Lab 05 Markov models

For Lab 05 | Modeling DNA Evolution you will generate and compare substitution rates under different nucleotide substitution models using two aligned sequences that are provided. You will use both analytic solutions as well as Markov Models to calculate your answers. The models are JC69 (Jukes and Canter 1969), K80 (Kimura 1980), HKY85 (Hasegawa, Kishino and Yano 1985).

The lab is split into the following sections:

- Introduction
- Probability theory
- Markov models
- Python modules, packages: NumPy and SciPy
- Nucleotide substitution models

Assignment

Follow the instructions in this document and answer the questions in the cell below each question. Submit your answers by uploading a PDF file to gradescope. To generate the pdf, first export the notebook as HTML: >File, >Export to ..., >HTML. Then, open the HTML in a browser and use your browser to print to PDF.

Check to make sure all your cells have been run and the **results** displayed in the PDF file.

Reminder, provide comments for any code you write to ensure partial credit.

Introduction

Many biological processes are stochastic and can be modeled using Markov chains. For example, changes in DNA and proteins sequences over time can be modeled using a Markov chain and such models form the basis for many of the inferrences we derive from comparing DNA or protein sequences to one another. Many different forms of a Markov model have been used for DNA sequence evolution, i.e. nucleotide substitution over time. These substitution models differ in terms of the parameters used to describe the rates at which one nucleotide replaces another during evolution. These models are frequently used in molecular phylogenetic analyses, such as during the calculation of the likelihood of a particular tree, and they are used to estimate the substitution rate and the amount of time that separates two sequences.

Before going into the specifics of the models, we will review a few concepts in probability theory.

Probability theory

The probability of something, an event happening, is the number of possible ways the event can occur divided by the number of all possible outcomes if all outcomes have an equal chance of occurring (fair coin or dice).

The probability of some event A, can be written as $P(A)$. If A indicates heads when flipping a fair coin and B indicates tails, the total number of possible outcomes is two. For a fair coin:

$$P(A) = \frac{1\ (way\ to\ get\ a\ head)}{2\ (possible\ outcomes)} = 0.5$$

If different outcomes have different chances of occurring then the probability of an event (A) is defined by its frequency as follows:

$$P(A) = \lim_{n_t \to \infty} \frac{n_A}{n_t}$$

where n_A is the number of events where A occurs and n_t is the total number of events.

A random variable or stochastic variable is a variable whose possible values are outcomes of a random phenomenon. For example a random variable can be heads or tails for a coin toss.

Rules

When there are two events, their combined probability depends on whether they are independent of one another. If events A and B are *independent* then

$P(A\ and\ B) = P(A) \times P(B)$ The probability of A and B both occurring is the probability of A times probability of B.

$P(A|B) = P(A)$ The probability of A given that B occurs is the same as the probability of A. Thus, when independent $P(A)$ does not depend on B.

If A and B are *dependent* then

$$P(A\ and\ B) = P(A|B) \times P(B) = P(B|A) \times P(A)$$

Addition rule:

$$P(A\ or\ B) = P(A)\ +\ P(B) - P(A\ and\ B)$$

If A and B are mutually exclusive, e.g. heads and tails, then $P(A\ and\ B)$ = 0 and

$$P(A\ or\ B) = P(A)\ +\ P(B)$$

If A and B are the only possible outcomes and are mutually exclusive events, then
$P(A) + P(B) = 1$

Markov models

A **stochastic process** is a sequence of random variables based on the same sample space. The possible outcomes of the random variables are called the set of possible states of the process. For example, the sequence of outcomes when rolling a fair, six-sided die is a stochastic process with discrete random variables.

A **Markov process** is a stochastic process that is *memoryless*. Memoryless is an attribute a process may possess wherein the probability of the next state is in NO way affected by an outcome of a previous state/outcome. It is however dependent on the current state.

A **Markov chain** is a discrete-valued Markov process. Discrete-valued means that the state space of possible values of the Markov chain is finite or countable. Thus, a *Markov chain* is a special type of stochastic process where the probability of the next state conditional on the entire sequence of previous states up to the current state is only dependent on the current state. Dice-based games, e.g. Chutes and Ladders, form a discrete time Markov chain: the probability of a move only depends on the current position on the board and the role of the dice.

A continuous-time Markov chain is one in which changes to another state can happen at any time along a continuous time interval. The time that a continuous-time Markov chain spends in a state has an exponential distribution. This is in part due to the exponential distribution being the only continuous distribution with the unique property of memorylessness: the waiting time to the next event is independent of when the last event occurred.

Stochastic process

Markov chains are just one type of stochastic process. Other types include Bernoulli process, Poisson process and Gaussian process, specified by the types of random variables that describe them. On the real number line, e.g. time, the Poisson process is a type of continuous-time Markov process known as a birth-death process. It is defined by the number of points in each finite interval as having a Poisson distribution. A special characteristic of the Poisson process is that the distance between two consecutive points on the real line will be an exponential random variable with parameter λ.

Properties of discrete time Markov Chains

A discrete time Markov chain is defined by a transition matrix P_{ij} of transitioning from state i to state j:

$$P_{ij} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \\ p_{41} & p_{42} & p_{43} & p_{44} \end{bmatrix}$$

where p_{ij} is the probability of going from state i to j: $P(X_{t+1} = j | X_t = i)$

In two steps we have:

$$P(2) = P \cdot P = P^2$$

In n steps we have:

$$P(n) = P^n$$

In the limit as n goes to infinity, we get the equilibrium values of being in each state:

$$\lim_{n \to \text{inf}} P^n = \pi \tag{1}$$

such that

$$\pi = P\pi$$

Python packages and modules

A package is a collection of Python modules: while a module is a single Python file containing code with specific functionality. A Python module can have a set of functions, classes or variables defined and implemented.

A module allows you to logically organize your Python code. Grouping related code into a module makes the code easier to understand and use. More specifically, a module is a Python object with arbitrarily named attributes that you can bind and reference.

Python modules or packages can be installed and used, but many are built in to Python. Here is a list of built in Python modules:
https://docs.python.org/3/py-modindex.html

Importing python modules

To use a module it must be imported. This can be done a couple of different ways. We will use the built in *math* module as an example. The math module contains a number of constants and functions, e.g. log(), log10(), exp(), etc.
https://docs.python.org/3.7/library/math.html

To import math:

In [1]:
```
# Import built-in module math
import math

# To see what has been imported
content = dir(math)
print(content)
```

```
['__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__', 'ac
os', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'comb', 'copysi
gn', 'cos', 'cosh', 'degrees', 'dist', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fab
s', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'in
f', 'isclose', 'isfinite', 'isinf', 'isnan', 'isqrt', 'lcm', 'ldexp', 'lgamma',
```

```
'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'nextafter', 'perm', 'pi', 'po
w', 'prod', 'radians', 'remainder', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau',
'trunc', 'ulp']
```

Here, the special string variable __name__ is the module's name, and __file__ is the filename from which the module was loaded. Python comes with a number of special variables and methods whose name is preceeded and followed by __, __name__ is just one.

__name__ defines the namespace that a Python module is running in. When Python is run, it will replace __name__ with it's namespace. When a packages is imported, its __name__ is replaced by the name of the module, but the code in that module will not be executed automatically.

When we import math, we can access its function for the exponential e^x using `math.exp(x)` .

In [2]:
```
math.exp(-1)
```

Out[2]:
```
0.36787944117144233
```

There are a few different ways to import a module.

To import just `exp()` :

In [3]:
```
from math import exp
exp(-1)
```

Out[3]:
```
0.36787944117144233
```

In this context we've imported just the function, and not the name space so it can be called by `exp()` not `math.exp()` .

We can also rename the name space when importing:

In [4]:
```
import math as m
m.exp(-1)
```

Out[4]:
```
0.36787944117144233
```

To import all of the math functions you can use * to indicate a match to any named function, variable, etc.

In [5]:
```
from math import *
log2(2)
```

Out[5]:
```
1.0
```

NumPy and SciPy

The NumPy Package is a Python package that contains modules with various functions that facilitate representation and calculation of mathematics, in particular linear algebra, Fourier transform, and random number capabilities.

SciPy (pronounced “Sigh Pie”) is a Python-based ecosystem of open-source software for mathematics, science, and engineering. In particular, these are some of the core packages: NumPy (Base N-dimensional array package), Pandas (data structures and analysis), Matplotlib (Comprehensive 2D Plotting), and more.

These packages are great for example:

- NumPy’s array type augments the Python language with an efficient data structure useful for numerical work, e.g., manipulating matrices. NumPy also provides basic numerical routines, such as tools for finding eigenvectors.
- SciPy contains additional routines needed in scientific work: for example, routines for computing integrals numerically, solving differential equations, optimization, and sparse matrices.
- The matplotlib module produces high quality plots. With it you can turn your data or your models into figures for presentations or articles. No need to do the numerical work in one program, save the data, and plot it with another program.

Matrix algebra

Lets define a transition matrix P using NumPy and calculate some transitions.

In [6]:
```
# Importing NumPy package
import numpy as np

# Define NumPy array
P = np.array( [[1/4, 1/2, 1/4], [1/3, 0, 2/3], [1/2, 0, 1/2]] )
print(P)
```

```
[[0.25       0.5        0.25      ]
 [0.33333333 0.         0.66666667]
 [0.5        0.         0.5       ]]
```

Lets define a vector X_0, as a vector of probabilities of being in state i that correspond to the 1st, 2nd, and 3rd rows of the transition matrix.

We can then take the dot product of this vector and the transition matrix to get the probability of being in each state at the next time-point: X_1

In [7]:
```
# Define vector
X0 = np.array([1,0,0])

# Take the dot product
X1 = X0.dot(P)
print(X1)

X2 = X1.dot(P)
print(X2)

X3 = X2.dot(P)
print(X3)
```

```
[0.25 0.5  0.25]
[0.35416667 0.125      0.52083333]
[0.390625   0.17708333 0.43229167]
```

Thus, X_1 tells us the probability of P_{12} in one step is 0.5, and P_{13} in one step is 0.25.

X_2 tells us the probability of transition from state 1 to state 2 in two steps is 0.125.

Similarly, we can use the dot product of the transition matrix to get the matrix of all transition probabilities over multiple steps:

In [8]:
```
# Define vector
P2 = P.dot(P)
print(P2)

P3 = P2.dot(P)
print(P3)
```

```
[[0.35416667 0.125      0.52083333]
 [0.41666667 0.16666667 0.41666667]
 [0.375      0.25       0.375     ]]
[[0.390625   0.17708333 0.43229167]
 [0.36805556 0.20833333 0.42361111]
 [0.36458333 0.1875     0.44791667]]
```

P3 tell us that probability of transitioning from state i to state j in three steps.

Question 1

Given a transition matrix (P) with four state {A, G, C, T}, calculate the probability of being in state A, G, C and T after 20 steps, given a current state of G.

What is the probability of being in state G at all 20 steps, given a current state of G.

What is the probability of being in state G after 20 steps given a current state of G *and* having transitioned through at least one A, C or T state in intermediate steps.

(4 points)

In [9]:
```
pij = 0.01
pii = 0.97
P = np.array( [[pii, pij, pij, pij], [pij, pii, pij, pij], [pij, pij, pii, pij],
# The states in the first through forth elements correspond to A, G, C, T
print(P)
```

```
[[0.97 0.01 0.01 0.01]
 [0.01 0.97 0.01 0.01]
 [0.01 0.01 0.97 0.01]
 [0.01 0.01 0.01 0.97]]
```

In [10]:
```
# Answer
```

Properties of continuous time Markov Chains

Given transition probabilities of a Markov chain, $P(t)$ for a matrix with entries:

$p_{ij} = P(X_t = j | X_0 = i)$

then the matrix $P(t)$ satisfies the Kolmogorov forward equation:

$P'(t) = P(t)Q$

Where P' indicates differentiation with respect to time t, and p_{ij} is the probability of being in state j at time t given a current state of i at time $t = 0$. The solution to this equation is given by the matrix exponential:

$P(t) = e^{Qt}$

Where Q is the instantaneous rate matrix, which describes the amount of change or flow from one state to another per unit time, and the sum of each row is 0:

$$\sum_{i=0}^{n} Q_{ij} = 0 \tag{2}$$

Which can be achieved by defining:

$$Q_{ii} = -\sum_{i \neq j}^{n} Q_{ij} \tag{3}$$

The solution to matrix exponentiation is given by:

$$P(t) = e^{Qt} = \sum_{n=0}^{\infty} Q^n \frac{t^n}{n!} \tag{4}$$

To exponentiate a matrix, we will need to import some scientific computing modules: NumPy and SciPy.

Nucleotide substitution models

Nucleotide substitution models are modeled as time-homogeneous, continuous time Markov chains. They have the following set of underlying assumptions:

- At any given site in a sequence, the rate of change from base i to base j is *independent* from the base that occupied the site prior to i (**Markov property**).
- Substitution rates do not change over time (**homogeneity**).
- The relative frequencies of A, G, C and T are at equilibrium (**stationarity**).

For DNA substitution models, the transition matrix is a 4x4 matrix $P(t)$. Each element gives the probability that state i will change to state j in time t, with state space $[A, G, C, T]$.

$$P_{ij}(t) = \begin{bmatrix} p_{AA}(t) & p_{AG}(t) & p_{AC}(t) & p_{AT}(t) \\ p_{GA}(t) & p_{GG}(t) & p_{GC}(t) & p_{GT}(t) \\ p_{CA}(t) & p_{CG}(t) & p_{CC}(t) & p_{CT}(t) \\ p_{TA}(t) & p_{TG}(t) & p_{TC}(t) & p_{TT}(t) \end{bmatrix}$$

Thus, $p_{AG}(t)$ is the probability of a transition from A to G in time t. In a fully parameterized model the instantaneous rate matrix is given by:

$$Q = \begin{bmatrix} -(\eta\pi_G + \delta\pi_C + \beta\pi_T)\mu & \eta\pi_G\mu & \delta\pi_C\mu & \\ \eta\pi_A\mu & -(\eta\pi_A + \epsilon\pi_C + \gamma\pi_T)\mu & \epsilon\pi_C\mu & \\ \delta\pi_A\mu & \epsilon\pi_G\mu & -(\delta\pi_A + \epsilon\pi_G + \alpha\pi_T)\mu & \\ \beta\pi_A\mu & \gamma\pi_G\mu & \alpha\pi_C\mu & - \end{bmatrix}$$

where $\eta,\ \delta,\ \beta,\ \epsilon,\ \gamma,\ \alpha$ are the transition rates parameters, π_N is the equilibrium nucleotide frequency of base N $[A, G, C, T]$, and μ is the mutation rate. The rate of a nucleotide either not changing or re-arriving at its original base is the negative sum of the row rates. For example, given the current state is an A (first row), the rate of change to C is the product of the mutation rate (μ), the relative rate for this type of change (δ), and the equilibrium frequency of C (π_C).

The Q matrix, as shown above, represents the most generalized model possible and also one that is time-reversible (GTR: Generalized time-reversible). This model has 4 equilibrium frequencies of each nucleotide as well as 6 substitution rates governing different types of changes.

What is an instantaneous rate matrix and why do we use it? Recall that $P'(t) = P(t)Q$ and so $P'(0) = Q$ or the rate matrix is equal to the derivative of the transition probability matrix as time goes to zero.

If we assume that nucleotides are expected to occur at all the same frequencies, and that mutation rates are all equally likely as well, we get the simplest substitution model, created by Thomas Jukes and Charles Cantor in 1969.

Jukes-Cantor Model (JC69)

The Jukes-Cantor model is a simplified version of the GTR model. It has an instantaneous rate matrix:

$$Q = \begin{bmatrix} -3\mu & \mu & \mu & \mu \\ \mu & -3\mu & \mu & \mu \\ \mu & \mu & -3\mu & \mu \\ \mu & \mu & \mu & -3\mu \end{bmatrix}$$

Thus, the JC69 model has only one parameter, the mutation rate μ. In class we used α for the mutation rate to each base. It would make sense to use $\frac{\mu}{3}$, since mutation rate is better thought of

as the rate to any base and so the rate to a particular base is $\frac{\mu}{3}$. As you will see, this is the correct scaling.

However, the instantaneous rate matrix determines the continuous time Markov model and when using the model it can easily be converted to a different time scale as a matter of convenience. As a consequence, the JC69 model is also equivalent to the following model, with * representing negative row sums:

$ \begin{equation} Q = \begin{bmatrix}

- & 1 & 1 & 1 \ 1 & *& 1 & 1 \ 1 & 1 & *& 1 \ 1 & 1 & 1 & * \end{bmatrix} \end{equation} $

Scaling

Because μ is the mutation rate per generation and t is time in generations, we would like the substitution rate, μt, to be measured in units of substitutions per site. This can be accomplished by scaling the rate matrix such that when the substitution rate μt = 1, there is one substitution per site. A simple way to achieve this scaling is to scale μ, the total number of substitutions per unit time to 1, then t is a measure of substitutions per site. For example if we scale μ to 1 and t = 0.5 then μt = 0.5 substitutions per site.

To accomplish this scaling, we devide each cell in the matrix by a constant scaling factor. The total number of substitutions per unit time is the negative sum of the rate of no change times the equilibrium base frequencies:

$$\mu = -\sum_{i=1}^{n} \pi_i Q_{ii} \tag{5}$$

Thus, the scaling is achieved by dividing each element in Q by μ so that the total flux through the matrix is 1. In this way, $P(t) = e^{Qt}$ will give the transition probabilities such that $t = 1$ is the equivalent of 1 substitutions per site. The alternative would be to use a known value of μ, for example: 10^{-8}, and then t would be measured in generations and substitution rate would be calculated by μt. Therefore, the scaling is a matter of convenience.

First, lets see the equivalence of the two Q matrices above. * is the negative row sum = -3 and has the form $P(t) = e^{Q\mu t}$. The first Q matrix already has a row sum of zero because the diagonal is -3μ and so has the form $P(t) = e^{Qt}$ since μ is a constant inside each cell in the matrix. Thus, the two are equivalent.

Now, lets scale the Q matrix so that t = 1 is equivalent to one substitution per site.

Scaling the Jukes-Cantor model

To scale Q such that the total flux is 1, each element is divided by 3 = -(-3*0.25 + -3*0.25 + -3*0.25 + -3*0.25), giving:

$$Q = \begin{bmatrix} -1 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & -1 & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{3} & -1 & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & -1 \end{bmatrix}$$

The equivalent scaling is to divide by 3μ for the first Q matrix. The result is the same.

Calculating transition probabilities

Given a model (JC69), what is the transition matrix $P(t)$, given $t = 1$ substitution/site?

Lets first start by initializing a rate matrix Q and multiplying by the equilibrium base frequencies π. To do so we will import the NumPy package and use NumPy's two-dimensional array: https://docs.scipy.org/doc/numpy/reference/generated/numpy.array.html

In [11]:
```
import numpy as np
Q = np.array( [[0, 1, 1, 1], [1, 0, 1, 1],[1, 1, 0, 1],[1, 1, 1, 0]] )
pi=np.array([.25,.25,.25,.25])
Q = Q*pi
print(Q)
```

```
[[0.   0.25 0.25 0.25]
 [0.25 0.   0.25 0.25]
 [0.25 0.25 0.   0.25]
 [0.25 0.25 0.25 0.  ]]
```

Lets make the sum of each row zero by filling in the diagonal with the sum of each row. Documentation for sum and diagonal here:
https://docs.scipy.org/doc/numpy/reference/generated/numpy.sum.html#numpy.sum
https://docs.scipy.org/doc/numpy/reference/generated/numpy.fill_diagonal.html?highlight=fill_diagonal

In [12]:
```
zsum = Q.sum( axis=1 )
np.fill_diagonal( Q,-zsum )
print(Q)
```

```
[[-0.75  0.25  0.25  0.25]
 [ 0.25 -0.75  0.25  0.25]
 [ 0.25  0.25 -0.75  0.25]
 [ 0.25  0.25  0.25 -0.75]]
```

Scale the matrix so that μt = 1 is equivalent to 1 substitution per site.

$$\mu = -\sum_{i=1}^{n} \pi_i Q_{ii} \tag{6}$$

In [13]:
```
beta = 1/sum( zsum*pi )
Q = Q*beta
print(Q)
```

```
[[-1.          0.33333333  0.33333333  0.33333333]
 [ 0.33333333 -1.          0.33333333  0.33333333]
 [ 0.33333333  0.33333333 -1.          0.33333333]
 [ 0.33333333  0.33333333  0.33333333 -1.        ]]
```

We can then exponentiate the rate matrix using $t = 1$ to get the transition matrix $P(t)$, or $P(1)$.

To do so we will import a function from the linear algebra SciPy package:

https://docs.scipy.org/doc/scipy-0.15.1/reference/linalg.html

that exponentiates a rate matrix: `expm` .

https://docs.scipy.org/doc/scipy-0.15.1/reference/generated/scipy.linalg.expm.html

In [14]:

```
from scipy.linalg import expm

P = expm(Q*1)
print(P)
```

```
[[0.44769785 0.18410072 0.18410072 0.18410072]
 [0.18410072 0.44769785 0.18410072 0.18410072]
 [0.18410072 0.18410072 0.44769785 0.18410072]
 [0.18410072 0.18410072 0.18410072 0.44769785]]
```

We can check our results using the JC69 model equations.

Jukes and Cantor 1969 (JC69) model

Because the JC69 model is so simple, there is a closed form solution:

$$P_{ij}(t) = \frac{1}{4} - \frac{1}{4}e^{-4\alpha t}$$

$$P_{ii}(t) = \frac{1}{4} + \frac{3}{4}e^{-4\alpha t}$$

Using these two equations and with some math you can derive the substitution rate Kas a function of the frequency of differences between two sequences p:

$$K = -\frac{3}{4}\log(1 - p\frac{4}{3})$$

Because $\alpha = \frac{\mu}{3}$ we get the same transition probabilities as calculated by exponentiating the rate matrix.

In [15]:

```
Pii = 1/4 - 1*exp(-4/3)/4
print(Pii)
Pij = 1/4 + 3*exp(-4/3)/4
print(Pij)
```

```
0.18410071547106832
0.4476978535867951
```

Question 2

Write a function that takes two ungapped sequences as input and calculates the nucleotide substitution rate (per bp) using the Jukes-Cantor model.

The JC69 equation is: $K = -\frac{3}{4}\log(1 - p\frac{4}{3})$

(4 points)

In [16]:
```
# Import the log function from the math package
# This step will be described below, for now you can use log(x) to get the log c
from math import log

# Two aligned sequences are provided
A = 'GACCCATCTCTCACTAAATCAGTACTAAATGCACCCACATCATTATGCACGGCACTTGCC'
B = 'GATCCACCTCTCACTTAATGAGTAAAAGGTGCACCCCCATCATTATGCAGGGCACATGAC'

# Function that calculates JC69 substitution rate
def JC69(A,B):
    subs = 0
    return(subs)

# Answer
```

Question 3

Write a function that takes time t as input and calculates the 4x4 transition matrix $P(t)$ under the JC69 model. Matrix rows and columns are A, G, C, T and rows indicate the t = 0 state and columns indicate the state transition at time t. Use this function to calculate the transition matrix for t = 0.8.

Use this function to calculate the log10 probability of the two aligned sequences in SSA3.fasta (Seub_SSA3 and Spar_SSA3) under the JC69 model with t = 0.8. Because we assume sites to change independently of one another the probability of the two sequences is the product of the probabilities at each site. Gaps should not be included. The reason you need to use log10 probabilities is to avoid errors associated with multiplying probabilities. In other words add log10 probabilities rather than multiplying them.

(4 points)

In [17]:
```
# Answer
```

Maximizing the likelihood

The JC69 model has a solved equation that lets us estimate the substitution rate based on the number of differences. But what if we want to use the HKY85 model? There is no equation, but we can calculate the probability of the data given time t, equilibrium base frequencies π and the ratio of transitions to transversions, κ. (Note that the substitution rate is μt which we scale to 1 so that t actually represents the substitution rate.)

Under the HKY85 model:

$ \begin{equation} Q = \begin{bmatrix}

- & {\kappa \pi_G} & \pi_C & \pi_T \ {\kappa \pi_A} & & \pi_C & \pi_T \ \pi_A & \pi_G & & {\kappa \pi_T} \ \pi_A & \pi_G & {\kappa \pi_C} & * \end{bmatrix} \end{equation} $

One solution is to find the maximum likelihood of t by evaluating the probability of the data over a range of possible times (t values). There are various algorithms that can be used to do this, but if the search space is small we can get a pretty good estimate by evaluating a large number of t values to find the one with the highest probability.

Question 4

Write a function that returns a transition probability matrix $P(t)$ given time t, the ratio of transitions/transversions κ, and a vector of equilibrium nucleotide frequencies $[A, G, C, T]$, i.e. under the HKY85 model.

Use this function to evaluate the log10 probability of the SSA3 sequences with t taking values that range from 0.01 to 1 with a step size of 0.001. Print the largest log10 probability and the value of t, i.e. the value of t that is most likely given the data. Use κ = 4 and nucleotides frequencies set to the observed frequencies of the SSA3 sequences.

np.arange is useful for iterating over range of non-integers.

(4 points)

In [18]:
```
# Answer
```

Rate heterogeneity

Substitution rates are often heterogeneious. In coding sequences the first two positions in a coding predominantly cause nonsynonymous (amino acid) substitutions whereas changes in the third position predominantly cause synonymous substitutions (no amino acid change). It is thus often desirable to estimate substitution rates for different classes of sites.

Question 5

Use the JC69 model to calculate the substitution rate at all first, second and third codon positions in the two aligned sequences in SSA3. Assume the first position is 1 so the codon positions will be: 123123123123 etc in the sequences.

(4 points)

In [19]:
```

```

```
# Answer
```

Bonus Question

Use the HKY85 model to estimate the substitution rate at first, second and third codon positions of the SSA3 sequences. For each of the three classes of sites use a ratio of transitions/transversions κ, and a vector of equilibrium nucleotide frequencies $[A, G, C, T]$, estimated from the data. For example, use the nucleotide frequencies and transition/transversion ratios at first codon positions to estimate the substitution rate at first codon positions. To find the optimum substitution rate evaluate values of t that range from 0.01 to 1 with a step size of 0.001.

Your answer should print the values of π, κ, log(p) and t for first, second and third positions.

(2 points)

In [20]:
```
# Answer
```