"Computation of Brownian Motion in Python, a model to study evolution of polymorphism"

Rémy Morier-Genoud

Supervisors: Anna Kostikova, Nicolas Salamin
In Sven Bergmann's Class: "Solving biological problem that requires Math (2012)"

References

Butler M. A., King A. A., 2004. "Phylogenetic Comparative Analysis: A Modelling Approach for Adaptive Evolution", p 683 *in* The American Naturalist vol146 N°6.

Appendix from M. A. Butler and A. A. King, "Phylogenetic Comparative Analysis: A Modelling Approach for Adaptive Evolution".

Walsh B., 2004. "Markov Chain Monte Carlo and Gibbs Sampling". Lecture Notes for EEB 581.

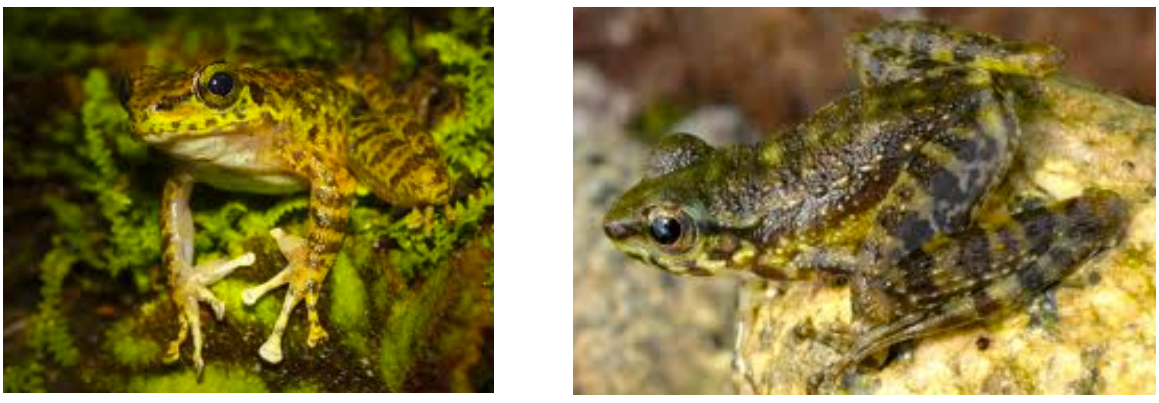
Summary

Brownian Motion implementation in Python

Implementation of Brownian Motion in *Python* should be divided in three main steps. First we need to implement the data: A phylogenetic tree and phenotypical traits measurements. Secondly we have to calculate a Variance & Covariance matrix based on distance from the tree. Thirdly we could optimize the Likelihood, which is calculated using a formula including Variance & Covariance matrix multiplication and the given traits (Butler et al. 2004), to estimated the unknown parameters θ & σ.

Tools

- Phylogenetic tree in *Newick* format is saved on a separate file.
- Phenotypical traits values are saved in a separated file in *csv* format.
- Variance & Covariance matrix calculation is done with a new algorithm.
- Likelihood estimation is done using *Numpy* Library.
- Likelihood optimization is done using *Scipy* Library.



Amolops marmoratus (left) and Amolops larutensis (right), www.google.ch/image.

Project's main steps summary & Python Code explanation

Data requires to run the model (Step 1)

Ultrametric Tree (separated file save in Newick format)

Implementation of the BM model in *Python* requires some data of specific forms. First we need a phylogenetic ultrametric tree in *Newick* format, on which the calculation of a matrix of Variance & Covariance will be based. The *Newick* format is a standardized format for phylogenetic tree, on which we could work with *Dendropy* library on *Python,* or easily elaborate some new algortithms to read this format. In this experiment, we work with the following tree (A Lineage of Chinese frogs, *Amolops sp.*) called *"frogs-2.tre"*:

```
((((larute:0.288231,spinap:0.111924)0.618000:0.017046,(hongk:0.181
915,(ricket:0.011689,wuyien:0.030254)0.897000:0.039276)0.942000:0.
047573)0.710000:0.03377,(marmor:0.21305,panhai:0.155244)0.893000:0
.067422)0.852000:0.027263,(tuberod:0.007844,loloen:0.032906)0.7820
00:0.014999)0.964000;
```

Note that actually, the implemented tree needs to be save in a file on one single strip, ended with a "; " character to be treated without problems by the algorithms.

Phenotypical trait value (separated file save in Csv format)

Secondly, we need some phenotypical traits values. They are simply measurements for a given trait in each actual taxon of the previously given tree. These numerical values are stocked on a *.csv* file, and can be easily simulated on *R* with the *rTraitCont()* function of *Ape* library. We then write a small function, *Read_traits()*, which converts the data of this .csv file in a *numpy.array()* in *Python* (in the same ordered as *list_taxa*). The phenotypical traits values should be, by example, the mean annual Temperature (*BioclimI*) in different regions of China where live each species of Amolops frogs. The array will look as the following:

```
numpy.array([23.2, 21.1, 20.2, 17.1, 17.6, 25.5, 26.1, 10.8, 10.8])
```

Building the Variance & Covariance matrix using a new algorithm (Step 2)

Get information from the Newick format tree

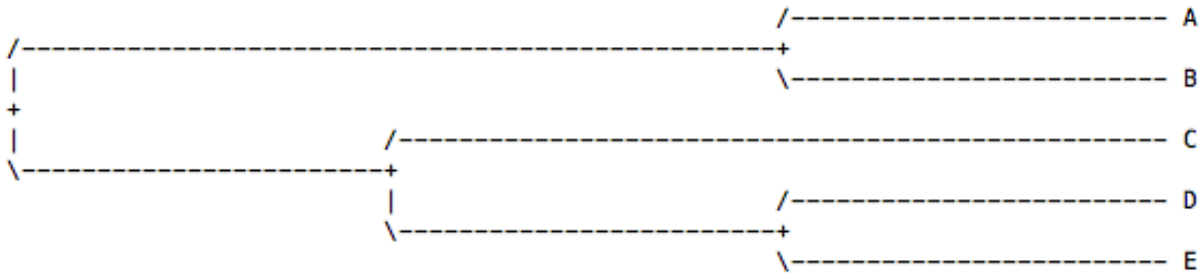
After data implementation (Step 1), we need to get some information from the tree. We want the length of edges shared by pairs of taxon in order to build the Variance & Covariance matrix. There were two options to calculate these values. Firstly we can use *Dendropy* library on *Python. Dendropy* is a module specialised on reading and calculation of phylogenetic trees. But the way the function of this library are shown and explained on tutorial, and the fact that no *Dendropy*'s functions already available seem to directly lead to the wanted matrix make us choose a second option. We then write a new algorithm to get the metric data from the tree in *Newick* format and compute them in a matrix (type list of list, in a way we can converted it in numpy array later).

New Algorithm, First part: Getting information from the tree

Here is given a simple example of a tree in *Newick* format:

((A:0.1,B:0.1)AncAB:0.6,(C:0.3,(D:0.1,E:0.1)AncDE:0.2)AncCDE:0.4)AncRoot:0.9;

The *Newick* format is built with names ("A", "B", "AncAB",etc), edges length, and some characters ("(", ")", ":", "," and ";") which structure the information in a standardize way and encoded the phylogenetic position of each taxa in the tree. Here is given the corresponding plot, visualize with *Dendropy*'s function *t.pint_plot()* (were *t* is the name of the example tree):



As we import the tree in *Newick* format in *Python* with the functions *open(file, "r")* and *t.readline()*, the imported tree is seen as a string element. So in a first time, we defined a list, *list_Parentheses*, in which brackets postitions in this string have to been stocked. Then we defined 2 other lists, *list_taxa*, in which the extracted name of each taxon will be stocked, and *list_info_taxa*, in which the corresponding position of the taxon on the string (marked by ":") will be stocked.

We associate each couple of brackets (an opening bracket and its ending partner) at their position in the string. Opening brackets are find screening the string for *"("* characters, while their ending partner are find using a simple counter varialbe, *P_open*, which counts the number of brackets still open. This counter begin at zero when an opening brackets is found, increase of one each time a new bracket is open and decrease of one each time a closing brackets is found. So it falls to zero when the closing bracket linked to the opening bracket found with screening is found. Then the position of the couple is saved in *list_Parentheses* and the algorithm go on screening for the next opening bracket.

Once *list_Parenthses* is filled with each couple of brackets position, we extract the names of taxa contained between each couple of brackets and add them after corresponding brackets position in *list_Parentheses*. A list, *list_taxa*, containing the names of all taxa in the order they are found in the string (for example: ['A', 'B', 'C', 'D', 'E']) is made in parallel, using the function *Is_taxa()* which check if the given name is already presents in *list_taxa* to avoid splitting. Each time a new taxon is added in *list_taxa*, the position of ":" character following the name of the taxon in the string is reported in one other list, *list_info_taxa*. The extraction of names is made cutting the string, knowing that each name is always preceded by a "(" or "," character and followed by ":" character, position of which is reported in *list_info_taxa*.

Example of *list_Parentheses* (for the given example tree):

```
[[1,13,['A','B']], [29,63,['C','D','E']], [36,48,['D', 'E']]]
```

Note that *list_Parentheses* is composed of smaller lists, were the first element is the opening bracket position in the string, the second element the ending bracket position, and the third element is a list filled with the name of each taxon presents between this two brackets.

New Algorithm, Second Part: Edges length shared by 2 taxa

Now that we have listed each taxon of the tree and stocked some information on the structure of the tree, we want to calculate the distance between 2 taxa. The main idea is to sum each edges of common ancestor to the two taxa given. In *Newick* tree, the edges value for each ancestor is written (example: (A:0.1,B:0.1)AncAB:0.6 means that the "A" and "B" have a common ancestor "AncAB", and that the distance between AncAB and its own ancestor (the preceding nodes on the tree) is equal to 0.6). An interesting characteristic of the *Newick* format is that each time that two taxa are contained between the same brackets, they shared a common ancestor, which is given right next to the ending bracket. So we can easily take two different taxa, search for each brackets where they are together, and sum the ancestor edge to obtain the distance shared by these two taxa. Indeed, the function *Check_for_taxa()* return a list containing the position in *list_Parentheses* of each couple of brackets containing both taxa. Secondly, a function *Find_Common_Edge_Distance()* catch each ancestor edge value and return the sum of all these values. As for the names of taxa before, the string must be cut at the right place to catch the value. So it begins to screen, at each ending brackets returned by *Check_for_taxa(),* for ":" character, which is the character right before the first character of the value. It cuts from this point to the next ")" or "," character. The cut strings must be change in float and then can be sum.

If the two given taxa are identical, which will be the case for the Variance & Covariance matrix's diagonal, a single value corresponding to the distance between the last node and the taxon must be added. If this is the case, the function *Taxa_ItSelf()* screen for the position of the given taxon in *list_taxa* and get the ":" position in the string in *list_info_taxa* at the same indice that *list_taxa*. Exactly as describe for *Find_Common_Edge_Distance()*, it cut the value right after the ":" character in the string and convert it to a float, in a way it should be added to *Find_Common_Edge_Distance()* restult to obtain the total distance of the edge of this taxon (because the distance shared by a taxon and itself is equal to its total edge length).

Example of shared distance: between D & E: 0.4+0.2 = 0.6; between D & D 0.6+0.1 = 0.7

New Algorithm, Third Part: Variance & Covariance matrix building

At this point, we are able to build the Variance & Covariance matrix easily. This matrix is a list of lists, where columns are small lists nested on a bigger list, *matrix*. Each cases of this matrix must be filled with the distance shared by two taxa, in a way that all combinations of taxa are done. The diagonal corresponds to total edge lengths for each taxon. To fill the matrix, we test each taxon with all the taxa of *list_taxa*, following the order of this list, which has to be the same order as the given array of phenotypical trait values. The *Calc_VARCOVAR()* function tests if the two given taxa are identical, and in this case calls *Find_Common_Edge_Distance()* and sums its result with *Taxa_ItSelf()* result, although only *Find_Common_Edge_Distance()* is required if the two taxa are different.

Example of Variance & Covariance matrix (for given example tree):

```
[[0.7, 0.6, 0, 0, 0], [0.6, 0.7, 0, 0, 0], [0, 0, 0.7, 0.4, 0.4],
[0, 0, 0.4, 0.7, 0.6], [0, 0, 0.4, 0.6, 0.7]]
```

Likelihood equation optimization with Numpy & Scipy libraries (Step 3)

Write a function which return Likelihood for given parameters

On the bases of the Variance & Covariance matrix calculated on the tree (Input1) and a given numpy.array (Input2) containing the phenotypical traits for each taxa on the tree, wee can estimated the strength of the drift (σ) which explain at best the evolutive story of the lineage and the phenotypical trait value of the last common ancestor (at the root of the tree) of this lineage (θ_{anc}). To estimate these two parameters, we have to check for their values minimizing the following Likelihood equation:

$$\text{Likelihood} = -\frac{(x-W0)\prime * V * (x-W0)}{2} - \frac{n * log\,(2\pi)}{2} - \frac{log(det(V) * \square)}{2}$$

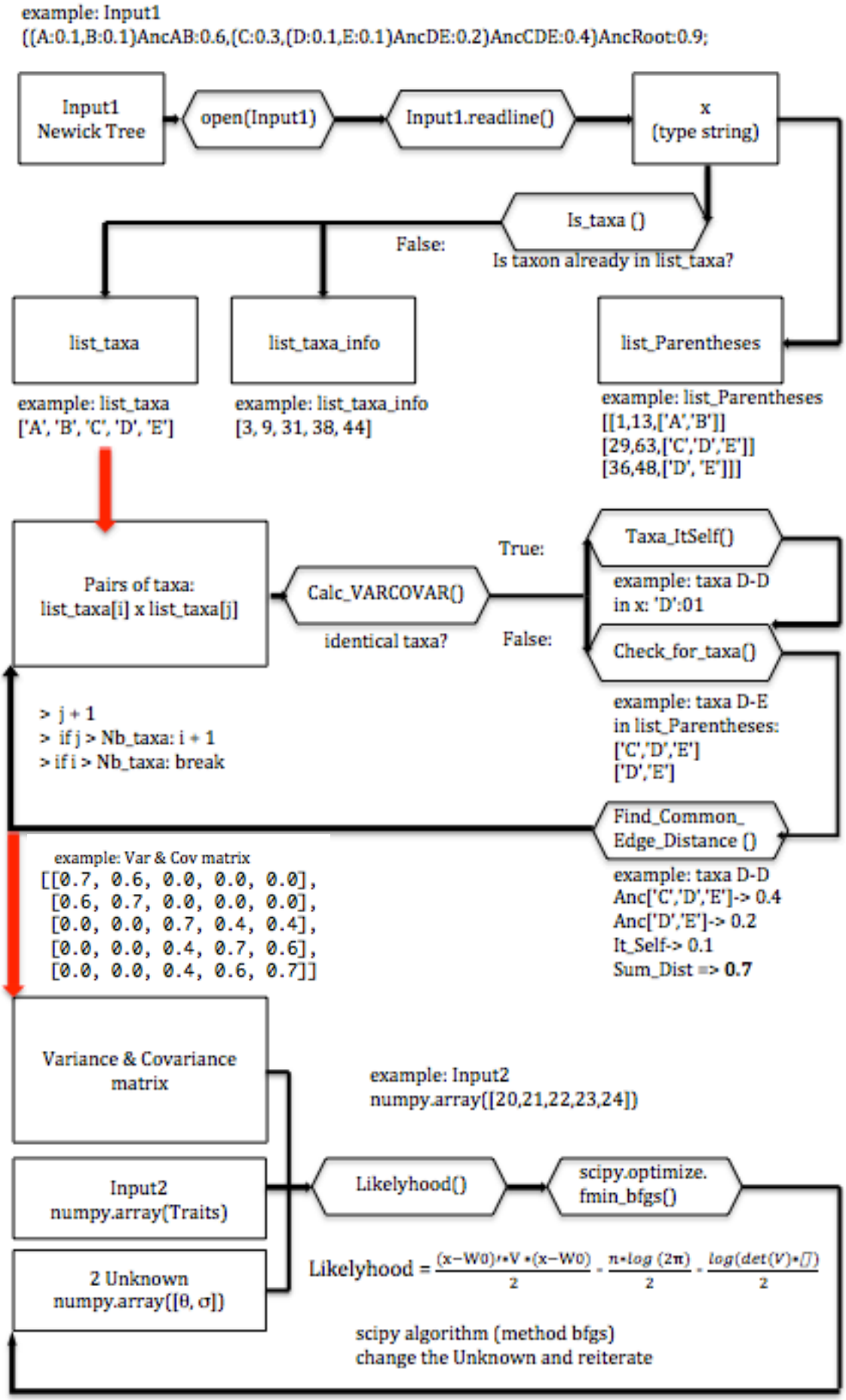
We write a function, *Likelihood()*, which return this Likelihood value estimated for a given θ and σ.

Optimize the Likelihood to estimate the 2 Unknown (θ & σ)

We use *scipy* library in *Python* to find the θ and σ which minimize the Likelihood returned by *Likelihood()* function. The function used in *scipy* library is *fmin_bfgs()*, based on Broyden-Fletcher-Goldfarb-Shanno algorithm.

Amolops ricketti, www.google.ch/image

Next page: General scheme of the program with examples


example: Input1
((A:0.1,B:0.1)AncAB:0.6,(C:0.3,(D:0.1,E:0.1)AncDE:0.2)AncCDE:0.4)AncRoot:0.9;
Input1
Newick Tree
open(Input1)
Input1.readline()
x
(type string)
Is_taxa ()
False:
Is taxon already in list_taxa?
list_taxa
list_taxa_info
list_Parentheses
example: list_taxa
['A', 'B', 'C', 'D', 'E']
example: list_taxa_info
[3, 9, 31, 38, 44]
example: list_Parentheses
[[1,13,['A','B']]
[29,63,['C','D','E']]
[36,48,['D', 'E']]]
Pairs of taxa:
list_taxa[i] x list_taxa[j]
Calc_VARCOVAR()
identical taxa?
True:
False:
Taxa_ItSelf()
example: taxa D-D
in x: 'D':01
Check_for_taxa()
example: taxa D-E
in list_Parentheses:
['C','D','E']
['D','E']
> j + 1
> if j > Nb_taxa: i + 1
> if i > Nb_taxa: break
Find_Common_
Edge_Distance ()
example: taxa D-D
Anc['C','D','E']-> 0.4
Anc['D','E']-> 0.2
It_Self-> 0.1
Sum_Dist => 0.7
example: Var & Cov matrix
[[0.7, 0.6, 0.0, 0.0, 0.0],
[0.6, 0.7, 0.0, 0.0, 0.0],
[0.0, 0.0, 0.7, 0.4, 0.4],
[0.0, 0.0, 0.4, 0.7, 0.6],
[0.0, 0.0, 0.4, 0.6, 0.7]]
Variance & Covariance
matrix
example: Input2
numpy.array([20,21,22,23,24])
Input2
numpy.array(Traits)
Likelyhood()
scipy.optimize.
fmin_bfgs()
2 Unknown
numpy.array([θ, σ])
Likelyhood = (x−W0)'•V•(x−W0)/2 − n•log(2π)/2 − log(det(V)•∏)/2
scipy algorithm (method bfgs)
change the Unknown and reiterate