"
## [1] "replicate # 43"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  11  out of  20"
## [1] "replicate # 44"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  10  out of  20"
## [1] "replicate # 45"
## [1] "Deleted  10  out of  20"
## [1] "Deleted  7  out of  20"
## [1] "replicate # 46"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  11  out of  20"
## [1] "replicate # 47"
## [1] "Deleted  5  out of  20"
## [1] "Deleted  0  out of  20"
## [1] "replicate # 48"
## [1] "Deleted  12  out of  20"
## [1] "Deleted  11  out of  20"
## [1] "replicate # 49"
```

```
## [1] "Deleted  5  out of  20"
## [1] "Deleted  11  out of  20"
## [1] "replicate # 50"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  0  out of  20"
## [1] "replicate # 51"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  0  out of  20"
## [1] "replicate # 52"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  0  out of  20"
## [1] "replicate # 53"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  10  out of  20"
## [1] "replicate # 54"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  0  out of  20"
## [1] "replicate # 55"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  0  out of  20"
## [1] "replicate # 56"
## [1] "Deleted  10  out of  20"
## [1] "Deleted  0  out of  20"
## [1] "replicate # 57"
## [1] "Deleted  10  out of  20"
## [1] "Deleted  10  out of  20"
## [1] "replicate # 58"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  9  out of  20"
## [1] "replicate # 59"
## [1] "Deleted  7  out of  20"
## [1] "Deleted  13  out of  20"
## [1] "replicate # 60"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  9  out of  20"
## [1] "replicate # 61"
## [1] "Deleted  13  out of  20"
## [1] "Deleted  0  out of  20"
## [1] "replicate # 62"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  0  out of  20"
## [1] "replicate # 63"
## [1] "Deleted  9  out of  20"
## [1] "Deleted  9  out of  20"
## [1] "replicate # 64"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  10  out of  20"
## [1] "replicate # 65"
## [1] "Deleted  12  out of  20"
## [1] "Deleted  10  out of  20"
## [1] "replicate # 66"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  10  out of  20"
## [1] "replicate # 67"
```

```
## [1] "Deleted  14  out of  20"
## [1] "Deleted  0  out of  20"
## [1] "replicate # 68"
## [1] "Deleted  14  out of  20"
## [1] "Deleted  13  out of  20"
## [1] "replicate # 69"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  0  out of  20"
## [1] "replicate # 70"
## [1] "Deleted  11  out of  20"
## [1] "Deleted  13  out of  20"
## [1] "replicate # 71"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  11  out of  20"
## [1] "replicate # 72"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  0  out of  20"
## [1] "replicate # 73"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  0  out of  20"
## [1] "replicate # 74"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  10  out of  20"
## [1] "replicate # 75"
## [1] "Deleted  8  out of  20"
## [1] "Deleted  0  out of  20"
## [1] "replicate # 76"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  0  out of  20"
## [1] "replicate # 77"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  9  out of  20"
## [1] "replicate # 78"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  9  out of  20"
## [1] "replicate # 79"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  0  out of  20"
## [1] "replicate # 80"
## [1] "Deleted  7  out of  20"
## [1] "Deleted  9  out of  20"
## [1] "replicate # 81"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  0  out of  20"
## [1] "replicate # 82"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  0  out of  20"
## [1] "replicate # 83"
## [1] "Deleted  13  out of  20"
## [1] "Deleted  11  out of  20"
## [1] "replicate # 84"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  13  out of  20"
## [1] "replicate # 85"
```

```
## [1] "Deleted  6  out of  20"
## [1] "Deleted  11  out of  20"
## [1] "replicate # 86"
## [1] "Deleted  8  out of  20"
## [1] "Deleted  7  out of  20"
## [1] "replicate # 87"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  0  out of  20"
## [1] "replicate # 88"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  0  out of  20"
## [1] "replicate # 89"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  12  out of  20"
## [1] "replicate # 90"
## [1] "Deleted  10  out of  20"
## [1] "Deleted  0  out of  20"
## [1] "replicate # 91"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  9  out of  20"
## [1] "replicate # 92"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  13  out of  20"
## [1] "replicate # 93"
## [1] "Deleted  10  out of  20"
## [1] "Deleted  0  out of  20"
## [1] "replicate # 94"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  10  out of  20"
## [1] "replicate # 95"
## [1] "Deleted  12  out of  20"
## [1] "Deleted  0  out of  20"
## [1] "replicate # 96"
## [1] "Deleted  10  out of  20"
## [1] "Deleted  0  out of  20"
## [1] "replicate # 97"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  0  out of  20"
## [1] "replicate # 98"
## [1] "Deleted  11  out of  20"
## [1] "Deleted  11  out of  20"
## [1] "replicate # 99"
## [1] "Deleted  0  out of  20"
## [1] "Deleted  0  out of  20"
## [1] "replicate # 100"
## [1] "Deleted  11  out of  20"
## [1] "Deleted  0  out of  20"
```

We can get the ranking in the first run. It is only necessary to run this line:

```
jack.Ranking.100[[1]][,vector]
```

```
##  [1] N K F C G B M H I D
## Levels: B C D F G H I K M N
```

4. Convert the initial values to a ranking

The initial ranking for Multitaxon is assigned as follows:

```
initial.Ranking <-  as.data.frame(Rank.Indices(Multi.Index.Calc(Multitaxon1)))
```

```
## [1] "Deleted  0  out of  20"
## [1] "Deleted  0  out of  20"
```

```
initial.Ranking[,vector]
```

```
##  [1] N K F C B G M D H I
## Levels: B C D F G H I K M N
```

5. Compare the whole ranking for the best index, 100 times.

```
jack.Ranking.100.comparison <- NULL

for (i in 1:100){
  jack.Ranking.100.comparison[i] <- all.equal(initial.Ranking[,vector],
                                              jack.Ranking.100[[i]][,vector])
}
```

Compute the number of hits or the Jackknife replicate ability to recover the initial ranking:

```
length(which(jack.Ranking.100.comparison==TRUE))
```

```
## [1] 35
```

As in the first example, get an estimation of the error.

```
length(which(as.data.frame(jack.Ranking.100.comparison)!=TRUE))
```

```
## [1] 65
```

And with this instruction we can identify the type of error.

```
jack.Mismatch <- jack.Ranking.100.comparison[jack.Ranking.100.comparison!=TRUE]

count.Jack.Mismatch <- gsub(" string mismatches","",jack.Mismatch)

count.Jack.Mismatch <- as.numeric(count.Jack.Mismatch)
```

This code computes the distribution of the error, which will rather be bell shaped.

```
hist(sort(count.Jack.Mismatch), main="Figure 6. Histogram of Error Distribution",  xlab= "Jack Mistmatc
```

Figure 6. Histogram of Error Distribution