Juraj Bergman

Python_MKT

Python version 3.3.2

e-mail: jurajbergman@hotmail.com
7/9/2013

Python_MKT

Introduction

Positive selection drives evolution towards phenotypes with advantageous features given particular environmental conditions. These advantageous features are better-known as adaptive mutations which can spread through the population because of their 'usefulness', thus allowing the survival of a species in a certain environment and time frame. The rate of this adaptive evolution can be quantified by examining polymorphism levels of the coding regions of genes [1]. Given the availability of DNA sequencing data, this can be easily achieved as shown by Smith & Eyre-Walker [2], who found that 45% of amino acid changes between *Drosophila simulans* and *Drosophila yakuba* were due to positive selection.

This module consists of various functions designed to quantify the effects of adaptive evolution and is designed using Python version 3.3.2. The main function of this module is the mkt() function which is an implemented McDonald-Kreitman test [3].

List of functions

mkt(list_of_list_of_list_of_seq)

This is the main function of the module; the McDonald-Kretimean test (MKT). The MKT is usually used to assess the level of positive selection acting between two closely related species. This is achieved by comparing polymorphism levels, as well as levels of fixed nucleotide differences (divergence) of homologous genes between the two species. For the MKT to applicable, it is necessary to have multiple aligned sequences of the gene of interest for at least one of the species (in order to be able to assess the polymorphism levels of the gene). The input of the function is a list of list of list of sequences. The sequences of genes have to be aligned strings before they are they are input into the function. The result of this function is a 2×2 table which shows the number of synonymous and non-synonymous polymorphisms and substitutions (divergence) as well as the the parameter alpha which signifies the proportion of adaptive mutations:

	Polymorphism	Divergence
Synonymous	P_s	D_s
Non-synonymous	P_n	D_n
Alpha = ???		

The function can be used to compare sequences of only one homologous gene, in which case alpha is calculated using the equation (as described by Smith NGC & Eyre-Walker [2]):

$$\alpha = 1 - \frac{D_s P_n}{D_n P_s}$$

or multiple homologous genes, in which case the D_s, P_n, D_n and P_s parameteres are pooled and the alpha parameter is calculated using the equation (as described by Smith NGC & Eyre-Walker [2]):

$$\overline{\alpha} = 1 - \frac{\overline{D_s}}{\overline{D_n}} \overline{\left(\frac{P_n}{P_s + 1}\right)}$$

Example 1. Comparison of one homologous sequence.

```
>>> mkt([[['AAAGGGCCCTTG', 'AAGGGGCGCGTG'],['ACAGGGCCCGGA']]])
               Polymorphism     Divergence
Synonymous     1                1
Non-synonymous 2                2
Alpha = 0.0
```

Notice that the innermost brackets ['AAAGGGCCCTTG', 'AAGGGGCGCGTG'] and ['ACAGGGCCCGGA'] contain aligned sequences of the same homologous gene (gene_a). The bracket ['AAAGGGCCCTTG', 'AAGGGGCGCGTG'] contains two versions of gene_a belonging to species_1 and the bracket ['ACAGGGCCCGGA'] contains a single version of gene_a belonging to species_2, so the input is analogous to:
mkt([[['species_1_gene_a_version_1',species_1_gene_a_version_2'],['species_2_gene_a_version_1']]]).

Example 2. Comparison of multiple homologous sequences.

```
>>> mkt([[['AAAGGGCCCTTG', 'AAGGGGCGCGTG'],['ACAGGGCCCGGA']],
     [['ATGGGA', 'AGAGGA'],['ATGCGA', 'ATGGGG']]])
               Polymorphism     Divergence
Synonymous     3                2
Non-synonymous 4                2
Alpha = 0.16666666666666674
```

The parameters D_s, P_n, D_n and P_s are now pooled and the the alpha represents the mean proportion of adaptive mutations. The input is analogous to:
mkt([[['species_1_gene_a_version_1',species_1_gene_a_version_2'],['species_2_gene_a_version_1']],[['species_1_gene_b_version_1',species_1_gene_b_version_2'],['species_2_gene_b_version_1',species_2_gene_b_version_2']]])

poly_ratio(list_of_seq)

Returns the total number of synonymous and non-synonymous mutations between a list of aligned coding sequences. The first value of the output list is the total number of synonymous mutations and the second value of the output list is the total number of non-synonymous mutations.

Example 1.

```
>>> poly_ratio(['ATGCACAAAGGG', 'ATGCTCAGAGGG', 'ATGCACAAGGGG'])
[1, 2]
```

div_ratio(list_of_seq_1, list_of_seq_2)

Returns the total number of synonymous and non-synonymous divergent changes between two lists of aligned sequences. list_of_seq_1 belongs to species_1 and list_of_seq_2 belongs to species_2. The first value of the output list is the total number of synonymous divergent changes and the second value of the output list is the total number of non-synonymous divergent changes.

Example 1.

```
>>> div_ratio(['ATGCACAAAGGG', 'ATGCTCAGAGGG', 'ATGCACAAGGGG'],
     ['ATGCACGGAGGG', 'ATGTTAAGGGGG'])
[1, 0]
```

codon_diff(list_of_seq)

Given a list of aligned sequences, it returns a dictionary with keys as codon positions and values as the variants of that codon.

Example 1.

```
>>> codon_diff(['AAAGGGCCC', 'AGAGGGCGC', 'AACGGGCCG','ACAGGGCCC'])
{0: ['AAA', 'AGA', 'AAC', 'ACA'], 1: ['GGG'], 2: ['CCC', 'CGC',
'CCG']}
```

multi_short_path(list_of_codon)

Returns the shortest mutation pathway between a list of codons as a list of two integers. The first integer of the list is the amount of synonymous mutations and the second is the number of non-synonymous mutations required for the most probable mutation pathway between codons. The pathway with the least overall amount of nucleotide substitutions (synonymous + non-synonymous) and the least amount of non-synonymous mutations is considered as the most probable. Codons with incomplete sequences are excluded from the final computation. The computation is made using Kruskal's alogrithm.

Example 1.

```
>>> multi_short_path(['AAA', 'AGA', 'AAG', 'CCC'])
[3, 2]
```

short_path(codon_1, codon_2)

Returns the shortest mutation pathway between two codons as a list of two integers. The first number of the list is the amount of synonymous mutations and the second is the number of non-synonymous mutations required for the most probable mutation pathway between two codons. The pathway with the least overall amount of nucleotide substitutions (synonymous + non-synonymous) and the least amount of non-synonymous mutations is considered as the most probable.

Example 1.

```
>>> short_path('GTG', 'AAA')
[1, 2]
```

helper(list_of_list)

Helper function for the div_ratio() function. Sorts a list of lists of two integers. The inner lists represent the number synonymous and non-synonymous mutations, where the first integer of an inner list is the amount of synonymous mutations and the second is the number of non-synonymous mutations required for the most probable mutation pathway between two codons. The sorting is done by the following hieararchy: [1, 0], [0, 1], [2, 0], [1, 1], [0, 2], [3, 0], [2, 1], [1, 2], [0, 3]. This hierarchy represents all possible substitutions between codons regarding the amount of synonymous and non-synonynmous mutations required. The first list of the

hierarchy is the [1, 0] list which represents one synonymous and zero non-synonymous mutations. The hierarchy is continued giving priority to the list with the least amount of overall nucleotide substitutions (synonynmous + non-synonymous) and the least amount of non-synonymous substitutions.

Example 1.

```
>>> helper([[0, 1], [1, 0], [2, 0], [0, 2], [1, 0], [1, 1]])
[[1, 0], [1, 0], [0, 1], [2, 0], [1, 1], [0, 2]]
```

nuc_to_aa(seq)

Returns a string of amino acids coded by the sequence.

Example 1.

```
>>> nuc_to_aa('ATGGCCATG')
'MAM'
```

get_aa(codon)

Returns the aminoacid coded by the codon.

Example 1.

```
>>> get_aa('CAA')
'Q'
```

pathways_a(codon_1, codon_2)

Returns all possible direct mutation pathways between two codons in aminoacid form.

Example 1.

```
>>> pathways_a('AAA', 'AGA')
K -> R
```

Example 2.

```
>>> pathways_a('ATG', 'CTA')
M -> L -> L
M -> I -> L
```

pathways_n(codon_1, codon_2)

Returns all possible direct mutation pathways between two codons in nucleotide form.

Example 1.

```
>>> pathways_n('AAA', 'AGA')
AAA -> AGA
```

Example 2.

```
>>> pathways_n('ATG', 'CTA')
ATG -> CTG -> CTA
ATG -> ATA -> CTA
```

Kruskal's algorithm

Adopted from 'https://github.com/israelst/Algorithms-Book--Python/blob/master/5-Greedy-algorithms/kruskal.py'

References

[1] Eyre-Walker A (2006), The genomic rate of adaptive evolution, Trends in Ecology and Evolution Vol. 21 No. 10: 569-575.

[2] Smith NGC & Eyre-Walker A (2002), Adaptive protein evolution in *Drosophila*, Nature Vol. 415: 1022-1024.

[3] McDonald JH & Kreitman M (1991), Adaptive protein evolution at the *Adh* locus in *Drosophila*, Nature Vol. 351: 652-654.