

Discrete time molecular evolution

Thomas Mailund

September 8, 2007

We can model the evolution of a single nucleotide over time as a Markov chain with four states, one for each possible base A , C , G , and T . This might be a bit simplistic, since we are ignoring selection working on DNA and such, but for selectively neutral nucleotides it seems reasonable enough. If we encode A , C , G , and T as 1, 2, 3, and 4, we can encode the Markov chain as the matrix

$$P = \begin{pmatrix} p_{A \rightarrow A} & p_{A \rightarrow C} & p_{A \rightarrow G} & p_{A \rightarrow T} \\ p_{C \rightarrow A} & p_{C \rightarrow C} & p_{C \rightarrow G} & p_{C \rightarrow T} \\ p_{G \rightarrow A} & p_{G \rightarrow C} & p_{G \rightarrow G} & p_{G \rightarrow T} \\ p_{T \rightarrow A} & p_{T \rightarrow C} & p_{T \rightarrow G} & p_{T \rightarrow T} \end{pmatrix}$$

where $p_{X \rightarrow Y}$ is the probability of mutating nucleotide X to nucleotide Y in one time unit (say a generation).

This is a very general model, with three free parameters per row (think about it!). By putting various constraints on the transition probabilities we get different classical models. One of the earliest models is the Jukes-Cantor model with only a single parameter, α :

$$P = \begin{pmatrix} 1 - 3\alpha & \alpha & \alpha & \alpha \\ \alpha & 1 - 3\alpha & \alpha & \alpha \\ \alpha & \alpha & 1 - 3\alpha & \alpha \\ \alpha & \alpha & \alpha & 1 - 3\alpha \end{pmatrix}$$

This model assumes that, in each generation, the probability of mutating X to $Y (\neq X)$ is α for all $Y (\neq X)$. The stationary distribution is $\pi = (\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4})$ (you should be able to show this in a few seconds!). It is not a particularly realistic model, but it sure is simple.

We will do a few exercises with the Jukes-Cantor model. In practise you would rarely use the model—it is too simplistic by far—but everything we do here generalises to more sophisticated models so we might as well start simple.

Simulating the evolution of a nucleotide: Write a function that, given nucleotide X , the mutation probability α , and a number of generations, g , evolves X into nucleotide Y (i.e. Y should be sampled from row X of P^g).

For modelling a string of DNA, we often assume that each nucleotide in the sequence evolves independently of all others. This is even less realistic than assuming Markov properties and such, but we need to come up with *some* kind of mathematical model, and this at least is simple to work with, so there goes...

A sequence Markov chain: Show that the model for sequences (of fixed length, N) is also a Markov chain. How many states does this Markov chain has? What is the probability $P_{i,j}$? What is the stationary distribution?

Simulating the evolution of a nucleotide sequence: Write a function that, given a sequence length, n , simulates X from the stationary distribution. Then write a function that, given X , the mutation probability α , and a number of generations, g , evolves X for g generations to obtain the sequence Y .

If possible, avoid using `for` loops here. They are about the slowest feature in R and shouldn't be used for iterating through long sequences. Remember that `sample()` takes a length parameter, so for X there is no need for a loop. For generating Y , have a look at the function `sapply()`.

Imagine now the following experiment: We take a bacteria DNA sequence, X , and put it in a petri dish. Then we let it evolve for g generations to obtain a new sequence, Y . Can we use this to learn something about α ?

We can define the *likelihood* of α , given X , Y and g , as the function

$$\text{lhd}(\alpha \mid X, Y, g) = \text{Prob}(X \rightarrow Y \mid g, \alpha)$$

that is, for each value of α , $\text{lhd}(\alpha \mid X, Y, g)$ is the probability of observing X evolving into Y in g generations.

Calculating the likelihood: Write a function that takes α , X , Y and g as parameters and calculates $\text{lhd}(\alpha \mid X, Y, g)$. To avoid loops, have a look at `mapply()`.

Can you also express the likelihood only in terms of identical and different nucleotides in X and Y ? You can use the (element by element) `==` test to count the number of equal and the number of different nucleotides:

```
no.eq <- sum(X == Y)
no.di <- sum(X != Y)
```

When working with the likelihood of α , we would prefer to keep X , Y , and g fixed and only have a function of α . In R we can write a function like that using so called *higher order functions*. The only magical thing about higher order functions is that functions can be considered values and returned from functions. So say you named the function above `full.lhd(alpha, X, Y, g)`, then you can define a function of α alone, `lhd(alpha)` using the function `make.lhd`:

```
make.lhd <-
  function(X, Y, g) function(alpha) full.lhd(alpha, X, Y, g)
lhd <- make.lhd(X, Y, g)
```

Make sure you understand what I am doing here. It is really quite useful.

Try simulating X and Y (for various g) and plot $\text{lhd}(\alpha \mid X, Y, g)$ for $\alpha \in [0, \frac{1}{3}]$. (Why this interval?) Try increasing the sequence lengths. If you run into

problems with *really* small values, try changing to log space and calculate $\log \text{lhd}(\alpha | X, Y, g)$ instead.

Maximum Likelihood estimates: When trying to estimate α from X, Y and g we will typically use the *maximum likelihood approach* and pick

$$\hat{\alpha} := \operatorname{argmax}_{\alpha} \text{lhd}(\alpha | X, Y, g).$$

From your experiments (and Fig.1) you should have a good idea about why.

The function `nlm()` can be used for general numerical minimisation. So you can estimate α using:

```
ML.alpha <- nlm(function(alpha) -log.lhd(alpha), 0.1)$estimate
```

(where we use `-log.lhd()` since `nlm()` minimises and we want to maximise). Often, this is the best we can do, but if you can be more clever about optimisation you are usually better off not relying on `nlm()`. This is one such case: the expression for $\text{lhd}(\alpha | X, Y, g)$ is quite simple, so you should be able to identify critical points analytically. This means that we can get the maximum with a few arithmetic operations instead of running an optimisation algorithm; always a better choice. We are not going to bother, though, 'cause the expression for P^g as a function of α is a bit tedious to work with, but if we really wanted to we could solve it. (Yeah!)

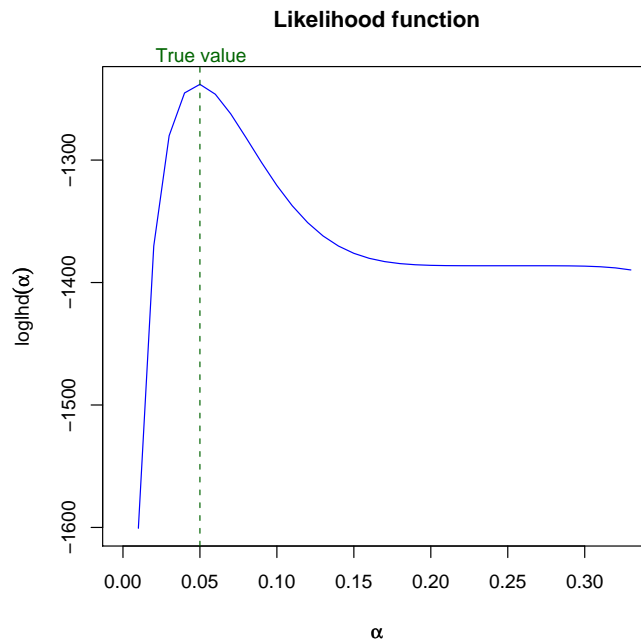


Fig. 1: Likelihood for α . X and Y had length 1000, X was evolved over 5 generations with the true α value of 0.05. The maximum likelihood estimate is pretty good here, but try playing around with the parameters and see what happens.

```
g <- 5
true.alpha <- 0.05
X <- sample.seq(1000)
Y <- evolve.seq(X,true.alpha,g)
log.lhd <- make.log.lhd(X,Y,g)

alphas <- seq(0,1/3,by=0.01)
plot(alphas,sapply(alphas,log.lhd),type='l',col='blue',
     main='Likelihood function',
     ylab=expression(loglh(alpha)), xlab=expression(alpha))
abline(v=true.alpha, col='darkgreen', lty='dashed')
mtext('True value',3,at=true.alpha,col='darkgreen')
```

Fig. 2: The R code used to produce Fig.1.