Homework 5

Computational Biology course
Handed out: 29.11.2021
Due date: 13.12.2021

Theory questions

We encourage you to answer these questions only after you have completed the programming task. Please keep your answers short, most of the questions can be answered in one or two sentences. Submit your answers in a pdf file named following the format Lastname_Firstname_HW5.pdf.

1. Why can't Fisher's exact test be directly used to check for correlation on the values of discrete traits at the tips of a phylogenetic tree?

2. In the independent contrast method, the contrasts Z_k are mutually independent and have identical variance. Explain why this is done.

3. Explain which steps of the algorithm focus on defining a set of independent variables, and which steps ensure that these have identical variance.

4. You are studying the length of canine teeth (i.e. "eye teeth" or "Echzähne") and the length of claws among apes. What could make you think that they evolved in an uncorrelated way, or on the contrary, that they evolved in a correlated way ? In which case would the two traits show a correlation? In which case would they show correlated normalized contrasts?

5. A strategy for comparing discrete characters while accounting for relatedness between individuals was presented in lecture 8. Imagine you want to perform a discrete character comparison using this strategy on two discrete traits that you are able to accurately observe on individuals at the tips of a known tree. What problem might you encounter when trying to fill in the contingency table of character changes?

Programming task

Task description

In this homework you will implement the independent contrast method which allows one to determine correlations between continuous traits such as height and weight between individuals/species that have shared evolutionary history. Your script should compute the *normalised independent contrasts* for trait values that evolved on a phylogenetic tree. In addition, you should perform linear regression on these normalised independent contrasts, as well as on the raw trait data at the tips, to see whether an analysis that disregards common evolutionary history would lead to a different conclusion regarding trait correlation. We provide the function `plot_data_and_regression()`, which should be called in the final function `get_trait_correlation()` to plot the linear regression line on the raw data in the left graph and on the contrasts in the right graph. The algorithm for computing all the necessary values to get the normalised contrasts is presented in pseudo-code form in the section Independent contrasts algorithm pseudocode.

As input your code should take:

1. `newick_tree`: a tree in newick format, e.g. `"(orangutan:10.25,(human:5.5,chimp:5.5):4.75);"`.
2. `traits_at_tip`: a list with two vectors, where the first vector contains the values of the first trait for the tips in the newick tree, and the second contains the values for the second trait, e.g. `traits_at_tip = list(trait1 = c(orangutan=2, human=1, chimp=4), trait2 = c(orangutan=5, human=3, chimp=9))`. The trait names should correspond to the tip names in the tree.

You should calculate the normalised independent contrasts given the input tree and the trait values at the tips. To do so, you will recursively walk through the tree starting at the root using the function called `get_contrasts_in_subtree()`. For each internal node including the root you should first compute all the necessary values for its children by calling `get_contrasts_in_subtree()` for the child nodes, and then use the computed values to calculate the values for the internal node in question. The recursive process will stop when you reach a tip in the tree, as the trait values at the tips are known, no contrast can be computed and the branch lengths do not need to be corrected.

Each time you reach an internal node k for which the values at the child nodes i and l are known, you should calculate the following values (see illustration in Figure 1) and store them appropriately:

1. the corrected branch length of the branch leading to the node k, i.e.

$$t'_k = \frac{t'_i t'_l}{t'_i + t'_l} + t_k$$

2. the value of both traits at this node, i.e. for each trait you will have to apply

the formula:

$$X_k^j = X_i^j \frac{t'_l}{t'_i + t'_l} + X_l^j \frac{t'_i}{t'_i + t'_l}$$

3. the normalised independent contrast for both traits at this node, i.e. for each trait you will have to apply the formula:

$$Z_k^j = \frac{X_i^j - X_l^j}{\sqrt{t'_i + t'_l}}$$

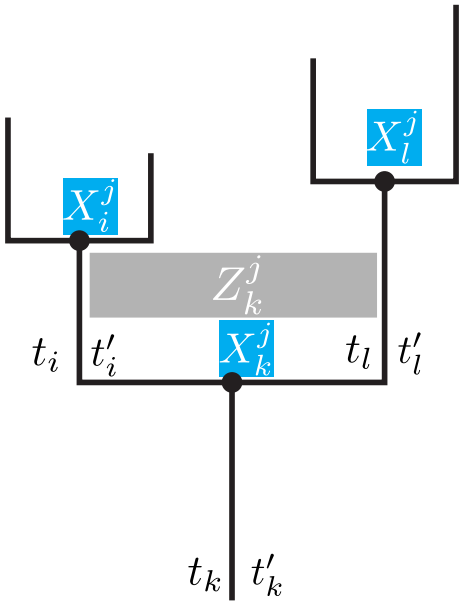

Figure 1: Sketch of the internal node k with original and corrected branch lengths (t_k and t'_k respectively), trait X_k^j and contrast Z_k^j.

As you traverse the tree you will use the functions:
(i) `calculate_trait_and_contrast()`, and
(ii) `calculate_corrected_branch_length()` to compute the values that you need.

The computed values should be stored in the list `description` containing the following three data structures:

- A matrix with the trait values for all nodes. This matrix is initialised with the trait values at the tips and `NA` for the internal nodes by the provided function `initialize_description()` and should be filled in recursively by your code. The rows in the matrix correspond to the node number in the tree structure generated by the two provided commands `tree = read.tree(text = newick_tree)` and `tree = reorder(tree, order = "cladewise")` from the newick tree.
- A matrix with the normalised independent contrasts. This matrix is initialised with `NA` by the provided function `initialize_description()`

and should be filled in recursively by your code. The rows in the matrix correspond to the node number in the tree structure generated by the two provided commands `tree = read.tree(text = newick_tree)` and `tree = reorder(tree, order = "cladewise")` from the newick tree. As contrasts are only calculated at internal nodes, the first rows representing the tips will remain `NA`.

- A vector with the corrected branch lengths. This vector is initialised with the original branch lengths by the provided function `initialize_description()` and should be updated with the corrected branch lengths recursively. The branch lengths are stored in a way that you can get the branch length leading to node k from its parent by simply calling `corrected_branch_lengths[k]`[1].

In the final function `get_trait_correlation()` you will calculate the contrasts for the whole tree and perform linear regression on the contrasts as well as on the original data. Linear regression in `R` is performed with the function `lm()` and requires a data frame as input (see section Linear regression in R for a detailed description). The function `plot_data_and_regression()` plots the two graphs for the raw data and the contrasts. This function is provided by us and helps you interpret the data as well as the independent contrasts.

Your script should follow the structure that we have laid out in the skeleton script called `CB_HW5_ComparativeMethods_skeleton.R`. This skeleton splits the assignment into smaller functions that are necessary to compute the final result. Each function is listed with the input parameters that it requires and the output that it must generate, as well as a short description of what it does. As in previous assignments, you only have to fill in the missing code (marked with `# ???`).

Remember to pay careful attention to the comments in the skeleton code, as these give many useful hints on how to go about implementing the algorithm. It is a very good idea to try to read through all of the skeleton code, including these comments, before starting with your implementation.

> Please do not change the structure of the code and the inputs and outputs of functions. The code will be tested and graded automatically and any structural changes will result in your code failing the tests.

To help you understand the skeleton structure, we have provided a cross-reference table showing which functions in the skeleton should use which others (see Table 1).

[1] Keep in mind that you only need the `corrected_branch_lengths` data structure to compute all the corrected branch lengths, you do not need to store the uncorrected values separately. After initialization, the vector will contain the original branch lengths which you should update as you walk through the tree. If your tree walking is implemented correctly, i.e. if you compute the values for node children for each node before computing anything at the node itself, you will have the corrected branch lengths for all nodes below the current node and the original branch length for the current node. After computing the corrected branch length you should simply replace the value in the vector.

Function name	Uses

`get_trait_correlation`	`initialize_description`
	`get_contrasts_in_subtree`
	`plot_data_and_regression`

`get_contrasts_in_subtree`	`get_node_children`
	`get_contrasts_in_subtree`
	`calculate_corrected_branch_length`
	`calculate_trait_and_contrast`

`calculate_trait_and_contrast`	`calculate_trait_at_internal_node` `calculate_contrast`

Table 1: Function cross-reference table

Testing

We provide you with a test suite which is very similar to the one we will use to grade the homework. You can use it to help verify that your code is functioning properly. However, be aware that you also need to do your own testing, as the test suite does not cover all the possible cases which might not be sufficient to ensure you code is bug-free. Moreover, note that the tests used to grade your final submission will differ from these, so you should make sure that your code behaves properly when different input values are used. A good first test is to try the trait values and the tree used in the pen and paper exercise in the tutorial to see if you get the same values that you computed by hand.

Independent contrasts algorithm pseudocode

τ: given phylogenetic tree;

k: a node in tree τ;

obs_k^j: value of trait j at the tip k;

t_k: branch length leading to node k;

t'_k: corrected branch length leading to node k;

X_k^j: value of trait j at node k;

Z_k^j: value of the contrast for trait j at the node k (only defined for internal nodes);

Contrast computation for node k for trait j

Data: Node n, tree τ, trait values at the tips of the tree obs^j

Result: t'_k, X^j_k, Z^j_k

if *k is a tip* **then**

$t'_k \leftarrow t_k$;

$X^j_k \leftarrow \text{obs}^j_k$;

$Z^j_k \leftarrow \text{NA}$;

else

Compute t'_i, X^j_i, Z^j_i where i is the first child of k;

Compute t'_l, X^j_l, Z^j_l where l is the second child of k;

$t'_k \leftarrow \frac{t'_i t'_l}{t'_i + t'_l} + t_k$;

$X^j_k \leftarrow X^j_i \frac{t'_l}{t'_i + t'_l} + X^j_l \frac{t'_i}{t'_i + t'_l}$;

$Z^j_k \leftarrow \frac{X^j_i - X^j_l}{\sqrt{t'_i + t'_l}}$;

end

Additional materials

Required Packages

This homework assignment requires that the following package is installed on your R system:

ape : a library providing data types and methods suitable for phylogenetics.

To install the package, enter the following command at the R command line:

```
install.packages("ape")
```

Remember that you only need to install a package once and then you can load it for use in you code using the `library("PackageName")`. The homework skeleton loads the aforementioned package for you.

Useful functions

We provide a list of R functions that you may find useful when writing your code (Table 2).

R function/operator	Description
`sqrt(value)`	Get the square root of the given `value`.
`data.frame(colname1 = vector1, colname2 = vector2)`	Creates a data frame with two columns named `colname1` and `colname2`, where each column is populated by the values in the given vectors, which have to be of the same length.
`lm(colname1 ~ colname2, data)`	Perform linear regression on a data frame `data` with columns titled `column1` and `column2`.
`summary(linreg)`	Gets the summary of the linear regression results (for more information see section Linear regression in R).

Table 2: Useful R functions.

Linear regression in R

First and foremost, R is a platform for performing statistical analyses. Therefore, fitting a linear function to a set of data is one of the things that can be done very easily.

The function R provides for performing linear regression is `lm()`. It can be used to fit a wide variety of linear models. We are specifically interested in finding the slope (or gradient) a and intercept b, so that the sum of the squares of the differences between y_i and $ax_i + b$ for all pairs (x_i, y_i) is minimized. To illustrate this, let us define a pair of vectors named `xvec` and `yvec` containing some example x and y data points:

```
xvec = c(1.390175, 2.437715, 3.325896, 4.214967, 5.692393)
yvec = c(3.817900, 6.399221, 9.355539, 12.10676, 15.20896)
```

The following command will perform the optimization and store the results in a variable named `res` [2]:

```
res = lm(y ~ x, data.frame(x = xvec, y = yvec))
```

The resulting variable now contains a lot of information including both the values of the fitting coefficients a and b and the goodness of fit. In order to access these more easily, R also provides a helper function named `summary()` which we will use to help access this information:

```
s = summary(res)
```

You can view this summary information by simply entering the name of the variable in which it is stored on the command line and pressing ENTER:

[2]Notice that the first argument to lm() specifies the "formula" to fit to the data. In this case a linear relation is assumed: $y \propto x$.

```
>s
Call:
lm(formula = y ~ x, data = data.frame(x = xvec, y = yvec))

Residuals:
       1        2        3        4        5
-0.05708 -0.32647  0.21280  0.54456 -0.37381

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.09185    0.50022   0.184 0.866023
x            2.72134    0.13455  20.225 0.000264 ***
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05
 '.' 0.1 ' ' 1

Residual standard error: 0.444 on 3 degrees of freedom
Multiple R-squared:  0.9927,    Adjusted R-squared:  0.9903
F-statistic: 409.1 on 1 and 3 DF,  p-value: 0.0002642
```

The intercept, b, and slope, a, are the values in the first and second rows respectively of the first column of the output under `Coefficients`.

In the assignment, you will need to assign these values, the R^2 value and the p value to variables. The following table describes how to access each of these values using the variable containing the summary:

Quantity	Command
Slope (a)	`s$coefficients[2,1]`
Intercept (b)	`s$coefficients[1,1]`
p-value	`s$coefficients[2,4]`
R^2 value	`s$r.squared`