BBTM Matrix generator specification

parse

This function takes as input a matrix as given in the supplementary info to the PLoS paper.

Comment lines beginning with # are ignored.

The first non-comment line is assumed to be the column titles. They are placed in a list.

Exception thrown if not all the one-letter resns are there.

A dictionary is created, mapping the resns to dictionaries that map the resns to None. The first coordinate given to this dictionary represents a row, the second represents a column.

Loop over the rest of the lines. First element is row coordinate. Loop over zip(rest of the line, list of column titles). Column title is column coordinate. Set double-dictionary[row-coord, col-coord] to int(element of 'rest of the line').

Return double-dictionary.

get\_params(p, b):

Takes a transition probability matrix and the log-odds matrix made from it. Returns λ and each such that each element is .

The first two elements of a single column determine λ.

Subtracting the second equation from the first,

The first step is to calculate λ, using this formula.  
This method of calculation gets very inaccurate, as p1,1 and p2,1 approach each other for high powers, as do b1,1 and b2,1. And, in the Jimenez-Morales and Liang paper, the **B** matrices often have differences between these elements on the same order of magnitude as the unit to which the values were rounded.

The first rows of p and b are enough to determine π.

Make an empty dictionary.

For p\_elem and b\_elem in zip(p, b):

update the dictionary, key is column title, value is π, calculated as so:

Return the dictionary.

p(q,t)

Given a rate matrix q, return the corresponding transition probability matrix for time t.

The problem is that the input and output are dictionaries, but I need a scipy matrix for the intermediate step.

Retrieve the keys of the matrix. This arbitrary ordering of them will be used to generate the scipy matrix.

Make an empty list mat\_as\_list.

For row\_name in keylist, append to mat\_as\_list a row: [q[row\_name][col\_name] for col\_name in keylist]

Make it a matrix, p\_as\_mat = scipy.linalg.expm(t\*matrix)

So the ith row is the keylist[i] row, and the jth column is the keylist[j] row

So make p a dictionary mapping the keys in keylist to dictionaries that map the keys in keylist to None.

Then, for coordinates i, j in matrix, p[keylist[i]][keylist[j]] to p\_as\_mat[i][j]