I am so confused about so many aspects of these bbTM matrices.

First of all:  
They are assymetric! What the fuck?

What it means that it's asymmetric is that

Which implies that

the transition probability is defined in terms of the probability of having *i* in the beginning of a transition and *j* at the end, :

I used to think that , that is, that . Definitely not necessarily true. I thought, " is a conjunction, and conjunctions are symmetric." But it is a conjunction of statements that refer to different times, with the one referring to the older time on the left. Flipping around the conjunction does not give you , since you now have the *newer* term on the left:

This can be taken as an archetype for the category of mistake of overgeneralizing a good rule. Usually when a function of two arguments is equivalent to a probability of a conjunction where each conjunct depends upon one of those arguments, I can flip the arguments without changing the function. But, if there's some other asymmetry—that is, if each conjunct is a *different* function of its corresponding argument—then I can't do that. I think a lot of the probabilistic nature of doing mathematics comes from things like this - general rules where you don't explicitly know the conditions under which you can use them, and don't check before using them.

Okay. So, now I don't think that. However, due to the symmetry of the Q matrix, the P matrix is also symmetrical.

And, assuming that for all *n*,

check it:

So by induction, for all n,

So each is symmetric. Also:sums of symmetric matricesd and scalar multiples of them are also symmetric, so,

is also symmetric. So each transition probability matrix is symmetric.

This means that

However, we know that

So this must mean that... the amino acid frequencies are different!

Well, of course they are.

I think the reason I was confused is because I did this derivation:

Then, the step based upon my misconception:

BUT... nope.

Something does still confuse me, though. Why is 5% of each amino acid a steady state of the markov chain? Doesn't this mean that the chain evolves to a state where you don't have the original amino acid frequencies?

>>> q = to\_mat(qout, published\_bbtm\_ordering)

>>> pi = np.array([pi\_out[i] for i in published\_bbtm\_ordering])

>>> np.dot(q, pi)

matrix([[ -2.07946335e-04, 2.98190098e-05, 3.10953719e-05,

2.33197969e-05, 1.26684385e-05, 3.05525760e-05,

2.50681164e-05, 1.40598134e-04, 5.11704450e-05,

2.28091403e-03, -2.81836565e-03, 1.62677536e-05,

3.64861281e-04, 5.94308423e-04, 2.51986979e-05,

9.48637248e-05, 1.76412148e-04, 2.65772614e-04,

-3.70339735e-04, -7.66258429e-04]])

But that's wrong. Multiplying Q by a steady state should give the zero vector.

Maybe... I mean, it's close to the zero vector, and Q is rounded. Hmm.

>>> p = to\_mat(transition\_probability\_matrix(qout, 40), published\_bbtm\_ordering)

>>> pi - np.dot(p, pi)

matrix([[ 0.00793147, -0.00112644, -0.00122252, -0.00086069, -0.00048807,

-0.00117655, -0.00094563, -0.00408665, -0.00189842, -0.03497234,

0.05637428, -0.00062124, -0.01180398, -0.01057608, -0.00094772,

-0.00359124, -0.00552641, -0.00897365, 0.01292379, 0.0115891 ]])

Are those close? Not very close... right?

>>> p = to\_mat(transition\_probability\_matrix(qout, 120), published\_bbtm\_ordering)

>>> pi - np.dot(p, pi)

matrix([[ 0.01664735, -0.00314795, -0.00363208, -0.00242257, -0.00139258,

-0.00335373, -0.00268671, -0.00693655, -0.00510678, -0.03167912,

0.07472359, -0.00178881, -0.02739489, -0.0082226 , -0.00267014,

-0.00979478, -0.01159798, -0.02055225, 0.03063228, 0.0203796 ]])

And that, even less so. Still though...

>>> r = [random.random() for i in range(20)]

>>> r = [i / sum(r) for i in r]

>>> r = np.array(r)

>>> r - np.dot(p, r)

matrix([[ 2.78184012e-02, 2.44876445e-03, -1.75925047e-03,

-5.71322956e-04, -1.09138137e-04, -5.59717941e-03,

2.19048927e-03, 7.47070088e-03, 3.28795177e-05,

1.80251525e-02, -4.16840794e-02, 3.87954453e-04,

-9.27486071e-03, -1.92658217e-03, -2.75354423e-04,

2.84459516e-03, -2.70285368e-02, 1.85927625e-02,

-1.99240041e-02, 2.83437972e-02]])

So, it evolves about as much as any random distribution... it doesn't evolve unusually little. And, I mean, these are pretty big, they're on the same scale as the numbers themselves:

>>> pi

array([ 0.103414, 0.012083, 0.016985, 0.010899, 0.000253, 0.023042,

0.003965, 0.071558, 0.016882, 0.064497, 0.168981, 0.009315,

0.018249, 0.088656, 0.018898, 0.025996, 0.050352, 0.045422,

0.135606, 0.115135])

So, yeah, I don't think pi is a steady state. Not like,

>>> u = np.array([.05 for i in range(20)])

>>> u - np.dot(p, u)

matrix([[ 7.75670527e-08, 5.85964748e-07, 5.69742311e-07,

1.17205282e-06, -1.78361410e-06, 1.10736321e-06,

5.93607391e-07, 4.28322730e-07, -5.49848414e-07,

4.19250249e-07, 1.56521479e-07, 6.57420064e-09,

5.10643604e-07, -1.97371805e-07, 1.17339575e-06,

-1.55074903e-08, -8.09024804e-07, 4.96440348e-07,

2.74457017e-08, 2.30358209e-07]])

There's something more like a steady state.

What about David's matrices? Are they better, in that sense?

>>> d = parse\_david('MTMout120.p')

>>> d = to\_mat(d, published\_bbtm\_ordering)

>>> pi - np.dot(d, pi)

matrix([[ 0.00118237, -0.00612475, -0.0067053 , -0.00471671, -0.00304407,

-0.00678844, -0.0055772 , -0.0156902 , -0.00965823, -0.05024322,

0.05385243, -0.00353401, -0.04883741, -0.02490173, -0.00502435,

-0.01668724, -0.02347631, -0.03325851, 0.01415969, 0.00135788]])

>>> u - np.dot(d, u)

matrix([[ 9.00000000e-08, -5.70000000e-07, 5.50000000e-07,

-5.55000000e-07, -5.85000000e-07, -5.45000000e-07,

-6.93889390e-18, -3.65000000e-07, -5.70000000e-07,

-3.00000000e-08, 6.65000000e-07, -5.75000000e-07,

-3.00000000e-07, 2.55000000e-07, -5.85000000e-07,

-4.55000000e-07, 8.55000000e-07, 4.00000000e-08,

4.00000000e-08, 2.35000000e-07]])

Nope.

So, its weird to use a model in which a distant ancestor sequence is a better match to a uniform modern sequence than a random modern sequence.