

## Basic Models of Nucleotide Evolution

Over time, nucleotides within a sequence can 'evolve' through substitution. This process can cause a nucleotide (T, C, A or G) to change into another nucleotide and is the main driving force behind evolution. For example, the nucleotide A in a sequence of DNA can change over time into the nucleotide C. This change may result in this sequence of DNA becoming inactive if the sequence was previously involved in protein synthesis as an exon, or may change the protein that the sequence codes. As proteins are the building blocks of organic life, this may cause large changes in an organism's features. Alternatively, this change may have no effect at all. On average, this form of mutation only occurs once or twice every million years. However, in assessing the evolution of species over hundreds of millions of years, models are useful in evaluating how one sequence of nucleotides may have evolved from another.

Models of nucleotide evolution can be used when examining two sequences of DNA of the same length that may be related. This type of model would be used to compare the two sequences by either assuming that one sequence evolved into the other or vice-versa, or assuming that they had evolved from a common 'ancestral' sequence of DNA. Applying the model would give the estimated number of nucleotide substitutions per site, called the distance, which would then be used to estimate a time. This time could then relate to when one sequence evolved from the other or would relate to how long ago that an 'ancestral' sequence of DNA would have diverged into each sequence.

In this paper, I will outline the principles and theory behind the main (most commonly used) models of nucleotide substitution, addressing each model chronologically and in some senses with increasing complexity. The models are as follows:

- Jukes and Cantor 1969 (JC69)
- Kimura 1980 (K80)
- Felsenstein 1981 (F81)
- Hasegawa, Kishino and Yano (HKY85)
- Tamura and Nei 1993 (TN93)

I will demonstrate how programming software may be used to process data using the formulae proposed within each model. From this I will explain how, continuing to use programming software, each model is capable of simulating the evolution of a nucleotide sequence over a given time.

### ***JC69 Model***

In terms of creating models that assess nucleotide substitution, the rate of substitution from one nucleotide to another and the time over which substitution has been allowed to act are key variables. Different models organise their use of rates in different ways but time is always used in the same way. The simplest model of nucleotide substitution is the Jukes and Cantor 1969 (JC69) model. This model assumes that the rate of substitution is the same between all nucleotides. Therefore, this model only requires a single parameter-denoting rate, along with a value for time. A 4x4 matrix can be created showing the rates of nucleotide substitution between the 4 nucleotides. This is known as matrix **Q**:

**Q =**

	A	T	C	G
A	-	$\alpha$	$\alpha$	$\alpha$
T	$\alpha$	-	$\alpha$	$\alpha$
C	$\alpha$	$\alpha$	-	$\alpha$
G	$\alpha$	$\alpha$	$\alpha$	-

Along the diagonal of this matrix, you can see that the rates of nucleotides changing into themselves are not displayed, as they are not regarded as substitutions. Also, the rows sum to 0. Using the rates in matrix **Q**, we can work out the probability of each nucleotide substitution occurring when  $t > 0$ , creating another matrix. This matrix is known as the transition probability matrix (**P(t)**) and is also a 4x4 matrix:

$$P(t) = \begin{pmatrix} \frac{1}{4} + \frac{3}{4}e^{-t\mu} & \frac{1}{4} - \frac{1}{4}e^{-t\mu} & \frac{1}{4} - \frac{1}{4}e^{-t\mu} & \frac{1}{4} - \frac{1}{4}e^{-t\mu} \\ \frac{1}{4} - \frac{1}{4}e^{-t\mu} & \frac{1}{4} + \frac{3}{4}e^{-t\mu} & \frac{1}{4} - \frac{1}{4}e^{-t\mu} & \frac{1}{4} - \frac{1}{4}e^{-t\mu} \\ \frac{1}{4} - \frac{1}{4}e^{-t\mu} & \frac{1}{4} - \frac{1}{4}e^{-t\mu} & \frac{1}{4} + \frac{3}{4}e^{-t\mu} & \frac{1}{4} - \frac{1}{4}e^{-t\mu} \\ \frac{1}{4} - \frac{1}{4}e^{-t\mu} & \frac{1}{4} - \frac{1}{4}e^{-t\mu} & \frac{1}{4} - \frac{1}{4}e^{-t\mu} & \frac{1}{4} + \frac{3}{4}e^{-t\mu} \end{pmatrix}$$

These formulae calculate the probability of one nucleotide evolving into another. They are achieved through the exponentiation of the Matrix **Q** using the Matrix Taylor series.

In terms of using the matrix **P(t)** with real-world or experimental data, a program can be written which will calculate the transition probabilities of each nucleotide substitution using the formulae in **P(t)**. Python is programming software that provides a basic but effective programming language, which can be used in these circumstances. We must first define a function that will implement the formulae of the matrix **P(t)** when given certain values to work from. These values are called parameters and in the case of working out the transition probabilities, we must input a value for the rate at which nucleotide substitutions will occur as well as a value for the time over which substitutions will occur.

The following code, written in Python, emulates the matrix **P(t)**:

```
IDLE 1.2.2
>>> from math import *
>>> def JC69(m,t):
    a = 0
    b = 1
    matrixq = [[a,b,b,b],[b,a,b,b],[b,b,a,b],[b,b,b,a]]
    x = 1/4.0 + 3/4.0 * exp(-4*m*t)
    y = 1/4.0 - 1/4.0 * exp(-4*m*t)
    a = x
    b = y
    matrixq = [[a,b,b,b],[b,a,b,b],[b,b,a,b],[b,b,b,a]]
    print matrixq

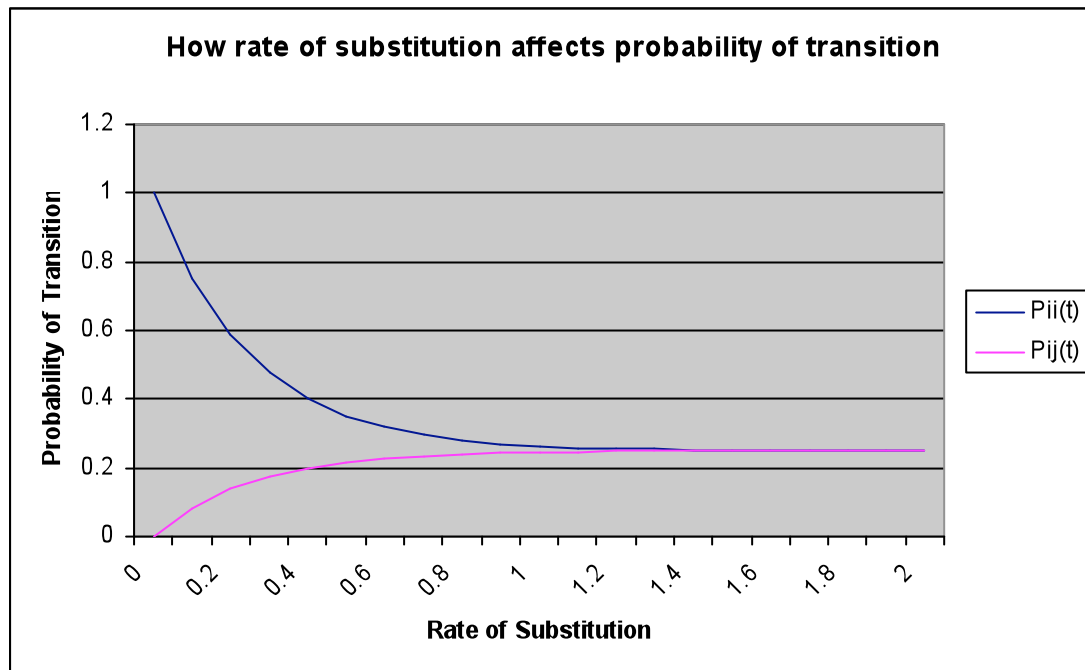
>>> # the value m denotes the rate
>>> # the value t denotes the time
>>> JC69(0.2,1)
[[0.58699672308791617, 0.13766775897069461, 0.13766775897069461, 0.13766775897069461],
 [0.13766775897069461, 0.58699672308791617, 0.13766775897069461, 0.13766775897069461],
 [0.13766775897069461, 0.13766775897069461, 0.58699672308791617, 0.13766775897069461],
 [0.13766775897069461, 0.13766775897069461, 0.13766775897069461, 0.58699672308791617]]
>>>
```

As shown at the bottom of the image, inputting an experimental rate (0.2) and time (1) tests the function 'JC69' used to calculate the transition probabilities. This is followed by a matrix displaying the probabilities row by row with nucleotide order T, C, A and G, in the same orientation as the matrix **Q**. In looking at the formulae used to calculate the transition probabilities, conclusions can be made to how the increasing rate or time will affect the resultant probabilities.

$$x = 1/4.0 + 3/4.0 * \exp(-4*m*t)$$
$$y = 1/4.0 - 1/4.0 * \exp(-4*m*t)$$

The exponential (exp) of a negative value gives a decimal number smaller than 1. If the negative value increases in size, the exponential of that value becomes smaller at an increasing rate. Therefore, as the negative value tends to infinity, the exponential of that value tends to 0. In looking at the above formulae X and Y, as the values of m (rate) and t (time) increase, the values being added to  $\frac{1}{4}$  in X and subtracted from  $\frac{1}{4}$  in Y become infinitely smaller. This results in the transition probabilities tending towards  $\frac{1}{4}$  for each nucleotide substitution. This supports the assumption that over an increased time or rate, so many nucleotide substitutions would have occurred that the target nucleotide is eventually random, with a probability of  $\frac{1}{4}$  for each nucleotide.

This is demonstrated in the following graph, taking increasing values for rate with a constant time of 1:



$P_{ii}(t)$  represents the probability that a nucleotide will not experience a substitution over a period of time ( $t$ ).  $P_{ij}(t)$  represents the probability that a nucleotide will experience a substitution and evolve into another nucleotide over a period of time. At the point when time = infinity, over which a nucleotide sequence had been allowed to evolve, the proportion of nucleotides of each type (T, C, A, G) will have reached  $\frac{1}{4}$  for each. This distribution of nucleotides is called the limiting distribution and as the rates of change are the same for all nucleotides in the JC69 model, this proportion will be maintained. This proportional equilibrium is called the stationary distribution.

### ***K80 Model***

Kimura and associates created a model proposing a more complex mix of rates between nucleotide substitutions in 1980. This model is commonly known as the K80 model and uses two rates as parameters along with time. Nucleotide substitutions can be classified as one of two types; transitions and transversions. Transitions are substitutions between nucleotides of the same or similar molecular structure; between purines or between pyrimidines, and are prone to occur more frequently to other substitutions. Nucleotides A and G are purine molecules and experience higher substitutions between each other, as well as nucleotides T and C which are pyrimidine molecules. All other substitutions are transversions and are known to occur less frequently than transitions. In 1980, the first mitochondrial sequences were published showing a definitive difference between the frequencies of transitions and transversions, transitions being noticeably higher. As a result, the K80 model was developed and implemented by Kimura and associates in response to these findings.

The rate matrix (**Q**) in the K80 model displays two rates; alpha (representing the substitution rates of the transitions) and beta (representing the substitution rates of the transversions). In the following representation of the matrix **Q**, alpha = K and beta = 1:

$$Q = \begin{pmatrix} * & \kappa & 1 & 1 \\ \kappa & * & 1 & 1 \\ 1 & 1 & * & \kappa \\ 1 & 1 & \kappa & * \end{pmatrix}$$

As with the rate matrix for the JC69 model, the diagonal elements of the matrix **Q** are not included, as these are not regarded as substitutions. The total substitution rate for any nucleotide would be **a + 2b** (K + 1+1). Deriving the transition probability matrix from the matrix **Q** is slightly more difficult than for the JC69 model, the transition probability matrix (**P(t)**) is as follows:

$$P(t) = \begin{pmatrix} p0(t) & p1(t) & p2(t) & p2(t) \\ p1(t) & p0(t) & p2(t) & p2(t) \\ p2(t) & p2(t) & p0(t) & p1(t) \\ p2(t) & p2(t) & p1(t) & p0(t) \end{pmatrix}$$

Where:

$$p0(t) = 1/4.0 + 1/4.0 * \exp(-4*b*t) + 1/2.0 * \exp(-2*(a+b)*t)$$

$$p1(t) = 1/4.0 + 1/4.0 * \exp(-4*b*t) - 1/2.0 * \exp(-2*(a+b)*t)$$

$$p2(t) = 1/4.0 - 1/4.0 * \exp(-4*b*t)$$

As with the JC69 model, we can also create a program that will emulate the transition probability matrix with relative ease by inputting the parameter values for alpha (a), beta (b) and time (t). Also, organising the formulae of the transition probability matrix in a similar way to the JC69 model using Python defines the following function:

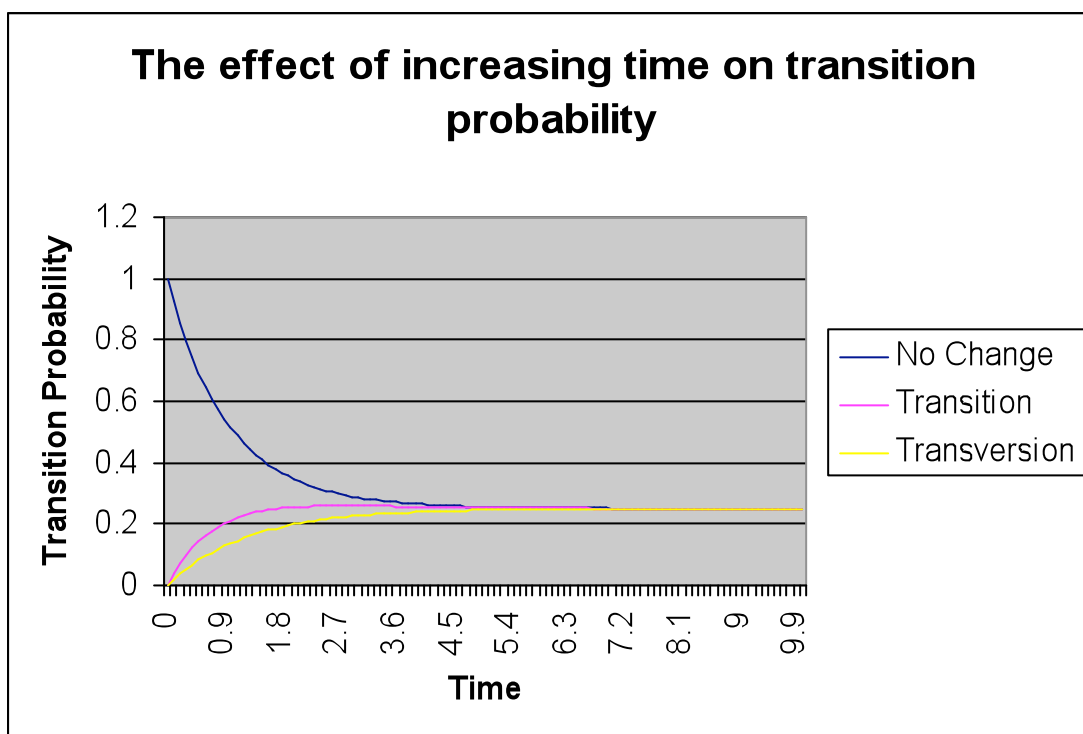
```
IDLE 1.2.2
>>> from math import *
>>> def K80(a,b,t):
    x = 1/4.0 + 1/4.0 * exp(-4*b*t) + 1/2.0 * exp(-2*(a+b)*t)
    y = 1/4.0 + 1/4.0 * exp(-4*b*t) - 1/2.0 * exp(-2*(a+b)*t)
    z = 1/4.0 - 1/4.0 * exp(-4*b*t)
    p = x
    q = y
    r = z
    matrixs = [[p,q,r,r],[q,p,r,r],[r,r,p,q],[r,r,q,p]]
    print matrixs

>>> # 'a' is alpha, 'b' is beta and 't' is time.
>>> K80(0.4,0.2,1)
[[0.51292934698540638, 0.21173513507320438, 0.13766775897069461, 0.13766775897069461],
 [0.21173513507320438, 0.51292934698540638, 0.13766775897069461, 0.13766775897069461],
 [0.13766775897069461, 0.13766775897069461, 0.51292934698540638, 0.21173513507320438],
 [0.13766775897069461, 0.13766775897069461, 0.21173513507320438, 0.51292934698540638]]
>>>
```

The function is tested using the parameters;  $a = 0.4$ ,  $b = 0.2$ ,  $t = 1$ . The transition probabilities for nucleotides experiencing no substitutions after  $t = 1$  are high, wherein the transition probabilities for transitions and transversions are relatively low in comparison. When considering the formulae used to calculate these probabilities, certain inevitable trends are recognisable:

$$\begin{aligned}
 x &= 1/4.0 + 1/4.0 * \exp(-4*b*t) + 1/2.0 * \exp(-2*(a+b)*t) \\
 y &= 1/4.0 + 1/4.0 * \exp(-4*b*t) - 1/2.0 * \exp(-2*(a+b)*t) \\
 z &= 1/4.0 - 1/4.0 * \exp(-4*b*t)
 \end{aligned}$$

$x$  represents the probability of a nucleotide experiencing no change over a given time. When  $t = 0$ ,  $x = 1$ : from this point,  $x$  decreases exponentially to the value of  $1/4$ .  $y$  represents the probability of a nucleotide experiencing a transition ( $A \leftrightarrow G$  or  $T \leftrightarrow C$ ) over a given time. At  $t = 0$ , the value of  $y$  is 0; when no time has passed, the probability of a given nucleotide experiencing any sort of substitution is 0. This is also true for transversional substitutions, represented by equation  $z$ . As time increases from 0, the transitional probabilities for both transversions and transitions increase, tending towards  $1/4$ . As the rates of transitional change are higher than those of transversional change, the transition probabilities for transitional substitutions increase towards  $1/4$  at a higher rate. The following graph represents the changes in the transitional probabilities of transitions, transversions and no substitution sites as time increases:



To create this graph, the values of alpha and beta were set to 0.4 and 0.2 respectively. These values simulate realistic values for the rates for transitions and transversions as observed rates have shown that transitional substitutions occur at a

higher frequency to transversional substitutions. Time ranges from 0 to 10, increasing by 0.1 within each interval.

### **HKY85 and TN93 Models**

Hasegawa, Kishino and Yano developed a model in 1985 that combined elements of both the K80 and F81 models. This is known as the HKY85 model and incorporates multiple parameters to create a more realistic simulation of how nucleotide sequences essentially behave. First of all, the HKY85 model assumes that the rates of substitution differ between each nucleotide. A single value would define the rates for a target nucleotide having been evolved into. For example, a value for the rate of T would define the rates by which any nucleotide would be substituted to result in the creation of the nucleotide T. These rates are known as base frequencies and within this model, the base frequencies are deemed unequal. Further parameters are included to distinguish between the rates of transitions and transversions as within the K80 model. After the first mitochondrial sequences were published in 1980, the difference between the rates of transitions and transversions was made definitive and so most nucleotide evolution models created after 1980 incorporate parameters that define the rates of transitions and transversions separately. The HKY85 model is seen to give a more accurate representation of nucleotide substitutions in comparison to the JC69, K80 and F81 models by accommodating multiple factors.

The following image represents the rate matrix **Q**:

$$Q = \begin{pmatrix} * & \kappa\pi_C & \pi_A & \pi_G \\ \kappa\pi_T & * & \pi_A & \pi_G \\ \pi_T & \pi_C & * & \kappa\pi_G \\ \pi_T & \pi_C & \kappa\pi_A & * \end{pmatrix}$$

The matrix is organised as the rate matrices for all previous models have been, the columns and rows are in the nucleotide order; T, C, A, G respectively. Within this representation of the matrix Q, K represents transitional substitutions. All other substitutions are assumed to be transversional other than the diagonal values of the matrix, which are not substitutions.  $\pi_T$  represents the rate of substitutions resulting in the formation of the nucleotide T as mentioned before.  $\pi_C$  represents the rate of substitutions resulting in the formation of the nucleotide C and so on.

Deriving the transition probability matrix (**P(t)**) is not as simple as with the previous models due to the matrix **Q** not being a diagonal matrix. Therefore, the matrix Q is initially diagonalized, followed by the exponentiation of the diagonal to produce the matrix **P(t)**:

```
[[ 'h', 'i', 'p', 'q'],
 [ 'j', 'k', 'p', 'q'],
 [ 'r', 's', 'l', 'm'],
 [ 'r', 's', 'n', 'o']]
```

Where:

$$\begin{aligned}
 h &= T + ((T*(A+G))/(T+C)) * \exp(-b*t) + (C/(T+C)) * \exp(-((T+C)*a+(A+G)*b)*t) \\
 i &= C + ((C*(A+G))/(T+C)) * \exp(-b*t) - (C/(T+C)) * \exp(-((T+C)*a+(A+G)*b)*t) \\
 j &= T + ((T*(A+G))/(T+C)) * \exp(-b*t) - (T/(T+C)) * \exp(-((T+C)*a+(A+G)*b)*t) \\
 k &= C + ((C*(A+G))/(T+C)) * \exp(-b*t) + (T/(T+C)) * \exp(-((T+C)*a+(A+G)*b)*t) \\
 l &= A + ((A*(T+C))/(A+G)) * \exp(-b*t) + (G/(A+G)) * \exp(-((A+G)*a+(T+C)*b)*t) \\
 m &= G + ((G*(T+C))/(A+G)) * \exp(-b*t) - (G/(A+G)) * \exp(-((A+G)*a+(T+C)*b)*t) \\
 n &= A + ((A*(T+C))/(A+G)) * \exp(-b*t) - (A/(A+G)) * \exp(-((A+G)*a+(T+C)*b)*t) \\
 o &= G + ((G*(T+C))/(A+G)) * \exp(-b*t) + (A/(A+G)) * \exp(-((A+G)*a+(T+C)*b)*t) \\
 p &= A * (1 - (\exp(-b*t))) \\
 q &= G * (1 - (\exp(-b*t))) \\
 r &= T * (1 - (\exp(-b*t))) \\
 s &= C * (1 - (\exp(-b*t)))
 \end{aligned}$$

Most of the transition probabilities differ for each substitution within this model; this more closely emulates how nucleotides would behave in real-life in comparison to the previous models. More factors are taken into account to achieve this and so the formulae increase in complexity as they accommodate a larger number of variables. Writing a function to carry out the formulae in the transition probability matrix is slightly more time-consuming than previous models but it is still achievable:

```

IDLE 1.2.2
>>> def HKY85(t,a,b,T,C,A,G):
    # a is alpha, T,C,A,G are the rates to each nucleotide, b is beta, t is time.
    hx = T + ((T*(A+G))/(T+C)) * exp(-b*t) + (C/(T+C)) * exp(-((T+C)*a+(A+G)*b)*t)
    ix = C + ((C*(A+G))/(T+C)) * exp(-b*t) - (C/(T+C)) * exp(-((T+C)*a+(A+G)*b)*t)
    jx = T + ((T*(A+G))/(T+C)) * exp(-b*t) - (T/(T+C)) * exp(-((T+C)*a+(A+G)*b)*t)
    kx = C + ((C*(A+G))/(T+C)) * exp(-b*t) + (T/(T+C)) * exp(-((T+C)*a+(A+G)*b)*t)
    lx = A + ((A*(T+C))/(A+G)) * exp(-b*t) + (G/(A+G)) * exp(-((A+G)*a+(T+C)*b)*t)
    mx = G + ((G*(T+C))/(A+G)) * exp(-b*t) - (G/(A+G)) * exp(-((A+G)*a+(T+C)*b)*t)
    nx = A + ((A*(T+C))/(A+G)) * exp(-b*t) - (A/(A+G)) * exp(-((A+G)*a+(T+C)*b)*t)
    ox = G + ((G*(T+C))/(A+G)) * exp(-b*t) + (A/(A+G)) * exp(-((A+G)*a+(T+C)*b)*t)
    px = A * (1 - (exp(-b*t)))
    qx = G * (1 - (exp(-b*t)))
    rx = T * (1 - (exp(-b*t)))
    sx = C * (1 - (exp(-b*t)))
    h = hx
    i = ix
    j = jx
    k = kx
    l = lx
    m = mx
    n = nx
    o = ox
    p = px
    q = qx
    r = rx
    s = sx
    matrix = [[h,i,p,q],[j,k,p,q],[r,s,l,m],[r,s,n,o]]
    print matrix

>>> from math import *
>>>
>>> HKY85(1,0.3,0.1,0.35,0.3,0.25,0.1)
[[0.88723486840062349, 0.079458227911962387, 0.023790645491010121, 0.0095162581964040489]
, [0.092701265897289331, 0.87399183041529649, 0.023790645491010121, 0.0095162581964040489]
], [0.033306903687414166, 0.028548774589212143, 0.9111501774013766, 0.026994144321997093]
, [0.033306903687414166, 0.028548774589212143, 0.067485360804992789, 0.8706589609183808]]
>>>

```

Parameters for time, transition rate, transversion rate and the base frequencies must be defined in order to generate the transition probability matrix. The function is then tested with experimental parameters, generating the matrix at the bottom of the image. At  $t = 0$ , the diagonal elements of  $P(t)$  are at 1 whilst all other values are at 0. This is because at  $t = 0$ , we would not expect any substitutions to have occurred to a nucleotide sequence. As time tends to infinity, the probabilities of the diagonal

elements decrease, as all other elements increase, to their respective base frequencies. This would be the result of the nucleotides in the sequence reaching a stationary distribution: when the proportions of each nucleotide match their respective base frequencies. These proportions would be maintained, as further substitutions would continue to generate the same proportions of nucleotides. Therefore, in this case, the stationary distribution is also the limiting distribution.

The difference between the rates of substitution of transitions and transversions was well established and resounds within most nucleotide models created after the K80 model. However, within transitions a further difference in rates can be distinguished. Nucleotides A and G are known as purine molecules and nucleotides T and C are known as pyrimidine molecules; the difference being the molecular structures of the nucleotides. Generally, purines and pyrimidines tend to have different rates of substitution; therefore, a more recent model to those discussed so far has been developed to accommodate for this factor. In 1993, Tamura and Nei proposed a new model, which included parameters that would distinguish between the rates of pyrimidines and purines respectively. This model is commonly known as the TN93 model and introduces the parameters; alpha 1 and alpha 2 in replacement of the single alpha parameter present in the HKY85 model for transitional rates. The rate matrix for this model is therefore very similar to that of the HKY85 model, as well as the transition probability matrix:

$$Q = m_{\tau} \times \begin{pmatrix} & A & C & G & T \\ A & * & \pi_C \alpha_1 & \pi_G & \pi_T \alpha_1 \\ C & \pi_A \alpha_1 & * & \pi_G \alpha_1 & \pi_T \alpha_2 \\ G & \pi_A & \pi_C \alpha_1 & * & \pi_T \alpha_1 \\ T & \pi_A \alpha_1 & \pi_C \alpha_2 & \pi_G \alpha_1 & * \end{pmatrix}$$

Matrix P(t) =

```
[[ 'h', 'i', 'p', 'q'],
 [ 'j', 'k', 'p', 'q'],
 [ 'r', 's', 'l', 'm'],
 [ 'r', 's', 'n', 'o']]
```

Where:

```
h = T + ((T*(A+G))/(T+C))*exp(-b*t) + (C/(T+C))*exp(-((T+C)*a1+(A+G)*b)*t)
i = C + ((C*(A+G))/(T+C))*exp(-b*t) - (C/(T+C))*exp(-((T+C)*a1+(A+G)*b)*t)
j = T + ((T*(A+G))/(T+C))*exp(-b*t) - (T/(T+C))*exp(-((T+C)*a1+(A+G)*b)*t)
k = C + ((C*(A+G))/(T+C))*exp(-b*t) + (T/(T+C))*exp(-((T+C)*a1+(A+G)*b)*t)
l = A + ((A*(T+C))/(A+G))*exp(-b*t) + (G/(A+G))*exp(-((A+G)*a2+(T+C)*b)*t)
m = G + ((G*(T+C))/(A+G))*exp(-b*t) - (G/(A+G))*exp(-((A+G)*a2+(T+C)*b)*t)
n = A + ((A*(T+C))/(A+G))*exp(-b*t) - (A/(A+G))*exp(-((A+G)*a2+(T+C)*b)*t)
o = G + ((G*(T+C))/(A+G))*exp(-b*t) + (A/(A+G))*exp(-((A+G)*a2+(T+C)*b)*t)
p = A * (1 - (exp(-b*t)))
q = G * (1 - (exp(-b*t)))
r = T * (1 - (exp(-b*t)))
s = C * (1 - (exp(-b*t)))
```

### **Simulation of nucleotide sequences**

The previously discussed models of nucleotide substitution all allow for the generation of probabilities that determine how a nucleotide sequence will or has evolved based on likelihood. From this, a function can be used to simulate how a sequence of nucleotides may evolve based on these probabilities. For example, taking the principles of the simplest model, JC69, we can say that the probabilities for a nucleotide changing into one of the other nucleotides are equal. Therefore, when simulating a scheduled substitution of a nucleotide, because each transition probability is the same, the target nucleotide can be randomly chosen and the sequence mutated. If the transition probabilities were unequal, the target nucleotide would be randomly chosen but with incorporated bias favouring more probable transitions.

A function must be designed to first generate a random time at which a mutation will occur based on the total substitution rates of all the nucleotides of the sequence using the rate matrix **Q**. A time interval over which mutations will occur must be outlined, for simplicity the interval from  $t = 0$  to  $t = 1$  is used often (**timex**). To begin mutation, a sequence of nucleotides must be provided; through the use of a function, a nucleotide sequence of any length can be generated (**genseq**). Using the **timex** function, a list of times is generated when a rate is inputted into the function. In this case, the total rate for all nucleotides of the sequence is inputted and a list of times generated randomly, these times are used as the times of mutation. This technique cannot be used for more complex models of nucleotide evolution as they assume unequal transition probabilities and so after a substitution, the total rate would change with the departure of one nucleotide and the creation of a new nucleotide.

In basing simulation using the JC69 model; the transition probability matrix for the JC69 model is used to generate the probabilities for mutations or for no changes. The **genseq** and **timex** functions are both used to generate a sequence of nucleotides and to then create a list of times at which mutations will take place. Please look to the functions sections towards the end of this report for definitions of each function. The following is a sequence of nucleotides before and after mutation using the JC69 transition probability matrix:

Before

```
>>> genseq(10)
['G', 'G', 'A', 'T', 'C', 'C', 'G', 'C', 'A', 'A']
>>>
```

After

```
['G', 'A', 'G', 'T', 'A', 'C', 'C', 'C', 'A', 'G']
>>>
```

Although 5 differences are visible from the initial sequence to the sequence after mutation, 7 actual mutations had occurred with two of the mutations acting on the same starting nucleotide, the 8<sup>th</sup>, with the second mutation returning the 8<sup>th</sup>

nucleotide back to its starting state (nucleotide C). 7 mutations were achieved using the **timex** function and inputting a value of 4.5 for rate (at).

Simulation of mutation using the K80 model requires a slightly different method, as does simulation using the HKY85 and TN93 models due to the differing principles and parameters between each model. These principles are quite easily summarisable:

- K80- as transitions and transversions must be distinguished between as they occur at different rates, the function written for simulating mutation under the probabilities generated by the K80 model accounts for this. This then results in transition mutations and transversion mutations occurring at different rates to the nucleotide sequence being mutated accordingly.
- HKY85- As the HKY85 model utilises several different parameters and therefore rates to distinguish probabilities, the function written to simulate under the principles of this model uses multiple rates when conducting a mutation. Also, as each nucleotide is subject to different rates of mutation, the total rate by which any mutation will occur using the **timex** function is updated after any nucleotide is mutated and changed into another to account for this change.
- TN93- the function simulating mutation under the principles of the TN93 model acts in the same way as the function used for the HKY85 model. The only difference is that the TN93 model introduces an additional rate, breaking the rate for alpha (transitions) into alpha1 (transitions between pyrimidines) and alpha2 (transitions between purines).

The functions written for the simulation of the mutation of a nucleotide sequence are included in the appendix and are labelled accordingly.

### ***Maximum Likelihood Estimates (MLE) - JC69 & K80 Models***

Maximum likelihood estimates are used to estimate parameter values for a statistical model when applying that model to a dataset. In the case of nucleotide substitutions, the statistical models fitted to data are the models of nucleotide substitution and the parameter estimated is the value for rate and time. Rate and time are dealt with as a single value as they cannot be distinguished from one another; the single value (**at**) can be produced by the product of a number of different combinations of values of either alpha or time.

The dataset used will be two sequences of nucleotides of equal lengths of which one sequence will be assumed to have evolved from the other through several mutations. The total length of a sequence is represented by the letter **n** and the differences (numbers of nucleotides which differ between each sequence) is represented by the letter **k**.

### ***JC69***

To explain the theory behind acquiring the maximum likelihood estimate, the binomial distribution must be considered. The following is the probability mass function (pmf) of the binomial distribution:

$$\binom{n}{k} p^k (1 - p)^{n-k}$$

$n$  = The total length of a sequence.

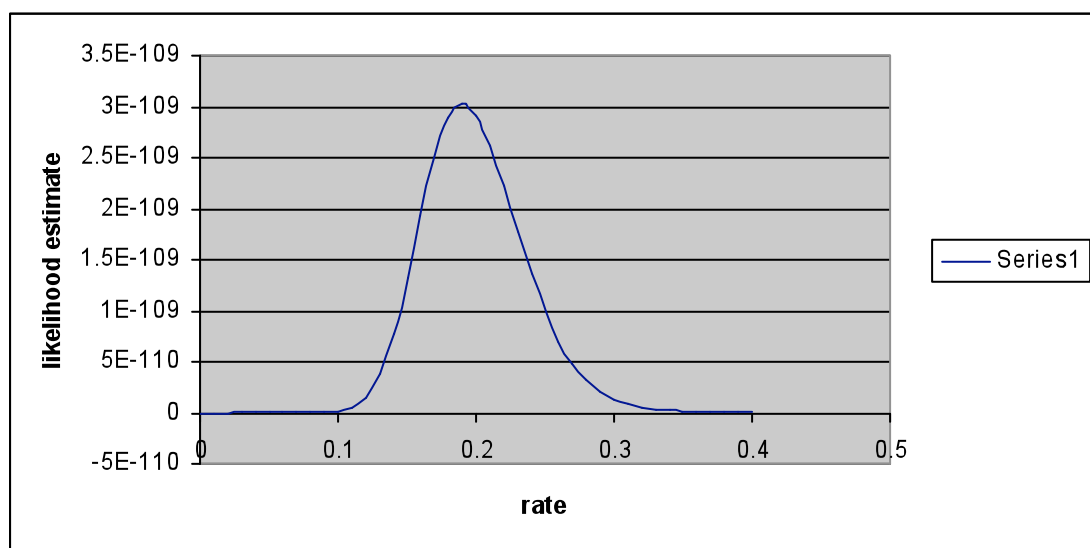
$k$  = The number of differences between the two sequences.

The probability mass function is used to calculate the probability when a variable ( $at$ ) is exactly equal to the value proposed for the variable. For example, if a value for  $at$  is inputted into the probability mass function, the value calculated will represent the probability that the value for  $at$  used to calculate the probability is correct. In replacement of the variable  $p$  is the equation used in the transition probability matrix for the JC69 model to calculate the probability of a mutation occurring. The equation used in replacement of  $1-p$  is the equation from the transition probability matrix of the JC69 model used to calculate the probability of a mutation not occurring. The following equation is the probability mass function, altered to include the variables mentioned above with the total length of a sequence ( $n$ ) as 100 and the number of differences ( $k$ ) as 40. The notation  $\text{pow}(x,y)$  represents the value  $x$  to the power of  $y$ :

Probability mass function =  $l$

$$l = \text{pow}(1/4.0 + 3/4.0 * \exp(-4*m), 60) * \text{pow}(1/4.0 - 1/4.0 * \exp(-4*m), 40)$$

The variable  $m$  represents the value  $at$ . Finding the value of  $at$  with the highest probability can be found through trial and error, however using PYTHON all values of  $at$  within an interval can be tested and plotted onto a graph:



The probability mass function equation displayed above was used to generate the data to plot this graph. The values of  $m$  ( $at$ ) within the interval 0 to 0.4 were tested and a peak probability was acquired. The peak represents the value of  $m$  ( $at$ ) with the highest probability of resulting in the value of  $k$  and therefore is the maximum likelihood estimate. In this case, the maximum likelihood estimate is 0.19 for  $at$ .

### **K80**

To find the maximum likelihood estimate using the principles of the K80 model is approached in a very similar way as with the JC69 model. The probability mass

function is adjusted so that two values are estimated as there are two parameters for rates in the K80 model, alpha and beta.

$$Pmf = p0^{(n - k - j)} * p1^k * p2^j$$

Where:  $p0$  = the equation used from the transition probability matrix of the K80 model to calculate the probability of no mutation occurring.

$p1$  = the equation used from the transition probability matrix of the K80 model to calculate the probability of a transition mutation occurring.

$p2$  = the equation used from the transition probability matrix of the K80 model to calculate the probability of a transversion mutation occurring.

$n$  = the total length of a sequence.

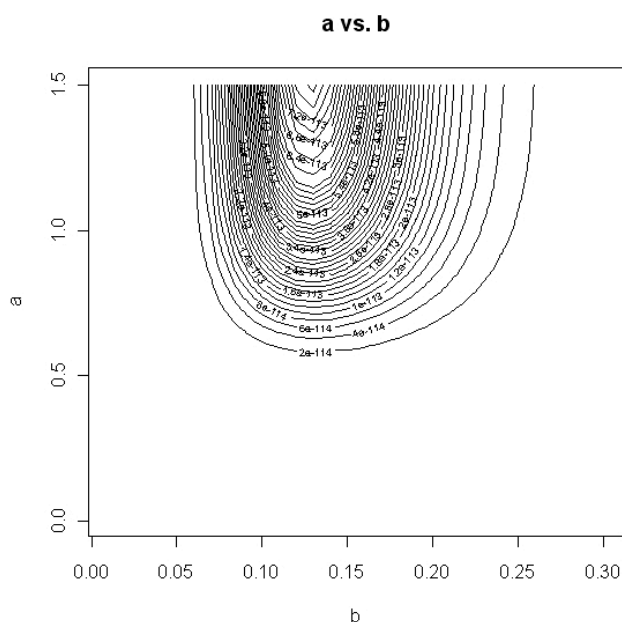
$k$  = the number of differences between two sequences that have resulted from transition mutations.

$j$  = the number of differences between two sequences that have resulted from transversion mutations.

Probability mass function =  $l$

$$l = \text{pow}(1/16.0 + (1/16.0 * \exp(-4 * b)) + (1/8.0 * \exp(-2 * (a + b))), 60) * \text{pow}(1/16.0 + 1/16.0 * \exp(-4 * b) - 1/8.0 * \exp(-2 * (a + b)), 30) * \text{pow}(1/16.0 - 1/16.0 * \exp(-4 * b), 10)$$

**a** and **b** represent the values for the rates of transitions (alpha) and transversions (beta) respectively. Using PYTHON a table can be generated showing the probabilities of a value of **a** being most likely when **b** is of another value. The values in this table can be plotted graphically using a contour plot. The following is a contour plot generated using the equation for probability mass function displayed above, however the total length of a sequence (**n**) is 100, the number of differences that have resulted from transition mutations (**k**) is 30 and the number of differences between two sequences that have resulted from transversion mutations is 10:



The lines become concentrated around the maximum likelihood estimates for the values of alpha (rate of transitions) and beta (rate of transversions). The estimate for the most probable value of  $\mathbf{b}$  is clearly centred on the interval between 0.12 and 0.14. Unfortunately, the value for  $\mathbf{a}$  is not visible as the limits of this contour graph do not show where the lines of the graph centre on the y-axis.

Maximum likelihood estimates are used in conjunction with models of nucleotide evolution mainly to estimate the time taken for one sequence of nucleotides to evolve into another, assuming that one sequence is the ancestor of the other. Although only a value for  $\mathbf{at}$ , the product of both rate and time, is achievable if an average rate (or rates in the case of multiple parameter models) is known. Using the known value for rate, the variable of time can be distinguished and so the time taken for one sequence to mutate into the other is calculatable. Practically, biologists and statisticians have adopted this method when attempting to calculate the time taken for particular species (such as humans) to have evolved from ancestral species (such as lesser evolved primates). By assessing the same sections of DNA from the two species of the same length, the number of differences may be recorded used to estimate a time using the maximum likelihood method.

### ***Conclusions***

As my investigation was not an experiment as such but rather the translation of statistical models onto software so as to use these models in practical situations, my conclusion would be to state that the programmes that I have written to emulate these statistical models have been successful and so may be applied to practical datasets. This translational link, between statistical models and new computing software embodies the basic principles of bioinformatics and allows demonstrations of how statisticians and biologists can therefore use these models when dealing with mutated sequences of DNA.

If I had further research time and possibly slightly more options in terms of computing software, there are multiple areas that I would have expanded within my project and report. First of all, I would have included a step-by-step explanation of the Taylor Series expansion allowing for readers to understand the mathematical theory behind obtaining the transition probability matrix from the rate matrix of a nucleotide model. Also, I would have explored further models of nucleotide evolution, as there are many more significant models that have not been mentioned. These models would have broadened the scope of my project and would have depicted further steps by which each model was chronologically improved. Within the last section of this report, the maximum likelihood estimation of the JC69 and K80 models, I believe that this section could be progressed further. With access to alternative computing software that could plot multi-dimensional graphs, I would have extended the calculation of maximum likelihood estimates into estimating the parameters for the HKY85 and TN93 models.

### ***References:***

- Computational Molecular Evolution (Yang 2006)
- [www.wikipedia.org](http://www.wikipedia.org)

- [www.python.org](http://www.python.org)
  - <http://docs.python.org/lib/module-random.html>
  - <http://docs.python.org/lib/module-random.html>
  - [http://www.tau.ac.il/~doronadi/F81\\_model.doc](http://www.tau.ac.il/~doronadi/F81_model.doc)
  - [http://www.megasoftware.net/WebHelp/part\\_iv\\_evolutionary\\_analysis/computing\\_evolutionary\\_distances/distance\\_models/nucleotide\\_substitution\\_models/hc\\_jukes\\_cantor\\_distance.htm](http://www.megasoftware.net/WebHelp/part_iv_evolutionary_analysis/computing_evolutionary_distances/distance_models/nucleotide_substitution_models/hc_jukes_cantor_distance.htm)
  - <http://www.ncbi.nlm.nih.gov/books/bv.fcgi?rid=hmg.figgrp.1080>
  - <http://www.ncbi.nlm.nih.gov/books/bv.fcgi?rid=hmg.figgrp.1080>
- <http://www.ncbi.nlm.nih.gov/books/bv.fcgi?rid=hmg.figgrp.1080>
- Evolutionary Trees from DNA Sequences:
    - A Maximum Likelihood Approach (Joseph Felsenstein 1981)
    - A Novel Use of Equilibrium Frequencies in Models of Sequence Evolution (Nick Goldman and Simon Whelan)

## Functions

**Genseq**- the generation of a random sequence of nucleotides is essential to the simulation of nucleotide substitution. To define a function to generate a sequence, a parameter for the length of the sequence must be defined. In this case, **n** is used. The function randomly chooses a letter, representing each nucleotide, from the list "ACGT" using the in-built 'randint' function. The chosen letter is added to a list; the process of choosing a letter is then repeated **n** times creating a list or 'sequence' **n** nucleotides long.

```
>>> def genseq(n):
    list = []
    for i in range (1,n+1):
        i = "ACGT"[randint(0,3)]
        list = list + [i]
    print list
```

**Timex**- this function allows for the generation of a cumulative set of times that represent when mutations will occur to a nucleotide sequence. This function is only used within the simpler models of substitution as it assumes that transition probabilities are the same for each nucleotide. An in-built function (random.expovariate) takes a value for rate as a parameter and generates another value using this rate value. Inputting a higher rate value will increase the probability of the in-built function generating a smaller value. Values are generated using the same rate value and are displayed cumulatively to represent the times at which events occur according to the inputted rate value. This process is terminated when the cumulative time value increases over 1 as we are only interested in mutations occurring within times 0 and 1. This function is effective, as theoretically, if events occur at a higher rate, more events will occur in a given time.

```
>>> def timex(e):
    t = random.expovariate(e)
    while t < 1:
        print t
        t = t + random.expovariate(e)
    print t
```

**Intgen**- this function was created to generate a list, of length **n**, of random numbers. These random numbers denote at what points mutations will occur. The **timex** function is initially used to calculate the number of mutations that will occur in an allotted time. The number of calculated mutations will then signify the length of the

list of random numbers. Each number within this list refers to the  $n^{\text{th}}$  nucleotide of a sequence being mutated. That nucleotide will then be mutated.

```
>>> def intgen(n,p):  
    list = []  
    for i in range(1,n+1):  
        x = randint(0,p-1) * (i/i)  
        list = list + [x]  
    print list
```

## Appendix:

### K80 - Function for simulation of mutation of nucleotide sequence

```
>> def tran(S,V):
    p = random.expovariate(S)
    y = random.expovariate(V)
    if p < y:
        print "S"
    else: print "V"

>> def solveK80(n):
    for i in range(0,n):
        x = list[list2[i]]
        y = tran(0.3,0.1)
        if y == 'S':
            if x == 'A':
                list[list2[i]] = 'G'
            else:
                if x == 'G':
                    list[list2[i]] = 'A'
                else:
                    if x == 'T':
                        list[list2[i]] = 'C'
                    else:
                        if x == 'C':
                            list[list2[i]] = 'T'
        else:
            if x == 'A':
                list[list2[i]] = "GCT"[randint(0,2)]
            else:
                if x == 'G':
                    list[list2[i]] = "ACT"[randint(0,2)]
                else:
                    if x == 'C':
                        list[list2[i]] = "AGT"[randint(0,2)]
                    else:
                        if x == 'T':
                            list[list2[i]] = "ACG"[randint(0,2)]

    print list
```

## HKY85 - Function for simulation of mutation of nucleotide sequence

```

>>> def HKY85(a,b,T,C,A,G,m):
    hx = T + ((T*(A+G))/(T+C))*exp(-b*1) + (C/(T+C))*exp(-(T+C)*a+(A+G)*b)*1
    ix = C + ((C*(A+G))/(T+C))*exp(-b*1) - (C/(T+C))*exp(-(T+C)*a+(A+G)*b)*1
    jx = T + ((T*(A+G))/(T+C))*exp(-b*1) - (T/(T+C))*exp(-(T+C)*a+(A+G)*b)*1
    kx = C + ((C*(A+G))/(T+C))*exp(-b*1) + (T/(T+C))*exp(-(T+C)*a+(A+G)*b)*1
    lx = A + ((A*(T+C))/(A+G))*exp(-b*1) + (G/(A+G))*exp(-(A+G)*a+(T+C)*b)*1
    mx = G + ((G*(T+C))/(A+G))*exp(-b*1) - (G/(A+G))*exp(-(A+G)*a+(T+C)*b)*1
    nx = A + ((A*(T+C))/(A+G))*exp(-b*1) - (A/(A+G))*exp(-(A+G)*a+(T+C)*b)*1
    ox = G + ((G*(T+C))/(A+G))*exp(-b*1) + (A/(A+G))*exp(-(A+G)*a+(T+C)*b)*1
    px = A * (1 - (exp(-b*1)))
    qx = G * (1 - (exp(-b*1)))
    rx = T * (1 - (exp(-b*1)))
    sx = C * (1 - (exp(-b*1)))
    hs = hx
    i = ix
    js = jx
    ks = kx
    ls = lx
    ms = mx
    ns = nx
    os = ox
    ps = px
    qs = qx
    rs = rx
    ss = sx
    TT = i + ps + qs
    TC = js + ps + qs
    Tt = rs + ss + ms
    TG = rs + ss + ns
    x = ((list.count('T'))* TT) + ((list.count('C')) * TC) + ((list.count('A')) * Tt) + ((list.count('G')) * TG)
    t = random.expovariate(x)
    while t < 1:
        B = list[randint(0,m-1)]
        if B == 'A':
            f = random.expovariate(rs)
            g = random.expovariate(ms)
            c = random.expovariate(ss)
            if f < g and f < c:
                list[p] = 'T'
            else:
                if g < f and g < c:
                    list[p] = 'G'
                else:
                    if c < f and c < g:
                        list[p] = 'C'
        else:
            if B == 'G':
                f = random.expovariate(rs)
                a = random.expovariate(ns)
                c = random.expovariate(ss)
                if f < a and f < c:
                    list[p] = 'T'
                else:
                    if a < f and a < c:
                        list[p] = 'A'
                    else:
                        if c < f and c < a:
                            list[p] = 'C'
            else:
                if B == 'C':
                    f = random.expovariate(js)
                    a = random.expovariate(ps)
                    g = random.expovariate(qs)
                    if f < a and f < g:
                        list[p] = 'T'
                    else:
                        if a < f and a < g:
                            list[p] = 'A'
                        else:
                            if g < f and g < a:
                                list[p] = 'G'
            else:
                if B == 'T':
                    c = random.expovariate(i)
                    a = random.expovariate(ps)
                    g = random.expovariate(qs)
                    if c < a and c < g:
                        list[p] = 'C'
                    else:
                        if a < c and a < g:
                            list[p] = 'A'
                        else:
                            if g < a and g < c:
                                list[p] = 'G'
    x = ((list.count('T'))* TT) + ((list.count('C')) * TC) + ((list.count('A')) * Tt) + ((list.count('G')) * TG)
    t = t + random.expovariate(x)
    print list

```

## TN93 - Function for simulation of mutation of nucleotide sequence

```

>>> def solveTN93 (a1,a2,b,T,C,A,G,m):
    hx = T + ((T*(A+G))/(T+C))*exp(-b*1) + (C/(T+C))*exp(-((T+C)*a1+(A+G)*b)*1)
    ix = C + ((C*(A+G))/(T+C))*exp(-b*1) - (C/(T+C))*exp(-((T+C)*a1+(A+G)*b)*1)
    jx = T + ((T*(A+G))/(T+C))*exp(-b*1) - (T/(T+C))*exp(-((T+C)*a1+(A+G)*b)*1)
    kx = C + ((C*(A+G))/(T+C))*exp(-b*1) + (T/(T+C))*exp(-((T+C)*a1+(A+G)*b)*1)
    lx = A + ((A*(T+C))/(A+G))*exp(-b*1) + (G/(A+G))*exp(-((A+G)*a2+(T+C)*b)*1)
    mx = G + ((G*(T+C))/(A+G))*exp(-b*1) - (G/(A+G))*exp(-((A+G)*a2+(T+C)*b)*1)
    nx = A + ((A*(T+C))/(A+G))*exp(-b*1) - (A/(A+G))*exp(-((A+G)*a2+(T+C)*b)*1)
    ox = G + ((G*(T+C))/(A+G))*exp(-b*1) + (A/(A+G))*exp(-((A+G)*a2+(T+C)*b)*1)
    px = A * (1 - (exp(-b*1)))
    qx = G * (1 - (exp(-b*1)))
    rx = T * (1 - (exp(-b*1)))
    sx = C * (1 - (exp(-b*1)))
    hs = hx
    i = ix
    js = jx
    ks = kx
    ls = lx
    ms = mx
    ns = nx
    os = ox
    ps = px
    qs = qx
    rs = rx
    ss = sx
    TT = i + ps + qs
    TC = js + ps + qs
    TA = rs + ss + ms
    TG = rs + ss + ns
    x = ((list.count('T')) * TT) + ((list.count('C')) * TC) + ((list.count('A')) * TA) + ((list.count('G')) * TG)
    t = random.expovariate(x)
    while t < 1:
        p = randint(0,m-1)
        B = list[p]
        if B == 'A':
            f = random.expovariate(rs)
            g = random.expovariate(ms)
            c = random.expovariate(ss)
            if f < g and f < c:
                list[p] = 'T'
            else:
                if g < f and g < c:
                    list[p] = 'G'
                else:
                    if c < f and c < g:
                        list[p] = 'C'
        else:
            if B == 'G':
                f = random.expovariate(rs)
                a = random.expovariate(ns)
                c = random.expovariate(ss)
                if f < a and f < c:
                    list[p] = 'T'
                else:
                    if a < f and a < c:
                        list[p] = 'A'
                    else:
                        if c < f and c < a:
                            list[p] = 'C'
            else:
                if B == 'C':
                    f = random.expovariate(js)
                    a = random.expovariate(ps)
                    g = random.expovariate(qs)
                    if f < a and f < g:
                        list[p] = 'T'
                    else:
                        if a < f and a < g:
                            list[p] = 'A'
                        else:
                            if g < f and g < a:
                                list[p] = 'G'
                else:
                    if B == 'T':
                        c = random.expovariate(i)
                        a = random.expovariate(ps)
                        g = random.expovariate(qs)
                        if c < a and c < g:
                            list[p] = 'C'
                        else:
                            if a < c and a < g:
                                list[p] = 'A'
                            else:
                                if g < a and g < c:
                                    list[p] = 'G'
    x = ((list.count('T')) * TT) + ((list.count('C')) * TC) + ((list.count('A')) * TA) + ((list.count('G')) * TG)
    t = t + random.expovariate(x)
    print list

```