HyPhy: Hypothesis Testing Using Phylogenies

Sergei L. Kosakovsky Pond[1] and Spencer V. Muse[2]

[1] Antiviral Research Center, University of California, San Diego CA 92103, `spond@ucsd.edu`,

[2] Bioinformatics Research Center, North Carolina State University, Raleigh NC, 27695-7566, `muse@stat.ncsu.edu`

1 Introduction

The field of molecular evolution, though wide-reaching in its breadth, can be compartmentalized into two types of investigations: studies of phylogeny, and studies of the molecular evolutionary process. Of course, each of these two categories encompasses many different types of questions, and many investigations require studies of both phylogeny and evolutionary process, but the proposed binary classification is a useful construct. Software for molecular evolution is focused disproportionately on problems relating to phylogenetic reconstruction, with a number of outstanding comprehensive packages to choose from. On the other hand, software for addressing questions of the molecular evolutionary process tends to be found in stand-alone programs that answer only one or two quite specific problems. The *HyPhy* system, available for download from `www.hyphy.org`, was designed to provide a unified platform for carrying out likelihood-based analyses on molecular evolutionary data sets, the emphasis of analyses being the molecular evolutionary process, that is, studies of rates and patterns of the evolution of molecular sequences.

HyPhy consists of three major components: a high-level programming language designed to facilitate the rapid implementation of new statistical methods for molecular evolutionary analysis; a collection of pre-written analyses for carrying out widely-used molecular evolutionary methods; a graphical user interface that allows users to quickly and interactively analyze data sets of aligned sequences using evolutionary models and statistical methods that they design using the software system. This chapter is intended to provide an overview of the key elements of each of the three system components, including both specific details of the basic functionality as well as a conceptual description of the potential uses of the software. The nature of the package prevents the creation of an exhaustive "cookbook" of available methods. Instead, we hope to provide a collection of fundamental tools and concepts that allow users to begin using *HyPhy* to carry out both existing and new methods of data analysis.

1.1 Standard Analyses

The second of the three enumerated *HyPhy* components was a collection of pre-written "standard" analyses. Since this section of the software is essentially just a collection of prepackaged analyses, we will not devote much time to a detailed discussion. However, we choose to describe it first in this chapter to illustrate the types of analyses that *HyPhy* has been designed to address. In Fig. 1 we show the initial Standard Analyses menu invoked by ANALYSES:STANDARD ANALYSES... (note the use of SMALL CAPS to indicate menu items, with submenus or selections separated by a colon). Each of the nine major headings includes a collection of routines that can be selected by the user. For example, the POSITIVE SELECTION menu item expands to offer five different analyses relating to the task of identifying nucleotide sites undergoing positive selection. A total of 35 batch files are included in the collection, and most of these files include a variety of options enabling users to select items such as evolutionary models or topology search methods. Topics include molecular clock tests, positive selection analyses, phylogenetic reconstruction, and model comparison procedures. The authors frequently add new standard analysis to the package. *HyPhy* includes the ability to perform web updates, which ensures that the distribution is kept up to date.
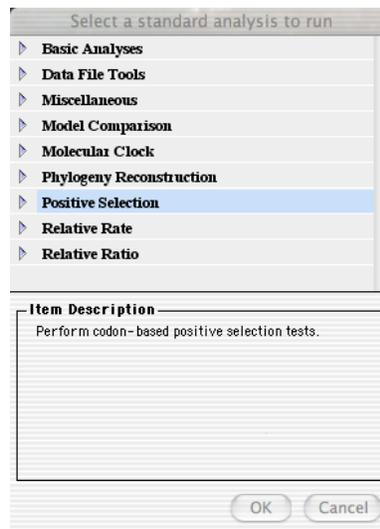

Fig. 1. *HyPhy* Standard Analyses Menu (Mac OS X)

2 Model fitting and hypothesis testing with the *HyPhy* Graphical User Interface.

2.1 Basic Analysis

The fundamental component of likelihood analyses of molecular evolutionary data is to fit a given phylogenetic tree with a specified model of evolution to an alignment and obtain maximum likelihood estimates (MLE) of all independent model parameters, which commonly include branch length parameters and character substitution rates (Felsenstein, 1981). Before we demonstrate how to use *HyPhy* for simple model fitting, we will introduce the fundamental components required of virtually every *HyPhy* data analysis.

1. Data Set. A *Data Set* is a multiple sequence alignment. *HyPhy* is able to read a variety of sequence formats, including as NEXUS, PHYLIP and FASTA.
2. Data Filter. A *Data Filter* specifies a part (or parts) of a data set. *HyPhy* provides powerful tools to select sites and sequences from a data set to analyze. The simplest data filter specifies the entire data set. Examples of non-trivial filters include: every first and second position in a codon, exon-intron-exon arrangements, or alignments sites matching a particular motif, such as glycosylation sites. We will often refer to data filters as *partitions*.
3. Substitution Models. We also need to provide stochastic models describing how character substitutions occur along branches in a phylogenetic tree. *HyPhy* includes a multitude of standard "named" models and provides unparalleled flexibility for the users to define their own models. A substitution model is specified by its *instantaneous rate matrix* and the vector of equilibrium character frequencies. For instance, one of the most commonly used nucleotide substitution models is the HKY85 model (Hasegawa et al., 1985), whose instantaneous rate matrix is given by:

$$Q = \begin{array}{c} \\ A \\ C \\ G \\ T \end{array} \overset{\begin{array}{cccc} A & C & G & T \end{array}}{\left(\begin{array}{cccc} \star & \kappa\pi_C & \pi_G & \kappa\pi_T \\ \kappa\pi_A & \star & \kappa\pi_G & \pi_T \\ \pi_A & \kappa\pi_G & \star & \kappa\pi_T \\ \kappa\pi_A & \pi_C & \kappa\pi_G & \star \end{array} \right)},$$

where κ denotes the ratio of transversion and transition rates and π_i is the base frequency of nucleotide i, $i = A, C, G, T$. We use $\star$ as a notation to indicate that the diagonal elements of rate matrices are defined so that the sum of each row in the rate matrix is 0. This condition ensures that the *probability transition matrix*

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

defines a proper transition probability function (i.e. the sum of each row in P is 1).

4. Tree. A phylogenetic tree specifies the evolutionary history of extant sequences, represented in the data set. It can either be given, or inferred from the data/model combination. While most other software packages force the evolutionary process to follow the same model along every branch, in *HyPhy* the user can have multiple models, with different rate matrices at each branch. Therefore the notion of the tree in *HyPhy* is more than just the evolutionary relationships, but rather the *combination* of a tree topology AND substitution models attached to tree branches. The distinction in *HyPhy* between a tree and a topology is an important one, as we will illustrate through later examples.

5. Likelihood Function. A combination of a data filter and a tree (which includes both topology and model information) is sufficient to define the probability of the observed data given model parameter values, *i.e.*, the likelihood function. The likelihood function object in *HyPhy* is a convenient way to combine multiple data filter/tree objects (with shared or distinct model parameters) into a single likelihood function, which can be then be maximized to obtain MLEs of all model parameters.

Example: Basic Analysis

We are now conceptually prepared to set up the simplest nucleotide sequence analysis with the help of *HyPhy* graphical user interface. Our example data set is the p51 subunit of the reverse transcriptase gene of HIV-1, obtained as one of the reference alignments from the Los Alamos HIV database, `hiv-web.lanl.gov`. This data set is included as an example file with *HyPhy* distribution.

Preparing the data.

First we must load the sequence alignment. We accomplish this by starting *HyPhy* and selecting File:Open:Open Data File menu command from the *HyPhy* console window. The file we wish to open is named `p51.nex` and can be found in the `data` directory of *HyPhy* standard installation. Alternatively, all example alignments used in this chapter can be downloaded from `www.hyphy.org/pubs/HyphyBookChapter.tgz`

    HyPhy will load the sequences and open a **Data Panel** (Fig. 3) We will explore some the features of the data panel interface in later examples. For now, we wish to define a data filter (partition), in this case - simply the entire alignment. Select all sites in the alignment by using the Edit:Select All menu command and create a new partition by choosing Data:Selection→Partition. The program creates a data filter with all the sites selected in the sequence viewer, assigns a default name and color to the partition, updates the navigation bar and selects the newly created partition. One can edit the name and color of a partition by double clicking on the partition row in the "Analysis Setup" area, or choosing Data:Partition
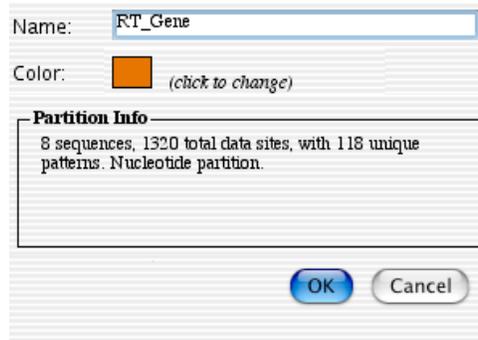
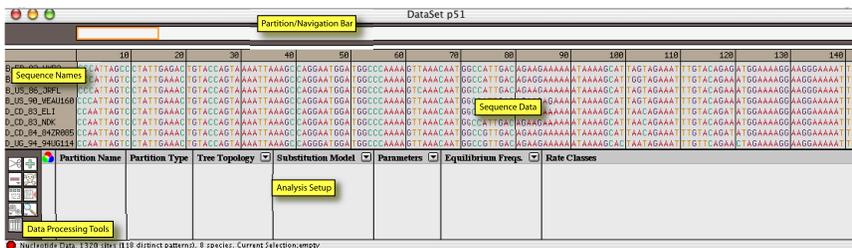**Fig. 2.** Partition properties dialog

Fig. 3. *HyPhy* data panel (Mac OS X)

PROPERTIES, with the partition row selected. Let us rename the partition to "RT_Gene" (for technical reasons *HyPhy* doesn't allow spaces in the names of partitions) as shown in Fig. 2.

Specifying the model.

Once the data has been filtered, we may assign a tree topology and a model to the partition by clicking on the pulldown arrows in the appropriate columns of the "Analysis Setup" table (Fig. 4). The data file `p51.nex` already in-
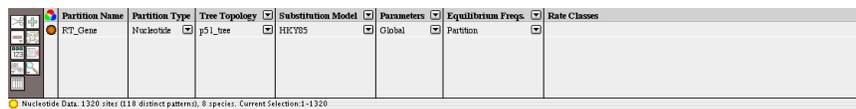

Fig. 4. Analysis Setup

cluded a tree topology, automatically loaded by *HyPhy* and made available in the "Tree" pulldown list. For the model, let us choose substitution matrix

HKY85, with Global parameters (in this case meaning that there is a single transition transversion ratio κ for every branch in the tree) and equilibrium frequencies gathered from the partition, so that entries of the frequency vector π are simply the frequencies of characters observed in the data. Once all the necessary analysis components have been successfully assigned to at least one data partition (RT_Gene in this case), the status light in the bottom left corner of the window will change from red to yellow, indicating that we are now ready to create a likelihood function.

Likelihood Function.

We will denote the likelihood function of the model parameters Θ, given a data set $\mathcal{D}$ and a tree $\mathcal{T}$ by

$$L(\Theta|\mathcal{D}, \mathcal{T})$$

HyPhy is then able to obtain maximum likelihood parameter estimates $\hat{\Theta}$ by maximizing $L(\Theta|\mathcal{D}, \mathcal{T})$ over the possible values of Θ.
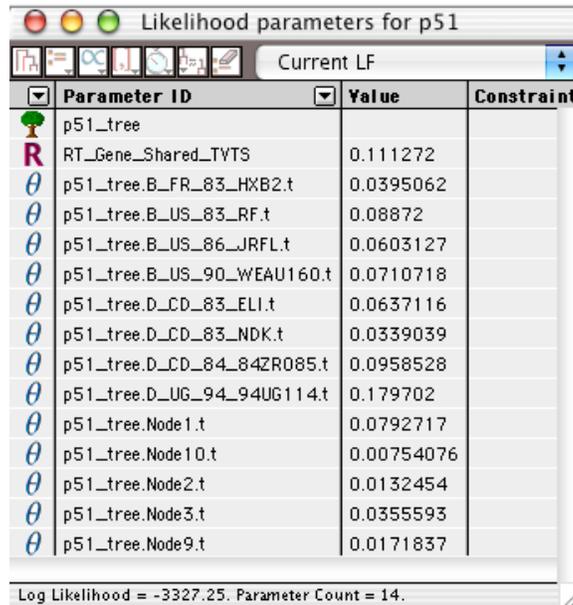
Let us now create and optimize the likelihood function. First, we select LIKELIHOOD:BUILD FUNCTION. *HyPhy* creates the likelihood function as requested and prints out some diagnostic messages to the console:

```
Created likelihood function 'p51_LF' with
 1  partitions,
 1  shared parameters,
 13 local parameters,
 0  constrained parameters.
Pruning efficiency 764 vs 1534 (50.1956 % savings)
```

The number local parameters refer to the branch length parameters t. An unrooted binary tree on n sequences, will have a total of $2N - 3$ branches. In our case $N = 8$ and thus there are 13 branch length parameters to estimate. Pruning efficiency numbers show the computational savings that *HyPhy* was able to realize using the column sorting ideas of (Kosakovsky Pond and Muse, 2004). Now, choose LIKELIHOOD:OPTIMIZE to instruct *HyPhy* to proceed with fitting selected models to the data and obtaining parameter MLEs.

Results.

We are now ready to examine model fitting results. For this example, *HyPhy* produces maximum likelihood estimates of 14 model parameters by numerical optimization of the likelihood function. The program reports a text summary to the console, and also opens a graphical parameter table display, as shown in (Fig. 5). The status bar of the parameter table displays a one-line snapshot of the likelihood analysis: maximum log-likelihood for our RT dataset was estimated at -3327.25, and 14 independently adjusted parameters were used. Knowledge of these two quantities is sufficient to evaluate various information theoretic criteria for relative goodness of fit, such as Akaike Information

Fig. 5. Graphical Parameter Display

Criterion (Akaike, 1974) as well as to perform likelihood ratio tests for nested models.

Notice how *HyPhy* groups items in the parameter table by class: trees, global parameters (shared by all tree branches) and local parameters (those which affect a single branch); each item is labeled both by name and with an appropriate icon. The single global parameter is the transversion:transition ratio, κ, of the HKY85 model, and is labeled as RT_Gene_Shared_TVTS. By default, each shared parameter is prefixed with the name of the data partition to which it is attached (RT_Gene in this case). While at first the names of local parameters may appear confusing, *HyPhy* uses a uniform naming scheme for all local model parameters: *tree name.branch name.parameter name*. For instance, p51_tree.B_FR_83_HXB2.t, refers to a local parameter t along the branch ending in B_FR_83_HXB2 in the tree p51_tree. Leaf names in the tree correspond to sequence names in the data file, while $NodeN$, where N is an integer, are default names given to unlabeled internal nodes in the tree (users can give internal nodes custom names as well). Parameter estimates can be exported in a variety of formats by invoking FILE:SAVE.

Let us now open a tree window to visualize the evolutionary distances between HIV-1 sequences in the example by double clicking on the tree row in the parameter table. *HyPhy* will open a tree viewer panel, as shown in Fig. 6. A common measure used to assess evolutionary distances is the expected number of substitutions per site E_s along a particular branch, equal to the weighted

trace of the rate matrix:

$$E_s = -\sum_j \pi_j Q_{jj} \tag{1}$$

The *HyPhy* tree viewer automatically scales branches on E_s, although the scaling may be changed by the user.
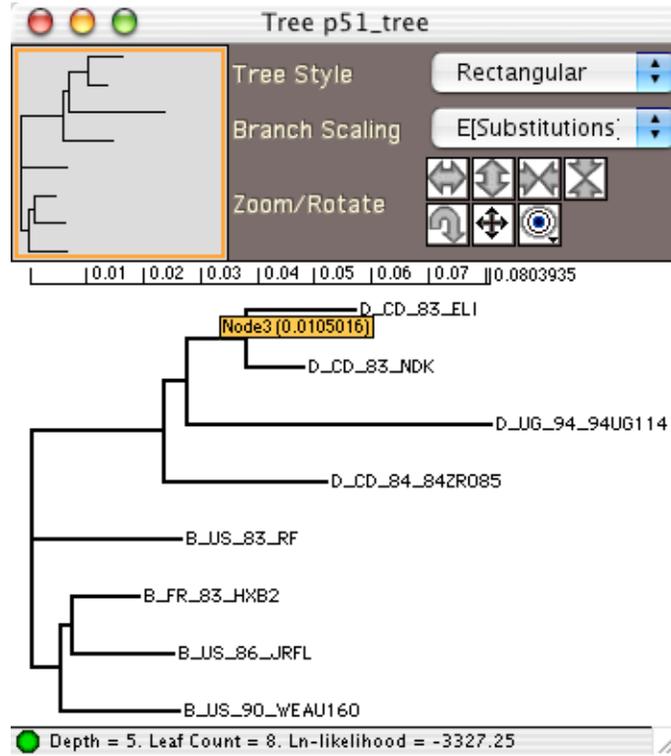
Fig. 6. *HyPhy* Tree Viewer for `p51.nex`, scaled on the expected number of substitutions per site inferred by HKY85, with an example of a tooltip branch length reporter

Confidence Intervals

All parameter estimates will be affected by stochastic noise, arising out of finite data sample size. For instance, substitution bias parameters often have large relative estimation errors, when compared to branch length parameters. *HyPhy* allows the user to obtain confidence intervals on parameter estimates using the asymptotic normality of MLEs. Likelihood theory states that MLEs of model parameters are distributed as multivariate normal around the "true"

parameter values, and the covariance matrix of the normal distribution can be estimated by inverting the observed Fisher information matrix:

$$\hat{I}\left(\hat{\Theta}\right) = \left(\left. \frac{\partial^2 \log L(\Theta|\mathcal{D}, \mathcal{T})}{\partial \theta_i \partial \theta_j} \right|_{\Theta=\hat{\Theta}} \right)$$

The Fisher information matrix measures the curvature of the log-likelihood surface. Flat surfaces around the maximum do not inspire high confidence in estimated parameter values, while steep surfaces lead to sharp estimates.

  HyPhy can be instructed to construct the covariance matrix as well as the confidence intervals for each parameter estimate based on the estimated variance of the normal distribution, either for every parameter, or for selected parameters (conditioned on the values of others). Select all the parameters in the table by choosing EDIT:SELECT ALL and then LIKELIHOOD: COVARI-ANCE AND CI, and set "Estimation Method" to "Asymptotic Normal[finer]" in the ensuing dialog box. "Crude" and "Finer" estimates differ in how *HyPhy* computes the Fisher Information Matrix (which must be done numerically, because analytic derivatives of the likelihood function are not available in general). *HyPhy* will open two *chart windows* - the 95% confidence interval window for all selected parameters, and the covariance matrix.
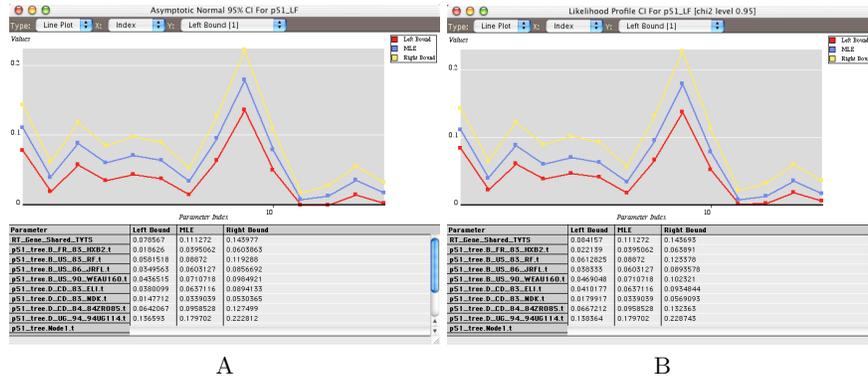
Fig. 7. *HyPhy* confidence interval estimates using (A) asymptotic normality of MLEs and (B) Profile plots using 95% levels of χ_1^2

Likelihood Profile

Confidence intervals based on asymptotic normality rely upon many assumptions that may be violated for short alignments or parameter-rich models. For example, such confidence intervals are always symmetric about the maximum likelihood estimate, and if the likelihood surface is "skewed" around the MLE, such intervals may be a poor representation of the real variance in parameter

estimates. A second approach to determining statistical support for a parameter value estimate is to employ *likelihood profile* confidence intervals, obtained by inverting a likelihood ratio test.

Suppose we wish to compute a confidence interval CI_i^α of level α for a single model parameter θ_i. A common method is to first fix all other model parameters $\theta_{i'}, i' \neq i$ at their maximum likelihood estimates. We can now think of the likelihood function, as a function of a single parameter θ_i. Thus, a restricted version of the full likelihood function is

$$\bar{L}(\theta_i) = L(\theta_i | \mathcal{D}, \mathcal{T}, \hat{\theta}_{i'}).$$

Clearly, the maximum likelihood estimate for θ_i using the restricted likelihood is the same as that given by the full likelihood function: $\hat{\theta}_i$.

Consider two hypotheses: $H_0 : \theta_i = x$ versus $H_A : \theta_i \neq x$. These hypotheses can be tested using the restricted likelihood function and a one degree of freedom likelihood ratio test (assuming that $\hat{\theta}_{i'}$ is not on the boundary of the parameter space):

$$2[\log \bar{L}(\hat{\theta}_i) - \log \bar{L}(x)] \sim \chi_1^2$$

If $\hat{\theta}_j$ is on the boundary, then the asymptotic distribution changes to

$$2[\log \bar{L}(\hat{\theta}_i) - \log \bar{L}(x)] \sim \frac{\chi_1^2 + \chi_0^2}{2}$$

Using this observation, a confidence region can be defined as all those values x for which we fail to reject H_0, i.e. all those x for which the likelihood ratio statistic is less than the α percentile of the corresponding χ^2 or mixture distribution. If we also assume that the likelihood function is monotone (has no local maxima), then we find the boundaries of the confidence interval by tracing the log-likelihood function plot until the desired difference from the maximum is obtained in both directions (see Fig. 8).

There are a couple of issues with this approach: (i) we assume sufficient data for the asymptotic likelihood distributions to be applicable, which may fail for short alignments, or models which are too parameter rich; and (ii) we are obtaining the confidence intervals for one parameter at a time, rather than a confidence region for all parameters (this is mostly due to technical difficulties with finding such a region when there are many model parameters), thus ignoring covariation among parameter estimates.

The first issue may be resolved, to an extent, by accepting or rejecting H_0 using a non-LRT criterion, such as AIC (Akaike, 1974) The procedure is exactly the same, but the cut-off level is no longer determined by the asymptotic χ^2 distribution, but rather by an information theoretic parameter addition penalty. For AIC, $2[\log \bar{L}(\hat{\theta}_i) - \log \bar{L}(x)] \leq 2$ would place x in the confidence interval.

Also, to see how reasonable the asymptotic normality assumption is, one could check whether a quadratic approximation to the log-likelihood holds

well. The quadratic approximation for the log restricted likelihood around the maximum likelihood estimate $\hat{\theta}_i$ can be derived from a Taylor series expansion:

$$\log \bar{L}(x) \approx \log \bar{L}(\hat{\theta}_i) + \left.\frac{d}{d\theta_i} \log \bar{L}(\theta_i)\right|_{\hat{\theta}_i} \left(x - \hat{\theta}_i\right) + \left.\frac{d^2}{d\theta_i^2} \log \bar{L}(\theta_i)\right|_{\hat{\theta}_i} \left(x - \hat{\theta}_i\right)^2$$

Because $\hat{\theta}_i$ maximizes the likelihood function, the first derivative term vanishes and we have the desired quadratic approximation

$$\log \bar{L}(x) - \log \bar{L}(\hat{\theta}_i) \approx \left.\frac{d^2}{d\theta_i^2} \log \bar{L}(\theta_i)\right|_{\hat{\theta}_i} \left(x - \hat{\theta}_i\right)^2$$

By plotting the likelihood profile and the quadratic approximation on the same graph, one can see how well the χ^2 approximation to the likelihood ratio test will work. *HyPhy* offers each of the above confidence interval estimation
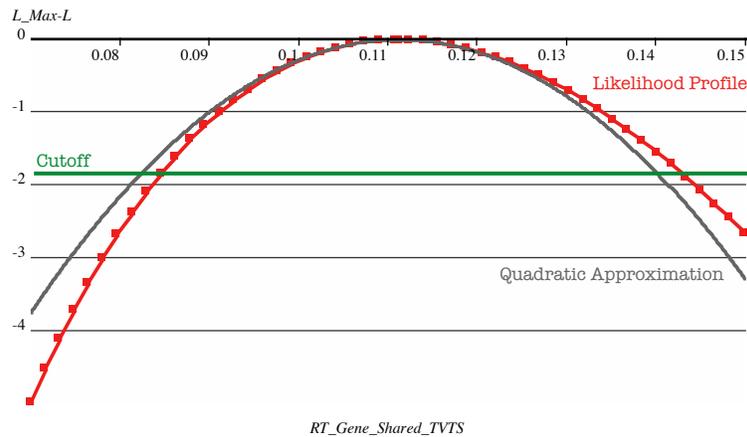
Fig. 8. Likelihood Profile plot, with a quadratic approximation and a 95% χ_1^2 cutoff levels

techniques via Likelihood:Covariance and CI and Likelihood:Profile Plot from the parameter table window.

Saving the analysis.

HyPhy can store all the information needed to recreate the analysis we just performed in a single NEXUS file. This feature can be invoked by switching back to the data panel, selecting File:Save and choosing the format option to include the data in the file. Let us save this simple analysis as `p51_HKY85.bf` in the "Saves" directory of the *HyPhy* installation.

2.2 Local Branch Parameters

Almost all treatments of likelihood analysis of molecular sequence data assume that there is only one parameter per branch in the phylogenetic tree - branch length - and that other model parameters are shared by all branches. However, it may be often be desirable to relax this assumption. For example, to test if a group of branches (such as a single lineage or a clade) have different substitution process parameters than the rest of the tree, it is necessary to compare likelihoods of constrained and unconstrained models. *HyPhy* provides a general mechanism for defining an arbitrary number of branch-specific and shared model parameters. Consider the HKY85 model discussed in the previous section. Rewrite the rate matrix as:

$$Q(\alpha, \beta, \pi) = \begin{array}{c} \\ A \\ C \\ G \\ T \end{array} \begin{array}{cccc} A & C & G & T \\ \begin{pmatrix} \star & \beta\pi_C & \alpha\pi_G & \beta\pi_T \\ \beta\pi_A & \star & \beta\pi_G & \alpha\pi_T \\ \alpha\pi_A & \beta\pi_G & \star & \beta\pi_T \\ \beta\pi_A & \alpha\pi_C & \beta\pi_G & \star \end{pmatrix} \end{array}$$

This may seem like a different matrix altogether, but if one sets $t = \alpha$ and $\kappa = \beta/\alpha$, we return to the previous parameterization if $\beta > 0$. In fact, this new parameterization allows the transition rate (α) to be 0, and transvertion rate (β) to be non-zero, whereas the first (more common) parameterization does not. Even more importantly, we can now let each branch have a separate α and β, which is equivalent to allowing every branch to have its own transition/transversion ratio. We declare such a model to be fully local, as opposed to the fully global model of the previous section. Obviously, there is a range of intermediate models where some of the branches share transition/transversion ratios while others are free to vary.

To specify the fully local HKY85 model in *HyPhy* for our example dataset, all that must be done differently is to select "Local" in place of "Global" in the "Parameters" column of the analysis setup table in Fig. 4. You can either start a new analysis from scratch, or continue from where we left off in the global analysis of the previous section. In the latter scenario, *HyPhy* will display a warning message, because changing substitution models causes a fundamental change in the likelihood function (i.e. a different set of parameters and rate matrices). Next, invoke LIKELIHOOD:BUILD FUNCTION and observe that the resulting likelihood function has 26 local parameters (2 per branch, as requested). Upon selecting LIKELIHOOD:OPTIMIZE, a parameter table is once again shown, and we observe that the log-likelihood has improved to -3320.84. A quick glance at the likelihood score improvement of 7 units for 12 additional parameters suggests that there is insufficient evidence favoring the fully local model over the fully global model.

The rate parameter names in the parameter table for this analysis end with "trst" and "trsv", which hopefully mean "transition" and "transversion". *HyPhy* allows one to look at the rate matrix and map parameter names to

what they actually stand for, in case parameter names are less descriptive. To see how that is done, let us open the "Object Inspector" window (use WINDOW:OBJECT INSPECTOR on the Mac and FILE:OBJECT INSPECTOR on Windows). In the newly opened window (Fig. 9(A)) select "Models" from the pulldown option list, and scroll through the rather long list of models, until you find one in bold (meaning that this model is currently used in an active likelihood function), named "RT_Gene_HKY85_local". Again, the name of the data partition is incorporated in the model identifier for easy reference. Double click on that model and examine the rate matrix as shown in Fig. 9(B). The equilibrium frequencies for this model (π) are the actual proportions of A,C,G and T in the RT gene alignment, and "trst" are indeed the rates for $A \leftrightarrow G$ and $C \leftrightarrow T$ substitutions, while "trsv" are the rates for all other substitutions. By default, *HyPhy* will automatically multiply rate matrix entries by the appropriate π, hence there is no need to include them in the rate matrix explicitly.
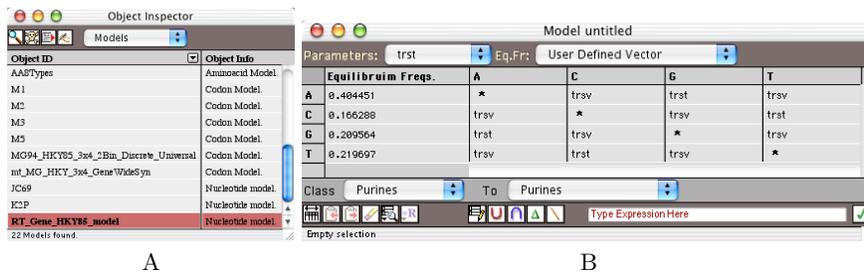

Fig. 9. (A) Models in the "Object Inspector" (B) HKY85 local model for the RT gene

Let us now open the tree window for the local model (Fig. 10(A)). Recall that branch lengths are given by (1). The tree looks very similar to the global HKY85 tree from Fig. 6. However, a more interesting comparison would be to see if the transition and transversion ratios vary from branch to branch. *HyPhy* allows scaling of tree display on any local model parameter - "trst" and "trsv" in this instance.

Double click on the tree name in the parameter table once again to open another instance of the tree window - very useful for side by side comparisons. Scale one of the trees on "trst" and another on "trsv" (Fig. 10(B,C)). Notice how while the shapes are still similar, branch lengths are not quite proportional between trees, while they would be if all branch transition/transversion ratios were the same.

As a matter of fact, *HyPhy* tree viewer allows scaling on any function of model parameters. Let us define the transition/transversion ratio parameter. For every branch it is simply: $ratio = trsv/trst$. To define this scaling pa-
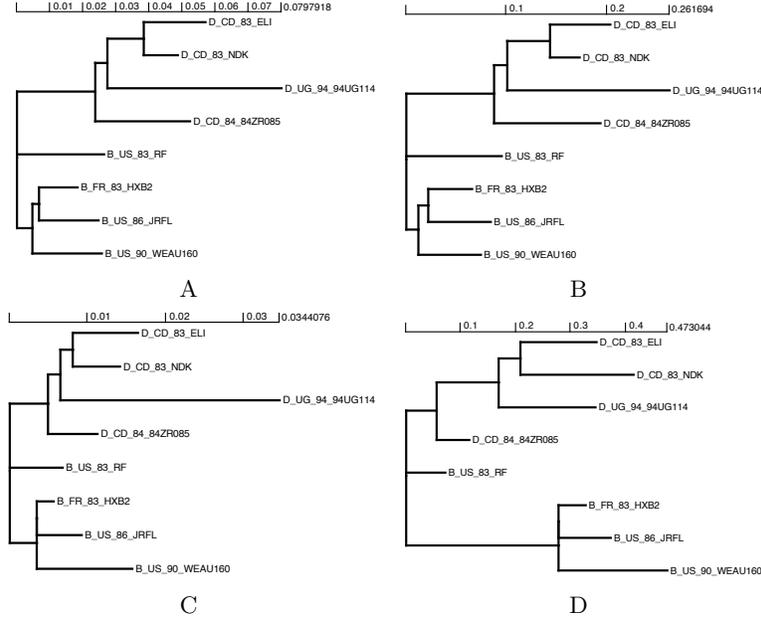
Fig. 10. RT gene tree under HKY85 local model scaled on (A) expected number of substitutions per site (B) transition rates (C) transversion rates (D) transversion/transition ratios

rameter, switch to a tree window, select all branches (EDIT:SELECT ALL) and choose TREE:EDIT PROPERTIES. The dialog box which appears shows all available local branch parameters. Click on the "Add User Expression" button (the + icon), type in the formula for the expression, rename it to "ratio", and select "OK" (Fig. 11). *HyPhy* has added "ratio" to the list of local parameters (not estimable parameters, but rather functions of other parameters). You can view the value of each branch ratio in the parameter table, and scale the tree on the transition/transversion ratio (Fig. 10(D)). The difference in branch to branch ratios are quite striking.

The *HyPhy* tree viewer can automatically label each branch of the tree with any function of branch model parameters. As an example, we will label each branch with the number of transitions E_t and the expected number of transversions E_v per site expected to occur along that branch per alignment site. For the HKY85 local model

$$E_t = 2\beta(\pi_A\pi_G + \pi_C\pi_T), \quad E_v = 2\alpha[(\pi_A + \pi_G)(\pi_C + \pi_T)].$$

Note that E_t and E_v add up to the total branch length and are linear functions of the rates. Substituting the actual values of π for our data set (Fig. 9 B), we get
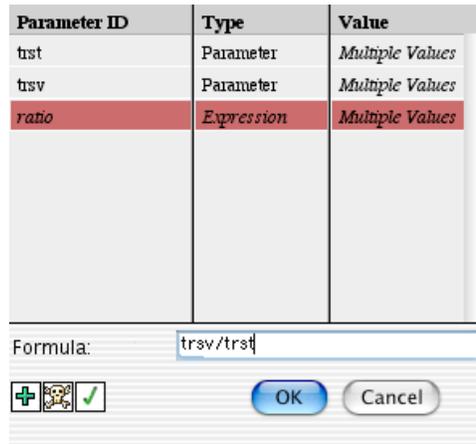
Fig. 11. New scaling parameter dialog

$$E_t = 0.242583\beta, \quad E_v = 0.474001\alpha.$$

Employ the same process we did for adding the ratio parameter, and define $E_t = 0.242583 * trst$ and $E_v = 0.474001 * trsv$. Now use Tree:Branch Labels:Above Branches and Tree:Branch Labels:Below Branches to label each branch with E_t and E_v, adjust fonts and alignments to your liking and check "Scale tree by resizing window" in the dialog opened with Tree:Tree Display Options. The final display should look like Fig. 12.

2.3 Multiple partitions and hypothesis testing.

Early attempts to model molecular evolution of protein coding sequences used the observation that the evolution in the first and second position of a codon and the third position follow markedly different patterns. Indeed, for universal genetic code, every substitution in the second codon position is nonsynonymous, i.e. it changes the protein encoded by the codon. For the first position all but 8 of all possible substitutions, which do not lead to stop codons, are nonsynonymous. In contrast, at the 3rd position 126 out of 176 substitutions are synonymous. Because random nonsynonymous substitutions are likely to be deleterious, it is commonly assumed that the substitution rate for the third position is different (typically much higher) than that in the first and second position. Our next task is to define a *HyPhy* analysis that treats first and second codon positions as one data partition and the third codon position as another, and to then fit a collection of models to the data. We will continue using the HIV-1 p51 subunit of the RT gene data set from `p51.nex`.

First, open the data panel with `p51.nex` and select all the sites in the alignment. Next, invoke one of the numerous data filtering tools in *HyPhy* - the combing tool, by clicking on the comb tool button in the data panel
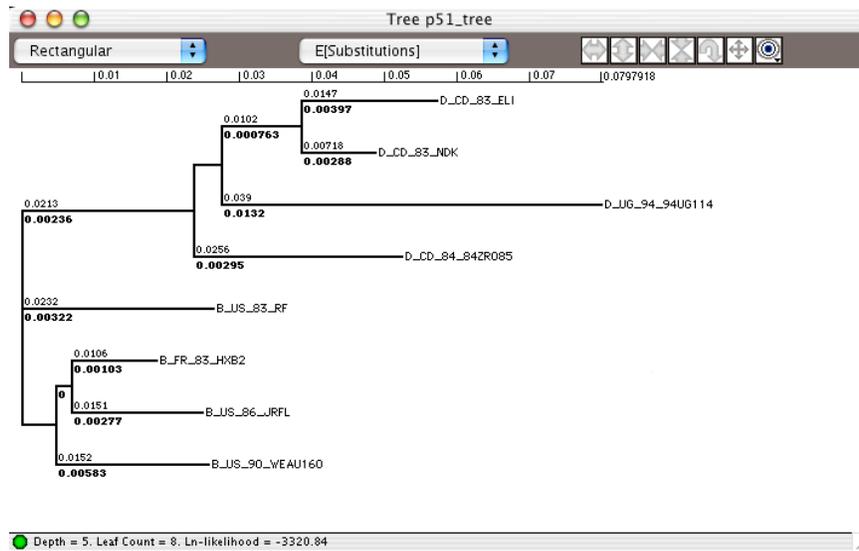
Fig. 12. RT tree scaled on expected number of substitutions per site, and labeled with the expected number of transitions and transversions per site (above and below, respectively)

(Fig. 3). To select the first two position in every codon, we need a comb of size 3, with first and second sites selected, and the third - omitted. In the combing dialog, set the size of the comb to 3 and check the boxes next to positions 1 and 2. Repeat the process, to define the partition with every third codon position (make sure that the first partition is NOT highlighted in the analysis set up table while you are applying the second comb, otherwise *HyPhy* will comb the partition again, effectively selecting every third column in the data partition of the first and the second positions we have just created). Rename the partitions to "First_Second" and "Third" respectively. Assign the same tree topology to both data partitions, the HKY85 model, with global parameter options and equilibrium frequencies collected separately from each partition. In the end, the data panel should look like in Fig. 13.

When we build the likelihood function, *HyPhy* prints out a message

```
Tree topology p51_tree was cloned for partition Third.
```

It is important to understand that while both partitions share the tree *topology*, for *HyPhy* a tree means *both* topology and models/parameters. The two partitions need to have two trees, with independent branch lengths and transversion/transition ratio parameters, κ_{12} and κ_3, assigned the names First_Second_Shared_TVTS and Third_Shared_TVTS by *HyPhy*.
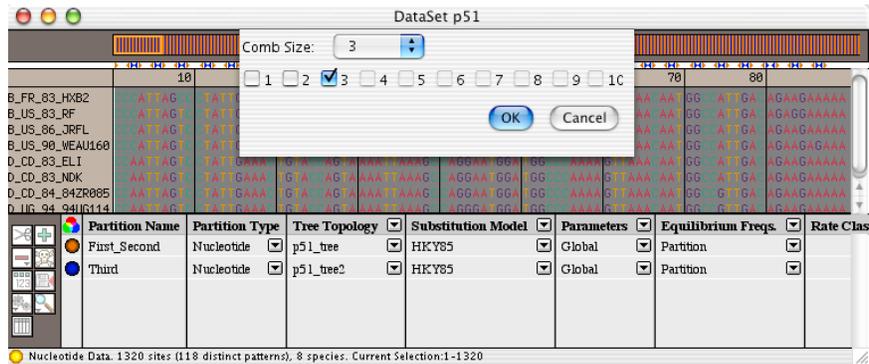
Fig. 13. Data panel with two data partitions and a comb filter dialog

After the models are fitted to the data, we observe that both the shapes of the trees (Fig. 14), and the transversion/transition ratios (0.198 versus 0.067) differ quite a lot between the partitions.

A careful reader might correctly point out that the analysis we have just performed could have been done by fitting HKY85 to each of the partitions separately. However, we will now illustrate what the joint likelihood function of both partitions can offer in terms of hypothesis testing.

Simple Hypothesis Testing.

Consider the null hypothesis $H_0 : \kappa_{12} = \kappa_3$, versus the full model alternative $H_A : \kappa_{12} \neq \kappa_3$. The analysis we just performed was for the full model, and before proceeding with the definition of the constraint in H_0, the MLEs for H_A must be saved. To do so, click on the pulldown menu in the parameter table (Fig. 4 and choose SAVE LF STATE. A collection of parameter MLEs and constraints constitute a state, i.e. a hypothesis. Name the state "Full Model", and choose SELECT AS ALTERNATIVE from the same pulldown menu.

Now, the constraint for the null hypothesis must be defined, and a new set of MLEs for all independent model parameters must be calculated. To define the constraint, select both transversion/transition ratio parameters (shift-click to select multiple rows), and click on the constraint (2nd) button.

Note, that the parameter table updated to reflect that one of the ratios is no longer independent. Next, we calculate a new set of parameter MLEs, by optimizing the likelihood function anew. Not surprisingly, for $H_0 : \kappa_{12} = \kappa_3 = 0.11$ which is between the independently estimated values.

Save the set of MLEs for H_0 as "Constrained", and then choose SELECT AS NULL. This instructs *HyPhy* to treat "Constrained" as the null hypothesis. With all the components of a hypothesis test in place, choose LRT from the same pulldown menu. *HyPhy* computes the likelihood ratio statistic $2 (\log L_A - \log L_0)$ and a p-value based on the asymptotic χ^2 distribution with (in this case) one degree of freedom:
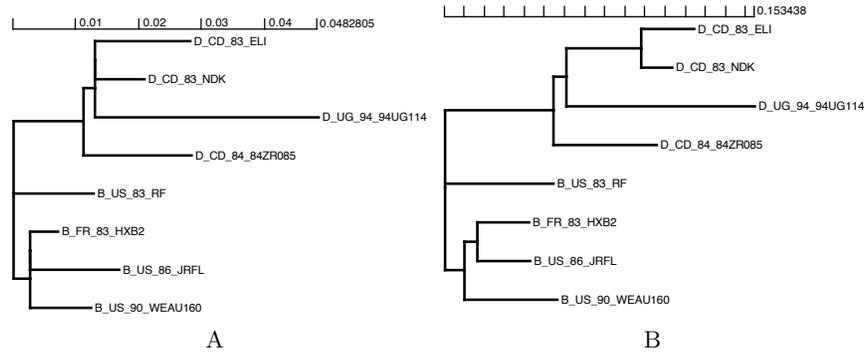
Fig. 14. HIV-1 RT scaled on the expected number of substitutions per site for (A) First and second codon positions, (B) Third codon position

```
Likelihood Ratio Test
     2*LR = 12.5286
     DF = 1
     P-Value = 0.000400774
```

The likelihood ratio test strongly rejects the null hypothesis of equal transversion/transition ratios between partitions.

Parametric Bootstrap.

The χ_1^2 distribution is the *asymptotic* distribution for the LRT statistic, and one would be well advised to realize that it may not always apply directly.

However, one can always verify the results of a χ^2 test by parametric bootstrap (Cox, 1961; Goldman, 1993). *HyPhy* has a very general way of simulating sequence alignments parametrically - it can do so transparently for any likelihood function, using current parameter values. For the purposes of this example, *HyPhy* simulates 1000 8-sequence alignments with 1320 sites each, using the model in the null hypothesis, i.e. constrained ratios. *HyPhy* then fits the models in H_0 and H_A to every simulated dataset, and tabulates the likelihood ratio test statistic. The resulting LRT distribution may then be used for obtaining significance values for the original LRT value and verifying how well the LRT statistic follows the asymptotic χ^2 distribution.

The parametric bootstrap function can be accessed via the same pulldown in the parameter table window. Enter the number of data replicates to be simulated and choose whether or not *HyPhy* should save data and parameter estimates for every replicate. A 1000 replicates should take 20-30 minutes on a desktop. *HyPhy* opens a summary bootstrap table, and adds simulated LRT statistic values as they become available, as well as keeping tabs on the current p-value. Replicates with larger values of the LRT than the original test are highlighted in bold. After bootstrapping has finished, you may open

a histogram or cumulative distribution function plot for the LRT statistic, as shown in Figure 15. Your simulation results will differ from run to run, but you should still obtain a p-value very close to the asymptotic χ^2 p-value and a LRT histogram mirroring the shape of a χ^2 with a single degree of freedom.
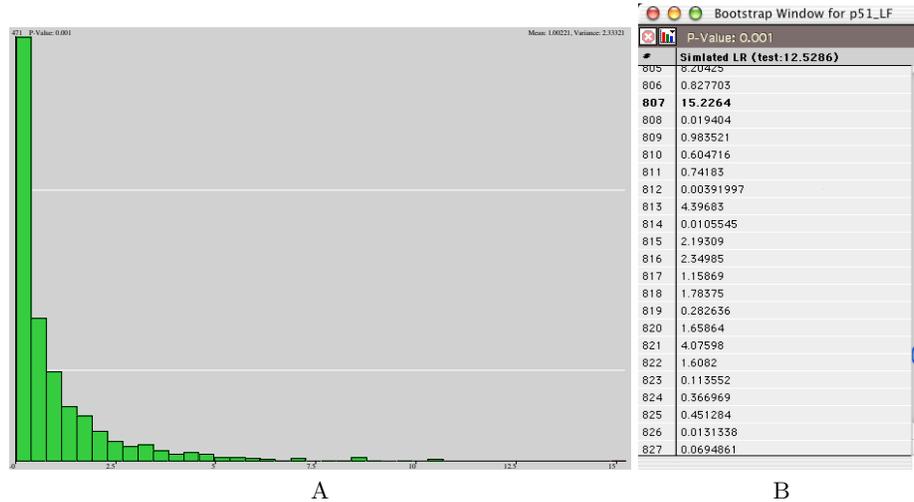
Fig. 15. (A) Simulated density for the likelihood ratio statistic (B) Bootstrapping window example.

Relative ratio test.

It is clear from Fig. 14 that the tree on the first and second positions $\mathcal{T}_{12}$ has much shorter branch lengths than the tree for the third position $\mathcal{T}_3$, which is to be expected. However, apart from a few internal branches, the overall shapes of the trees remain somewhat similar, suggesting that perhaps the only fundamental difference between nucleotide level substitution processes is the amount of change for the entire tree, while relative branch lengths $E_{st}(b_i)$ are the same among both trees. Mathematically, this constraint can be expressed as

$$E_s(b_i|\mathcal{T}_{12}) = R_R E_s(b_i|\mathcal{T}_3), \text{ for all branches } b_i,$$

where the parameter R_R is the relative ratio. As we have seen earlier, branch lengths for HKY85 are linear function of the branch length parameter t, thus it is sufficient to constrain t parameters to be proportional.

HyPhy has a built-in tool for easy specification of relative ratio constraints (Yang, 1996; Muse and Gaut, 1997) on trees or subtrees. To carry

out the relative ratio test, select two trees (or two branches that root the sub-trees, see below), and click on the relative ratio button (2nd from the right in the toolbar) in the parameter table. Name the ratio parameter, and then reoptimize the parameters. Use the technique from the previous example to save the full and constrained models and to carry out the likelihood ratio test, using either the asymptotic distribution or the parametric bootstrap. The result from the chi-squared distribution is:

```
Likelihood Ratio Test
 2*LR = 24.0092
 DF = 12
 P-Value = 0.0202825
```

The relative ratio hypothesis can therefore be rejected at the 0.05 level, but not at the 0.01 level. Application of the parametric bootstrap yields a comparable p-value.

Saving a complete analysis.

HyPhy is capable of saving an analysis and every hypothesis in a single file. Invoke File:Save from the *data panel*, and choose the format which includes sequence data in the resulting file dialog. If you later open the saved file by selecting File:Open:Open Batch File, the analysis and all the hypotheses you have defined will be available.

2.4 Codon Models

The natural unit of evolution for stochastic models of protein coding sequences is a codon. By modeling the substitutions on the level of codons, rather than nucleotides, inherently different processes of synonymous and non-synonymous substitutions can be handled adequately. By expanding the state space for the substitution process from 4 nucleotides to 61 non-stop codons in the universal genetic code, the computation cost increases dramatically, both when evaluating transition probability matrices and calculating the likelihood function itself. Modern computers can handle the added burden quite easily, though.

Consider a codon-based extension to the HKY85 model, which is similar to the (Muse and Gaut, 1994) model. We dub it MG94xHKY85. The 61×61 rate matrix for this model, which gives the probability of substituting codon x with codon y in infinitesimal time, is

$$
Q_{x,y}(\alpha, \beta, \kappa) = \begin{cases} \alpha\pi_{n_y} & x \to y \text{ 1-step synonymous transition,} \\ \alpha\kappa\pi_{n_y}, & x \to y \text{ 1-step synonymous transversion,} \\ \beta\pi_{n_y}, & x \to y \text{ 1-step nonsynonymous transition,} \\ \beta\kappa\pi_{n_y}, & x \to y \text{ 1-step nonsynonymous transversion,} \\ 0, & \text{otherwise.} \end{cases} \tag{2}
$$

As before, κ is the transversion/transition ratio. The parameter α denotes the synonymous substitution rate, while β provides the nonsynonymous substitution rate. The ratio of these two values, $\omega = \beta/\alpha$, can be used to measure the amount of selective pressure along a specific branch. The value π_{n_y} is the frequency of the "target nucleotide" for the substitution observed in the appropriate codon position in the data set. For instance, if $x = ATC$ and $y = AGC$, then π_{n_y} would be the frequency of nucleotide G observed at second codon positions in the alignment. The model only allows for one instantaneous nucleotide substitution between codon. For instance, $ATC \rightarrow AGG$ is not allowed to happen by two concurrent nucleotide substitutions, because such events have negligibly small probabilities. However, such changes are allowed via multiple substitutions as evidenced by the fact that all transition probabilities (entries in matrix $\exp[Qt]$) are non-zero for $t > 0$.

The specification of the model is completed by providing the equilibrium frequencies of the 61 codons. For a codon composed of three nucleotides i, j, k:

$$\pi_{ijk} = \frac{\pi_i^1 \pi_j^2 \pi_k^3}{1 - \pi_T^1 \pi_A^2 \pi_A^3 - \pi_T^1 \pi_A^2 \pi_G^3 - \pi_T^1 \pi_G^2 \pi_A^3}, \tag{3}$$

where π_n^k denotes the observed frequency of nucleotide n at codon position k. The normalizing term accounts for absence of stop codons TAA, TAG and TGA from the state space and the model.

Note that his model mixes local (α and β) and global (κ) parameters.

MG94xHKY85 applied to HIV-1 Integrase gene.

Following are the steps needed to apply a codon model to `integrase_BDA.nex`, found in the `Examples` directory of *HyPhy* standard distribution. This data file contains the integrase gene of 6 Ugandan subtype D, 3 Kenyan subtype A and 2 subtype B (Bolivia and Argentina) HIV-1 sequences sampled in 1999. The integrase gene is relatively conserved and is appropriate for between subtype comparisons.

1. Open the data file via FILE:OPEN:OPEN DATA FILE.
2. Select all the data and define a partition - it will be created as a nucleotide partition at first.
3. Switch partition type to "Codon". *HyPhy* will display a partition properties box. Rename the partition to "Integrase", but keep all other default settings.
4. Assign "Integrase_BDA_tree" topology, "MG94xHKY85_3x4" model and "Local" parameters option.
5. Build (LIKELIHOOD:BUILD FUNCTION) the Likelihood Function. Note that 38 local parameters (α and β for each of the 19 branches) and 1 global parameter (transversion/transition ratio) have been created.
6. Optimize (LIKELIHOOD:OPTIMIZE) the likelihood function. It should take a minute or so on a desktop computer. Open two tree displays, and scale

one on synonymous rates and the other on nonsynonymous rates. Notice the radical differences between the trees, both in lengths and shapes, as shown in Fig. 16.
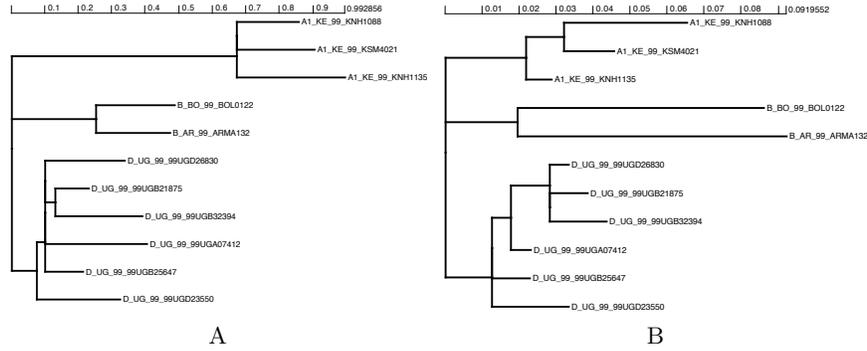

Fig. 16. HIV-1 integrase tree scaled on: (A) Synonymous rates α (B) Nonsynonymous rates β

Molecular Clock Tests.

When reversible models of evolution are used the rate parameters cannot be identified separately from the time parameters, because only their products are estimable. A set of sequences is said to have evolved under a molecular clock if the expected amount of evolution (measured in expected numbers of substitutions) from the most recent common ancestor to each of the descendent sequences is the same. Mathematically, we constrain the length of the paths between each sequence and the most recent common ancestor in the phylogenetic tree to be the same. For the tree in Fig. 17, a molecular clock would be imposed by the following two constraints: $t_2 = t_1$ and $t_3 = t_1 + t_4$. Note that imposing a molecular clock typically requires a rooted tree. Thus, it is desirable to have a separate outgroup sequence (or groups of sequences), which can be used to establish the root of a tree. For instance, in the HIV-1 integrase example (Fig. 16), subtype A sequences form an outgroup to both B and D subtype clades.

For coding sequences, it is often useful to impose molecular clocks on synonymous substitutions only. Synonymous substitutions are assumed to be relatively free of selective constraints, whereas nonsynonymous substitutions will be heavily influenced by purifying and positive selection.

HyPhy provides an easy way to impose molecular clock constraints on a subtree, using some or all model parameters. For MG94xHKY85, it can be shown that the expected number of substitutions per site has the form:
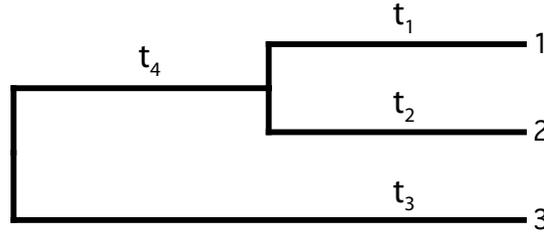
Fig. 17. Example of a molecular clock constraint

$$E_s(t, \alpha, \beta, \kappa, \pi) = t\alpha[f_1(\pi) + \kappa f_2(\pi)] + t\beta[g_1(\pi) + \kappa g_2(\pi)],$$

where f_1, f_2, g_1, g_2 are functions determined by the nucleotide composition of the alignment. The first term in the sum corresponds to the contribution of synonymous substitutions, the second to the contribution of non-synonymous substitutions. Since each is a multiple of the corresponding substitution parameter (α or β), imposing additive constraints on α and β will result in additivity of corresponding expected substitution quantities. Note again that the time parameter t is not estimable alone, and the parameters actually being estimated (and constrained) are $t\alpha$ and $t\beta$.

Thus, three types of molecular clocks may be tested for local codon models: (i) synonymous only, (ii) non-synonymous only and (iii) full (both synonymous and non-synonymous) rates.

Local Clock Tests on HIV-1 Integrase

We now address the following question: which of the three molecular clocks are supported for the D-subtype clade? We assume that the MG94xHKY85 model has been fitted to the data as described above.

1. Save the likelihood function state as "Full Model". Select it to be the alternative hypothesis for our tests.
2. Select the branch that is the most recent common ancestor of the D clade in the tree viewer. Invoke TREE:SHOW PARAMETERS IN TABLE. This action will locate two rows in the parameter table, with the parameters attached to that branch - "Node9". This method is a general way for locating branch specific model parameters in the table quickly - it also works for a multiple branch selection. Highlight one of the two identified rows.
3. Click on the molecular clock button (5th from the left) in the toolbar of the parameter table. A pulldown menu will appear with the parameters available for the molecular clock constraints. Choose to constrain "synRate", for the synonymous rate clock.

4. Optimize the likelihood function, save the new likelihood function state, as "Synonymous Clock", set it to be the null hypothesis. Perform the likelihood ratio test. The test will report the likelihood ratio statistic of 9.52, which yields the p-value of 0.09, using the asymptotic χ^2 with 5 degrees of freedom. This value is reasonably close to rejecting the molecular clock hypothesis, so a bootstrap p-value verification may be desirable. For codon data, bootstrapping is a time consuming process, so you may only choose to do 100 replicates. Our simulation yielded a p-value of 0.14, failing to reject the molecular clock.

5. Select "Full Model" from the pulldown menu in the toolbar of the parameter table, then go back to step 3 and repeat the steps 3 and 4, constraining non-synonymous rates first and then both rates. Likelihood ratio tests fail to reject either of the clocks.

6. Save the analysis from the data panel.

2.5 More general hypothesis testing

Example hypotheses of the previous section are all nested hypotheses, and can be obtained by constraining some of the model parameters in the more general hypothesis to reduce it to a particular case - the null hypothesis. Often, interesting biological questions can not be framed as nested hypotheses. For example, the question of whether a particular phylogeny with certain taxa constrained to be monophyletic is significantly different from the unconstrained phylogeny is a non-nested question. Another example would be determining which of two competing models better explains the data when the models are non-nested. *HyPhy* includes a rather general mechanism for non-nested hypothesis testing based on the parametric bootstrap (Cox, 1961; Goldman, 1993). All one needs to do is to define the competing models (by models, we mean more than just the substitution matrices) *on the same alignment*, and the testing can be accomplished by parametric bootstrapping.

Consider the example data set of the p51 subunit of HIV-1 reverse transcriptase from the previous sections. As an illustration of testing non-nested hypotheses, we will consider whether there is enough evidence to suggest that the JTT model (**?**) describes the data better than the Dayhoff model of amino acid evolution.

First, we must convert a codon alignment found in `p51.nex` into aminoacids.

1. Open the data file `p51.nex`, select all alignment columns and create a nucleotide partition

2. Change the data type of the partition to "Codon" obeying the universal genetic code

3. Select DATA:ADDITIONAL INFO:AMINOACID TRANSLATION. Choose "All" in the ensuing dialog box *HyPhy* will translate all the sequences in all the defined codon partitions, using the genetic code in that partition to amino

acids, create a new data set and open a new data panel displaying all the sequences

4. Let us now save the amino acid alignment to a separate data file. In the newly opened data panel with the amino acid alignment, create a partition with all the alignment sites and, with the partition row selected, click on the "Save Partition To Disk" button. Choose "NEXUS Sequential[Labels]" format in the file save dialog, and save the file as `p51.aa` in the "data" directory of the *HyPhy* distribution

Secondly, we evaluate the likelihood under the null hypothesis H_0: Dayhoff Model

1. Open the amino acid alignment `p51.aa`, select all alignment columns and create a protein partition named "p51"
2. Assign the included p51 tree topology and the "Dayhoff" substitution model to the "p51" partition
3. Build and optimize the likelihood function

The null model has 13 estimable parameters and yields a log-likelihood of -2027.28.

Next, we set up the alternative hypothesis H_A: JTT Model

1. Open the amino acid alignment `p51.aa`, *while the previous analysis is still open*. We need to keep both analyses in memory at the same time. Note how *HyPhy* renamed the new data panel "p512", to avoid a naming conflict with an already open window
2. Assign the tree topology found in the data file and the "Jones" substitution model to the data partition.
3. Build and optimize the likelihood function

The alternative model also has 13 adjustable parameters and yields the log-likelihood of -1981.61.

The JTT model provides a higher likelihood value, but since the models are not nested, we can not simply apply the likelihood ratio test to determine whether the difference is statistically significant. We can, however, use the parametric bootstrap to find a p-value for the test without relying on any asymptotic distributional properties.

1. Switch to either of the data panels, and invoke LIKELIHOOD:GENERAL BOOTSTRAP. *HyPhy* will display a bootstrap set-up window, which is very similar to the window we have seen in nested bootstrap examples.
2. Set the appropriate null and alternative hypotheses, by choosing the name of the data panel ("p51" should be the null and "p512" should refer to the alternative, if you have followed the steps closely).
3. Click on the "Start Bootstrapping" button, select PARAMETRIC BOOTSTRAP from the pulldown, enter 100 for the number of iterates.

4. *HyPhy* will perform the requested number of iterates (it should take five or ten minutes on a desktop), and yield the estimated p-value. In our simulation we obtained a p-value of 0, suggesting that the data are better described by the JTT model.

2.6 Spatial Rate Heterogeneity: Selective Pressure and Functional Constraints.

It is a well appreciated fact that evolutionary rates in sequences vary from site to site. Good substitution models should be able to include such rate variation and provide ways to infer rates at a given site. Consider the MG94xHKY85, modified to allow each site s to have it's own synonymous (α_s) and non-synonymous (β_s) rates. The rate matrix of MG94xHKY85 must be modified as follows

$$Q_{x,y}(\alpha, \beta, \kappa) = \begin{cases} \alpha_s \pi_{n_y} & x \to y \text{ 1-step synonymous transition,} \\ \alpha_s \kappa \pi_{n_y}, & x \to y \text{ 1-step synonymous transversion,} \\ \beta_s \pi_{n_y}, & x \to y \text{ 1-step non-synonymous transition,} \\ \beta_s \kappa \pi_{n_y}, & x \to y \text{ 1-step non-synonymous transversion,} \\ 0, & \text{otherwise.} \end{cases}$$

The most general estimation approach would be to estimate α_s and β_s separately for every site, but that would create too many parameters and result in estimability issues. Another idea, first proposed by Yang (1993); Nielsen and Yang (1998), is to treat site rate parameters (in their work only one parameter is allowed to vary per site), as a random variable, drawn from a certain (prior) distribution family, $F_{\boldsymbol{\eta}}(\alpha_s, \beta_s)$ whose parameters $\boldsymbol{\eta}$ are either given or estimated. The likelihood for an alignment with S sites, tree $\mathcal{T}$, and the vector Θ of model parameter values can be evaluated as:

$$L(\mathcal{D}|\mathcal{T}, \Theta) = \prod_{s=1}^{S} E\left[L(\mathcal{D}_s|\mathcal{T}, \Theta, \alpha_s = a, \beta_s = b)\right].$$

The expectation is computed using the distribution specified by $F_{\boldsymbol{\eta}}(\alpha_s, \beta_s)$. Site likelihoods, conditioned on the values of α_s and β_s may be evaluated using Felsenstein's pruning algorithm (Felsenstein, 1981). Unless $F_{\boldsymbol{\eta}}(\alpha_s, \beta_s)$ specifies a discrete distribution with a small number of classes, the expectation is computationally intractable. However, Yang (1994) introduced the approach of discretizing the continuous distribution of rates to obtain a computationally tractable formulation.

If site s in the alignment is following neutral evolution, then we expect to infer $\beta_s \approx \alpha_s$. For sites subject to functional constraints, non-synonymous mutations are almost certain to be highly deleterious or lethal, leading to purifying selection and $\beta_s < \alpha_s$. If $\beta_s > \alpha_s$, the site s is evolving under positive selective pressure, or undergoing adaptive evolution.

In contrast to existing methods that simply have sites varying according to their rates, *HyPhy* allows the user to identify multiple parameters that are free to vary over sites. In the following example we allow both synonymous and nonsynonymous rates to be variable across sites, leading to the possibility, for instance, that a particular site might have a fast nonsynonymous rate but a slow synonymous rate. We will consider the case of MG94xHKY85 applied to a codon data set with α_s and β_s sampled independently from two separate distributions. Because only products of evolutionary rates and times can be estimated, we let the **mean** of the distribution of α_s to be one. Widely used models of Nielsen and Yang (1998) assume that $\alpha_s = 1$ for every site s, thus our approach is a natural extension. For our example, we choose to sample α_s from a gamma distribution $\gamma(\alpha_s; \mu_\alpha)$ with mean 1 and shape parameter μ_α discretized into 4 rate classes by the method of Yang (1993). The nonsynonymous rates, β_s, are assumed to come from a mixture of a general γ distribution and a point mass at 0 to allow for invariable sites (REF). The density of this distribution is

$$\beta_s \sim R\left[P_I \delta_0(\beta_s) + (1 - P_I)\gamma(\beta_s; \mu_\beta)\right], \qquad (4)$$

where $0 \leq P_I \leq 1$ is the proportion of (non-synonymous) invariable sites, R is the mean of the distribution, and is the ratio of the means of the non-synonymous and synonymous distribution (similar to dN/dS). The density of the unit mean gamma distribution with shape parameter μ_β is $\gamma(\beta_s, \mu_\beta)$. The gamma portion of the distribution is discretized into 3 rates, and, with the invariant rate class, the total number of non-synonymous rate categories is 4.

To perform a maximum likelihood fit of this model in *HyPhy* we follow the following steps.

1. Open the data file `p51.nex`.
2. Select all data, create a single partition, change its data type to codon and name to RT_Gene.
3. Assign the tree, the model "MG94xHKY85x3_4x2_Rates" with "Rate Het." model parameters and 4 (per parameter) rate categories. The model we selected implements the extension to the MGx94xHKY85 model we have just discussed.
4. Build the likelihood function and optimize it. Depending on the speed of your computer this make take up to an hour.

Parameter estimates returned by the analysis are as follows:

```
RT_Gene_Shape_alpha    = 1.637
RT_Gene_Shape_beta_Inv = 0.708
RT_Gene_Shape_beta     = 1.174
RT_Gene_Shared_DNDS    = 0.527
```

HyPhy can also display the discretized distributions along with their continuous originals. This feature can be accessed via the pulldown in category

variable rows in the parameter table (Fig. 19). Density plots show the continuos density curve, the table of discrete rate classes and their visual representations. Dotted lines depict the bounds for the intervals that each rate class - a solid vertical line - represent.
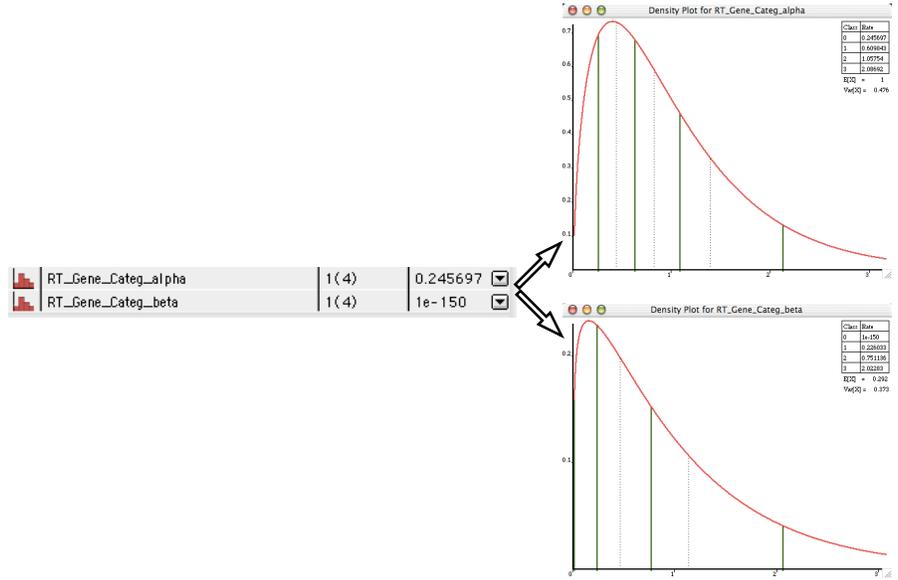

Fig. 18. Synonymous and non-synonymous distributions for the analysis of HIV-1 RT gene

It is immediately clear that synonymous rates are not constant across sites. Indeed the coefficient of variation for α_s, which is equal to $1/\sqrt{\mu}_{\alpha}$, is inferred to be 0.61, whereas we would expect a much smaller value were the synonymous rates indeed constant among sites.

The really interesting task is, however, to determine which sites are conserved, and which are evolving under selective pressure. An approach proposed by Yang et al. (2000) is to employ the empirical Bayes technique. To do so, we fix all model parameter estimates (more on the validity of that later), and compute the posterior probability $p_{i,j}^s$ of observing rates a_i, b_j at site s. *HyPhy* can compute the conditional likelihoods of every site (choose LIKELIHOOD:CATEGORIES PROCESSOR from the parameter table, see Fig. 19) given that the rates come from the category i, j:

$$l_{i,j}^s = L(\mathcal{D}_s | \mathcal{T}, \Theta, \alpha_s = a_i, \beta_s = b_j).$$

Thus, by Bayes rule

$$p_{i,j}^s = Pr\{\alpha_s = a_i, \beta_s = b_j | \mathcal{D}_s\} = \frac{l_{i,j}^s Pr\{\alpha_s = a_i, \beta_s = b_j\}}{\sum_{m,n} l_{m,n}^s}.$$

Consider two events at site s: positive selection, $PS_s = \{\alpha_s < \beta_s\}$, and negative or purifying selection $NS_s = \{\alpha_s > \beta_s\}$. For any event one can define the Bayes factor, which is simply the ratio of posterior odds of an event and its prior odds. If the Bayes factor of an event is significantly greater than 1, then the data support the event.

Fig. 19. Conditional site likelihoods module of *HyPhy*

Having opened the categories processor (Fig. 20), we proceed to perform the posterior Bayes analysis as follows

1. Create a new random variable $\beta_s - \alpha_s$. To do so, invoke CATEGORIES:DEFINE NEW VARIABLE and enter the expression (try to use the pulldown menu for quick access to category variables) $0.527RT\_Gene\_Categ\_beta - RT\_Gene\_Categ\_alpha$. We multiply by value of $R(= 0.527)$, because in *HyPhy* parameterization RT_Gene_Categ_beta refers to the expression inside the brackets in (4) - you can check that by opening the model display in "Object Inspector".
2. Expand the view for the new difference variable by clicking on the arrow next to it, and choose (shift-click or drag select) the event for positive selection: all positive values of the difference variable.
3. Perform empirical Bayes analysis by selecting CATEGORIES:EVENT POSTERIORS. In the window which opens, select type of "Bar Chart" and Y of "Bayes Factor". This display gives an easy overview of sites with large support for positive selection - for example with Bayes factor over 20.
4. Instruct *HyPhy* to find all the sites with the Bayes factor over 20. For this, select the Bayes factor column (click on the column header), and choose CHART:DATA PROCESSING:SELECT CELLS BY VALUE. *HyPhy* will prompt for the selection criterion: type in "cell_value>20". The results

are shown in Fig. 21 According to this criterion, there are 12 positively selected codons $35, 178, 179, 200, 211, 243, 272, 282, 329, 376, 377$ and 403.
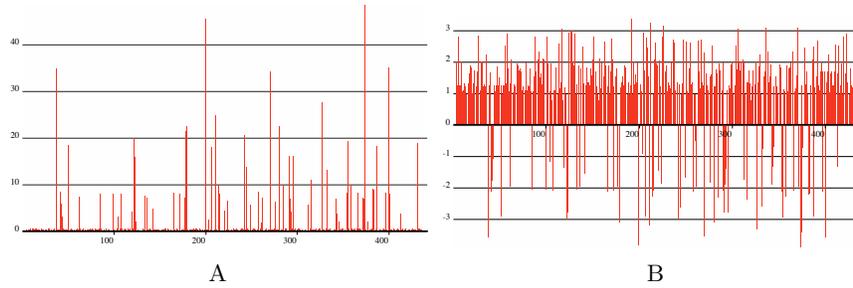

Fig. 20. (A) Bayes factor for the event of positive selection at a site. (B) Log of the Bayes factor for the event of negative selection at a site
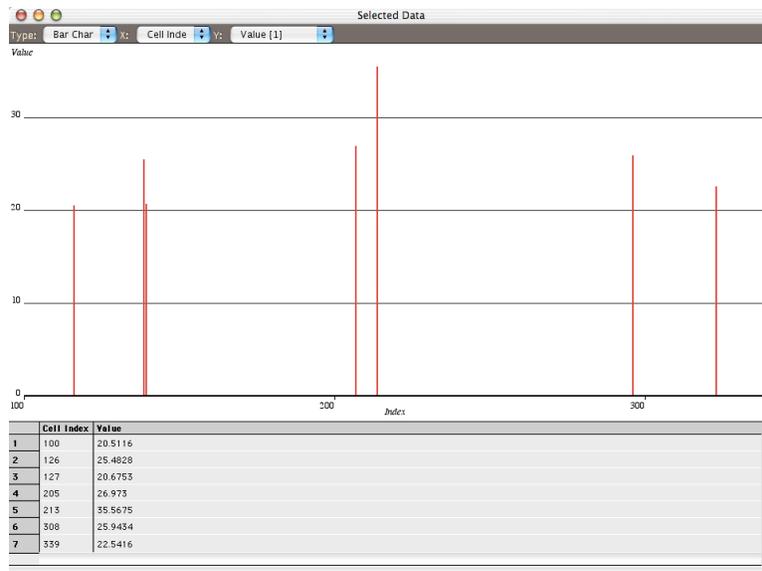

Fig. 21. Sites found to be under positive selection and supporting Bayes factors in HIV-1 RT gene

The weakness of empirical Bayes.

It has been argued that maximum likelihood empirical Bayes methods for detecting rates at sites may yield many false positives (ref). Alternatively, if very few sites in the alignment are under selective pressure, it is possible that the prior (and hence, posterior) distributions will place zero probability on *any* site being positively selected, exhibiting low power. The main shortcoming of empirical Bayes approaches is that parameter estimates are treated as correct values, and the uncertainties in estimation procedures are discounted altogether. If one were to compute 95% confidence intervals based on likelihood profiles with *HyPhy* one would discover that

$$\mu_\alpha \in (0.759, 10.175), \mu_\beta \in (0.589, 3.467)$$

$$P_I \in (0.642, 0.779), R \in (0.405, 0.688)$$

That is quite a range of variation, and change in any of those parameters would affect the conclusions of empirical Bayes methods. For instance, the most conservative (in terms of limiting false positives but also reducing power) estimates can be obtained by choosing maximum possible μ_α, μ_β and P_I and the minimum possible R. For this choice of parameters, the maximum Bayes factor at any site is a mere 17.914 and by our old criteria no sites are found to be under selective pressure. One should always realize that uncertainties in parameter estimates can greatly influence the conclusions of an empirical Bayes analysis, and it helps to compare various scenarios to assess inference reliability.

Further pointers.

HyPhy can run analyses like the one just described in parallel on distributed systems using Message Passing Interface (MPI). For instance, if 16 processors are available, computations of $l_{i,j}^s$ for each of the 16 possible rate class combinations (i, j) are placed automatically on separate processors, achieving speeds similar to those of a single rate analysis on a single CPU system, making analyses with hundreds of sequences in an alignment feasible. Refer to http://www.hyphy.org for more details.

   HyPhy also implements an ever-expanding collection of rapid positive/negative selection analyses for data exploration, loosely based on the counting method of Suzuki and Gojobori (1999), as well as site-by-site (and/or lineage specific) likelihood ratio testing. It is accessible via standard analyses and more details can be found in *HyPhy* documentation.

2.7 Mixed data analyses

As more and more organisms are being fully sequenced, methods and tools for analyzing multigene sequence alignments and, ultimately, genome-wide data

sets, are becoming increasingly relevant. In the small example that follows we will show how one can use *HyPhy* to begin to address such analytic needs.

We consider a sequence alignment of five sequences, each consisting of two introns and an exon, which can be found in `intronexon.nex` within the `Examples` directory.

We must partition the data into introns and exons. It is appropriate to consider two partitions: coding and non-coding. For more complex data sets one can easily define a separate partition for every gene, etc. First, create a partition which includes all of the data (EDIT:SELECT ALL, followed by DATA:SELECTION->PARTITION.

The exon spans nucleotide positions 90 through 275. One of the ways to create the partition for the exon is to locate alignment column 90 in the data panel and select it, then scroll to column 275 and Shift-click on it (this selects the whole range). Note that the status line of the data panel was updated to reflect your current selection. Make sure it shows "Current Selection: 90-275". An alternative approach is to start at column 90 and then click-drag to column 275. Yet another possibility is to choose DATA:INPUT PARTITION and enter $89 - 274$ (0-based indices).

Once the range has been selected, invoke DATA:SELECTION->PARTITION. We now have two partitions, overlapping over columns $90 - 275$, as shown in the Navigation Bar. The final step is to subtract the partitions to create a new partition for the introns. To do this, we select both partition rows in the data panel table (Shift-click selects multiple rows). Next, click on the "Subtract 2 Overlapping Partitions" button. Select the appropriate operation in the resulting pulldown. We have specified two non-overlapping partitions. Note that the intron partition is not contiguous. Rename the intron partition to "Introns" and the exon partition to "Exon". One could achieve this same partioning scheme by defining three partitions: 1-89, 90-275, 276-552, and joining the first one and the third one.

There is one more filtering step left to do before we can begin analyzing the data. As often happens with smaller subsets extracted from larger alignments, there are several alignment columns consisting entirely of deletions. Such columns do not contribute informational content to likelihood analyses, and should be removed. Select the "Exon" row in the partition table, click on the "Data Operations" button and select SITES WITH ALL DELETIONS. *HyPhy* will locate all such sites *inside the selected partition only* and select them. Create a partition with those sites, subtract it from the exon partition as discussed above, and delete the partition with uninformative sites (select its row and click on the "Delete Partition" button).

Since introns are not subject to functional constraints of coding sequences it makes sense to model their evolution with a nucleotide model (HKY85 with global options). For the exon partition a codon model is appropriate. Change the data type of "Exon" to "Codon", and apply the "MG94xHKY85x3_4" model with local options. The end result should look like Fig. 22 (A).
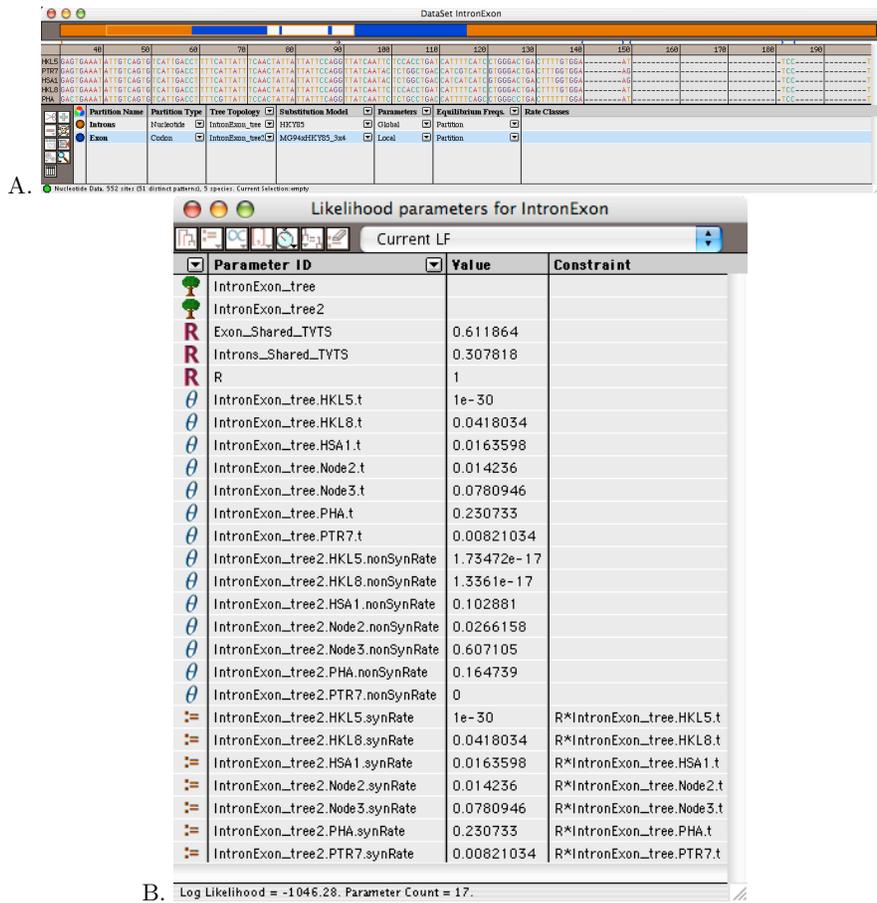
Fig. 22. Exon-Intron mixed analysis. (A) Data panel setup and (B) Parameter table with the relative ratio constraint

Next, build and optimize the likelihood function and open the parameter table. Our analysis includes two trees with the same topology (one for introns and the other for exons). The model for the intron tree has a single parameter per branch (branch length) and a shared transversion/transition ratio ($Exon\_Shared\_TVTS = 0.308$), whereas the model for the exon tree has two parameters per branch, synonymous and nonsynonymous rates and a shared transversion/transition ratio ($Introns\_Shared\_TVTS = 0.612$). (Note that we could use previously discussed methods for testing hypotheses to decide if the two transversion/transition ratios are different.)

One of the common assumptions made for analyses of molecular sequence data is that differences between coding and non-coding sequences can be explained by functional constraints and selective pressures on coding sequences,

namely by changes in rates of nonsynonymous substitutions. In other words, synonymous substitutions in coding regions and nucleotide substitutions in neighboring non-coding stretches should have comparable relative rates. This assumption may be violated if mutation rates vary along the sequence or if there is selection operating in non-coding regions. We will now test this hypothesis of a relative ratio between the introns and the exon in our example data sets. In other words, we want to see if the exon tree scaled by synonymous rates has the same patter of relative branch lengths as the intron tree. Mathematically, the set of relative ratio constraints constraints is:

$$exonTree.branch.synRate = R \times intronTree.branch.t,$$

where R is the (global) relative ratio, and the constraint is applied to every branch. For a small tree like ours, it is easy to use the proportional constraint tool in the parameter table interface module to define the constraints one at a time; however this could become very tedious for larger trees. Luckily, *HyPhy* includes a command designed to traverse given trees and apply the same constraint to every branch. As you will learn from the next section, at the core *HyPhy* is a programming language (HBL), and all of the interface features we have discussed previously use HBL behind the scenes. If the interface does not include a built-in tool for a specific constraint, the user may tap directly into HBL to carry out the task at hand. We will do just that for our example.

Open the parameter table for the Intron-Exon analysis we have just set up (make sure none of the parameters are constrained). Invoke LIKELIHOOD: ENTER COMMAND. *HyPhy* will take any input from the dialog box that appears, parse the commands contained therein and execute them. We need to invoke *ReplicateConstraint*, which is a powerful, but somewhat complicated command. If we were to impose the constraints by hand at every branch we would begin with

$$IntronExon\_tree2.HKL5.synRate = R \times IntronExon\_tree.HKL5.t,$$

and repeat applying the same constraint replacing "HKL5" with other branches in the tree. A single call using *ReplicateConstraint* will accomplish the same task:

```
global R = 1;
ReplicateConstraint(''this1.?.synRate:=R*this2.?.t'',
        IntronExon_tree2,IntronExon_tree);
```

The expression in quotation marks is the constraint *template*. "this1" is replaced with the first argument (IntronExon_tree2), "this2", with the second and so on. The '?' is a wildcard meaning *match any branch name*. *ReplicateConstraint* is a very handy command to know, and we refer the reader to examples contained in the *HyPhy* distribution. The "global R=1" command is needed to declare R as a shared parameter and initialize it (further details are provided in the next section). Enter the above commands into the dialog

box, and, if all went well, the parameter table will update and should look like Fig. 22 (B). Optimize the likelihood function, define the null hypothesis, and perform the likelihood ratio test. The asymptotic p-value of the test is 0.023, rejecting the hypothesis of relative ratio. Since our data set is rather small, we would be well advised to verify this result with parametric bootstrap. We obtained bootstrap p-value of 0.003 with 1000 replicates.

3 The *HyPhy* Batch Language

Underlying the *HyPhy* graphical user interface is a powerful interpreted programming language, HBL (*HyPhy* Batch Language). The authors originally developed HBL as a research tool to allow rapid development of molecular evolutionary analyses. The addition of the graphical interface is a more recent development, and provides access to many common types of analyses. However, the underlying programming language is considerably more powerful and flexible (albeit with a steeper learning curve). The goal of this section is to provide readers with a basic understanding of the fundamentals of HBL programming and an appreciation of the power of the language. In doing so, we shall make use of a series of *HyPhy* batch files, which are available for download at www.hyphy.org/pubs/HyphyBookChapter.tgz. Complete documentation of the batch language is available in the Batch Language Command Reference at www.hyphy.org, and can also be accessed via the built-in command reference in the *HyPhy* console.

3.1 Fundamental Batch File Elements: *basics.bf*

The basic task shared by most *HyPhy* batch files is the optimization of a likelihood function for a given alignment/model/phylogeny combination. Therefore, most every batch file will contain the following elementary components:

1. Input alignment data
2. Describe an evolutionary model of sequence change
3. Input or describe a phylogenetic tree
4. Define a likelihood function based on the alignment, phylogeny, and model
5. Maximize the likelihood function
6. Print the results to the screen and/or an output file.

The simple batch file *basics.bf*, reproduced in its entirety below, illustrates the HBL code necessary to fit the F81 model of sequence evolution to an alignment of four sequences.

```
DataSet myData = ReadDataFile ("data/four.seq");
DataSetFilter myFilter = CreateFilter (myData,1);
HarvestFrequencies (obsFreqs, myFilter, 1, 1, 1);
F81RateMatrix =
```

```
                {{* ,mu,mu,mu}
                 {mu,* ,mu,mu}
                 {mu,mu,* ,mu}
                 {mu,mu,mu,* }};
   Model F81 = (F81RateMatrix, obsFreqs);
   Tree myTree = ((a,b),c,d);
   LikelihoodFunction theLikFun = (myFilter, myTree);
   Optimize (MLEs, theLikFun);
   fprintf  (stdout, theLikFun);
```

Let us now explain how these nine statements accomplish the six key components enumerated above.

Input alignment data.

The task of preparing data for analysis in *HyPhy* consists of two steps. First, the data must simply be read from a data file. After the data is read, it must be "filtered". The process of filtering involves selecting the precise taxa and alignment positions to be analyzed, and identifying the "type" of the data (e.g. nucleotide, codon, dinucleotide).

```
   DataSet myData = ReadDataFile ("data/four.seq");
   DataSetFilter myFilter = CreateFilter (myData,1);
```

The first statement simply reads a sequence alignment into memory and names it *myData*. The HBL function automatically detects the sequence type (DNA) and the input format, then saves the data into a data structure of type *DataSet*, a predefined HBL data type. The second statement is the simplest version of the *CreateFilter* function. In this case, the function takes the alignment stored in *myData* and by default includes all of it in a structure named *myFilter*. The argument "1" indicates that the data should be treated as simple nucleotide data. Had we wanted the data to be interpreted as codons, the argument "3" would have been used instead. The *CreateFilter* command is quite powerful, and we will illustrate the use of some of its optional arguments in later examples. Multiple data filters may be created from the same data set.

Describe an evolutionary model of sequence change.

The next task in our simple analysis is the definition of a model of sequence change. One of the unique strengths of *HyPhy* is its ability to implement any special case of a general time reversible model (and, more generally, any continuous time Markov chain model, not necessarily time reversible), regardless of the dimensions of the character set. We rely on the fact that any special case of the general reversible model can be written in a form where entries in the substitution matrix are products of substitution parameters and character frequencies. Thus, we have adopted a convention of describing time-reversible

models with two elements: a matrix consisting of substitution rate parameters, and a vector of equilibrium character frequencies.

```
F81RateMatrix =
                {{* ,mu,mu,mu}
                 {mu,* ,mu,mu}
                 {mu,mu,* ,mu}
                 {mu,mu,mu,* }};
HarvestFrequencies (obsFreqs, myFilter, 1, 1, 1);
Model F81 = (F81RateMatrix, obsFreqs);
```

In our present example, the substituition parameter matrix of the F81 model is defined and named in an obvious fashion (the *HyPhy* matrix placeholder * is defined as "the negative sum of all non-diagonal entries on the row"). Next, the built-in function *HarvestFrequencies* tabulates the frequencies in *myFilter* and stores them in the newly created vector *obsFreqs*. The functions of the numerical arguments can be found in the Batch Language Command Reference. Finally, the matrix and frequencies are combined to form a valid substitution model using the *Model* statement.

For the F81 model, the instantaneous rate matrix is traditionally denoted

$$
\begin{array}{cccc}
 & A & C & G & T \\
\begin{array}{c} A \\ C \\ G \\ T \end{array} &
\left(\begin{array}{cccc}
-\mu(1-\pi_A) & \mu\pi_C & \mu\pi_G & \mu\pi_T \\
\mu\pi_A & -\mu(1-\pi_C) & \mu\pi_G & \mu\pi_T \\
\mu\pi_A & \mu\pi_C & -\mu(1-\pi_G) & \mu\pi_T \\
\mu\pi_A & \mu\pi_C & \mu\pi_G & -\mu(1-\pi_T)
\end{array}\right)
\end{array}
$$

Observe the similarity between this matrix and the *HyPhy* syntax. By default the *Model* statement multiplies each element of the rate matrix by the equilibrium frequency of an appropriate character, hence *HyPhy* declaration of F81 does not include the multiplication by elements of π. This behavior can be overridden by passing a third argument of 0 to the model statement (that is done, for example for the MG94 codon model).

Input or describe a phylogenetic tree.

HyPhy uses standard (Newick) tree definitions. Thus, the statement

```
Tree myTree = ((a,b),c,d);
```

defines a tree named *myTree* with four OTUs, or taxa, named a, b, c, and d, corresponding to the names in the *HyPhy* datafile. *HyPhy* will accept either rooted or unrooted trees; however, for most purposes rooted trees are automatically unrooted by *HyPhy* because likelihood values for unrooted trees are the same as those for rooted trees.

The *Tree* data structure is much more complex than simply describing a tree topology. The *Tree* variable includes both topology information, as well as evolutionary model information. The default behavior of a *Tree* statement is

to attach the most recently defined *Model* to all branches in the tree. Thus, it is often *critical* that the *Model* statement appears before the *Tree* statement. We will discuss more advanced uses of the *Tree* statement later.

Define a likelihood function based on the alignment, phylogeny, and model.

The likelihood function for phylogenetic trees depends on the dataset, tree topology, and the substitution model (and its parameters). To define a likelihood function, we use a statement like

```
LikelihoodFunction theLikFun = (myFilter,myTree);
```

We name the likelihood function *theLikFun*, and it uses the data in *myFilter* along with the tree topology and substitution model stored in in *myTree*. Recall that the *Tree* structure *myTree* inherited the *Model F81* by default.

Maximize the likelihood function.

Asking *HyPhy* to maximize the likelihood function is simple. The statement

```
Optimize (MLEs, theLikFun);
```

finds maximum likelihood estimates of all independent parameters and stores the results in the matrix named *MLEs*.

Print the results to the screen and/or an output file.

The simplest way to display the results of a likelihood maximization step is simply to print the likelihood function:

```
fprintf(stdout,theLikFun);
```

This C-like command prints the structure *theLikFun* to the default output device *stdout* (stdout is typically the screen). The results of this statement are the following:

```
Log Likelihood = -616.592813234418;
Tree myTree=((a:0.0136035,b:0.0613344)Node1:0.0126329,
c:0.070388,d:0.0512889);
```

When asked to print a likelihood function, *HyPhy* first reports the value of the log-likelihood. It follows with a modified version of the Newick tree description as shown in the output above. Each of the branches in the unrooted phylogeny has an associated branch length, measured in units of expected number of nucleotide substitutions per site. Those values appear after the colon following the label for each branch. For example, the estimated branch length leading to the tip "b" is 0.0613344. Note that the internal node in the tree has been automatically named "Node1" by *HyPhy*, and its associated branch length is 0.0126329. Values of the estimated substitution parameters or base frequencies could be displayed by printing *obsFreqs*. *HyPhy* also allows for more detailed user control of printed output, using a C-like fprintf syntax. The later examples will illustrate this functionality.

3.2 A Tour of Batch Files

Defining Substitution Models

Simple Nucleotide Models: modeldefs.bf.

One of the primary objectives of *HyPhy* is to free users from relying on the substitution models chosen by authors of software. While a relatively small set of model choices may be sufficient for performing phylogenetic analyses, having only a few potential models is often limiting for studies of substitution rates and patterns. To define a model in *HyPhy*, one needs only to describe the elements in a substitution rate matrix. If the characters being studied have n states, the rate matrix is $n \times n$. For example, nucleotide models are 4×4; models of amino acid change are 20×20; codon-based models might be 64×64. *HyPhy* can work properly with any member of the class of general time reversible models, regardless of the number of character states. Instantaneous rate matrices in this class of models satisfy the condition $\pi_i Q_{ij} = \pi_j Q_{ji}$, where π_i is the equilibrium frequency of character i (for nucleotide data,) and Q_{ij} is the ij^{th} entry in the instantaneous rate matrix. *HyPhy* comes with many predefined rate matrices for commonly used substitution models. You can find examples in the *Examples* and *TemplateBatchFiles* directories of the *HyPhy* distribution.

To illustrate the basics of model definition, we discuss the batch file *modeldefs.bf*:

```
SetDialogPrompt("Select a nucleotide data file:");
DataSet myData  = ReadDataFile(PROMPT_FOR_FILE);
DataSetFilter myFilter = CreateFilter(myData,1);
HarvestFrequencies(obsFreqs,myFilter,1,1,1);
F81RateMatrix  = {{*,m,m,m}{m,*,m,m}{m,m,*,m}{m,m,m,*}};
Model F81 = (F81RateMatrix, obsFreqs);
Tree myTree = ((a,b),c,d);
fprintf(stdout,"\n\n F81 Analysis \n\n");
LikelihoodFunction theLikFun = (myFilter, myTree);
Optimize(results,theLikFun);
fprintf(stdout,theLikFun);

fprintf(stdout,"\n\n HKY85 Analysis  \n\n");
HKY85RateMatrix = {{*,b,a,b}{b,*,b,a}{a,b,*,b}{b,a,b,*}};
Model HKY85 = (HKY85RateMatrix, obsFreqs);
Tree myTree = ((a,b),c,d);
LikelihoodFunction theLikFun = (myFilter, myTree);
Optimize(results,theLikFun);
fprintf(stdout,theLikFun);

fprintf(stdout,"\n\n Repeat F81 Analysis  \n\n");
```

```
UseModel(F81);
Tree myTree = ((a,b),c,d);
LikelihoodFunction theLikFun = (myFilter, myTree);
Optimize(results,theLikFun);
fprintf(stdout,theLikFun);
```

This batch file illustrates two new concepts. First, and most importantly, the lines

```
HKY85RateMatrix = {{*,b,a,b}{b,*,b,a}{a,b,*,b}{b,a,b,*}};
Model HKY85 = (HKY85RateMatrix, obsFreqs);
```

illustrate the definition of a new substitution matrix. In this case, we have defined the model of Hasegawa et al. (1985) and named the model HKY85. Those familiar with the HKY85 model, will probably recognize the form of the matrix: transitions occur with rate a and transversions occur with rate b, with each of those substitution parameters multiplied by the appropriate nucleotide frequency to provide the final instantaneous rates. The second important point to note is that we must associate the model with a tree before we can do anything useful. In this case, we simply redefined the old tree to use the HKY85 model instead of the F81 model (Recall that a tree consists of both the topology and the substitution matrices attached to its branches). When the statement `Tree myTree = ((a,b),c,d);` is issued, the variable *myTree* is assigned the topology ((a,b),c,d) and the branches are assigned the HKY85 substitution model, which was the most recently defined *Model*. If we wanted to preserve the original variable *myTree*, we could simply have defined a new *Tree* structure using a command like `Tree myNextTree = ((a,b),c,d);`

Finally, for completeness, we created a new *Tree* and assigned it the F81 model and reproduced the original F81 analysis. Those final steps illustrate how predefined *Models* can be assigned to *Trees* using the *UseModel* command.

Note also, the use of

```
SetDialogPrompt("Select a nucleotide data file:");
DataSet myData  = ReadDataFile(PROMPT_FOR_FILE);
```

to allow the user to locate the sequence file interactively, instead of hard-coding it into the batchfile.

More Nucleotide Models: models.bf.

One of the most general models of nucleotide substitution is the general time reversible model (REV). The instantaneous rate matrix for the REV model is

$$
Q_{\mathrm{REV}} = \begin{array}{c} \\ \mathtt{A} \\ \mathtt{C} \\ \mathtt{G} \\ \mathtt{T} \end{array}
\begin{array}{cccc}
\mathtt{A} & \mathtt{C} & \mathtt{G} & \mathtt{T} \\
\left(\begin{array}{cccc}
\star & \theta_0 \pi_C & \theta_1 \pi_G & \theta_2 \pi_T \\
\theta_0 \pi_A & \star & \theta_3 \pi_G & \theta_4 \pi_T \\
\theta_1 \pi_A & \theta_3 \pi_C & \star & \theta_5 \pi_T \\
\theta_2 \pi_A & \theta_4 \pi_C & \theta_5 \pi_G & \star
\end{array} \right)
\end{array}
$$

It is simple to implement this model in *HyPhy*. The statements

```
REVRateMatrix  = {{*,a,b,c}{a,*,d,e}{b,d,*,f}{c,e,f,*}};
Model REV = (REVRateMatrix, obsFreq);
```

do the job.

To illustrate these notions in a more usefule context, consider the batchfile *models.bf*. In that batchfile, models named F81, HKY85, REV, JC69, and K2P are defined, and each is fit to the same dataset and tree topology. *models.bf* also demonstrates a few useful *HyPhy* features. First, notice the definition of a vector of frequencies for use by the equal frequency models:

```
equalFreqs = {{0.25}{0.25}{0.25}{0.25}};
```

In a similar manner, we define the string constant *myTopology*:

```
myTopology = "((a,b),c,d)";
```

By changing the topology in the definition of *myTopology*, the entire analysis can be repeated using the new topology. This single step is faster than updating the topology for every *Tree* statement, and is particularly useful for topologies with many taxa. Finally, note the reuse of the three substitution matrices and the two frequency vectors. The original matrix definitions are used as templates by the *Model* statements.

Global vesus local parameters: *localglobal.bf*

Because the primary goal of *HyPhy* is to provide flexible modeling of the nucleotide substitution process, *HyPhy* includes a more general parameterization scheme than most phylogeny estimation programs. Perhaps the most important difference for the user to recognize is the distinction between *local* and *global* parameters. In the simplest form, a local parameter is one that is specific for a single branch on a tree. In contrast, a global parameter is shared by all branches. To illustrate, consider the output generated by the batch file *localglobal.bf* when run using (four.seq):

```
 Original (Local) HKY85 Analysis

Log Likelihood = -608.201788537279;
Tree myTree=((a:0.0143364,b:0.061677)Node1:0.0108616,
c:0.0716517,d:0.0526854);

 Global HKY85 Analysis

Log Likelihood = -608.703204177757;
Shared Parameters:
S=3.08185
```

```
Tree myTree=((a:0.0130548,b:0.0618834)Node1:0.0126785,
c:0.0717394,d:0.052028);
```

In *localglobal.bf* we have moved beyond the default settings of *HyPhy*, and the details of the batch file will be discussed below. For now, concentrate on the results. *localglobal.bf* performs two analyses of the data in *four.seq*, each using the HKY85 model of sequence evolution. The first, labeled "Original (Local) HKY85 Analysis", is the same analysis that was performed in the previous example (*models.bf*). In this analysis, each branch in the tree was allowed to have its own transition/transversion ratio.

The second analysis performed in *localglobal.bf* is an example of a *global* analysis. In contrast to the previous analysis, the "Global HKY85 Analysis" invokes a *global* transition-transversion ratio, S. In other words, all branches share the same value of S. The estimated global value of S (3.08185) is shown under the heading of `Shared Parameters`.

The local and global analyses use different numbers of parameters. The local analysis uses a transition and transversion rate for each of the 5 branches, along with 3 base frequencies, for a total of 13 parameters. The global analysis includes a transversion rate for each branch, 3 base frequencies, and a single transition-transversion ratio, for a total of 9 parameters. The global analysis is a special case of the local analysis; therefore, the log-likelihood value for the global analysis (-608.703) is lower than that of the local anaysis (-608.202). The fact that the addition of 4 parameters results in such a small difference in model fit suggests that the data harbor little support for the hypothesis that the transition-transversion rate varies among these lineages.

The code for *localglobal.bf* is the following:

```
SetDialogPrompt ("Please specify a nuleotide data file:");

DataSet myData  = ReadDataFile(PROMPT_FOR_FILE);
DataSetFilter myFilter = CreateFilter(myData,1);
HarvestFrequencies(obsFreqs,myFilter,1,1,1);

fprintf(stdout,"\n\n Original (Local) HKY85 Analysis  \n\n");
HKY85RateMatrix = {{*,b,a,b}{b,*,b,a}{a,b,*,b}{b,a,b,*}};
Model HKY85 = (HKY85RateMatrix, obsFreqs);
Tree myTree = ((a,b),c,d);
LikelihoodFunction theLikFun = (myFilter, myTree);
Optimize(results,theLikFun);
fprintf(stdout,theLikFun);

fprintf(stdout,"\n\n Global HKY85 Analysis  \n\n");
global S=2.0;
GlobalHKY85Matrix = {{*,b,b*S,b}{b,*,b,b*S}
    {b*S,b,*,b}{b,b*S,b,*}};
Model GlobalHKY85 = (GlobalHKY85Matrix, obsFreqs);
```

```
Tree myTree = ((a,b),c,d);
LikelihoodFunction theLikFun = (myFilter, myTree);
Optimize(results,theLikFun);
fprintf(stdout,theLikFun);
```

The code for the first analysis is identical to that from *models.bf*. The global analysis introduces a new statement:

```
global S=2.0;
```

This statement declares S to be a global variable. By default, the description of a model (and variables within that model) is used as a template that is copied for every branch on the tree. An important fact is that we can not later redefine S as a local variable. The scope of a variable is determined at the time of its creation and can not be altered. In the statement defining *GlobalHKY85Matrix*, one observes that b is used as the transversion rate, while transitions occur at rate $b * S$.

More complex models

HyPhy has support for an infinite number of substitution models. Any Markov chain model using any finite sequence alphabet can be used. Models for codon and amino acid sequences are available through the Standard Analyses menu selection. We refer users who are interested in writing code for such alphabets to the files in the *Examples* subdirectory.

Imposing constraints on variables

Simple Constraints: relrate.bf.

The primary reason for developing *HyPhy* was to provide a system for performing likelihood analyses on molecular evolutionary data sets. In particular, we wanted to be able to describe and perform likelihood ratio tests (LRTs) easily. In order to perform an LRT it is first necessary to describe a constraint, or series of constraints, among parameters in the probability model. To illustrate the syntax of parameter constraints in *HyPhy*, examine the code in *relrate.bf*:

```
SetDialogPrompt("Select a nucleotide data file:");
DataSet myData = ReadDataFile (PROMPT_FOR_FILE);
DataSetFilter myFilter = CreateFilter (myData,1);
HarvestFrequencies (obsFreqs, myFilter, 1, 1, 1);
F81RateMatrix = {{* ,mu,mu,mu}{mu,* ,mu,mu}
{mu,mu,* ,mu}{mu,mu,mu,* }};
Model F81 = (F81RateMatrix, obsFreqs);
Tree myTree = (a,b,og);
```

```
fprintf(stdout,"\n Unconstrained analysis:\n\n");
LikelihoodFunction theLikFun = (myFilter, myTree, obsFreqs);
Optimize (paramValues, theLikFun);
fprintf  (stdout, theLikFun);
lnLA=paramValues[1][0];
dfA=paramValues[1][1];

fprintf(stdout,"\n\n\n Constrained analysis:\n\n");
myTree.a.mu := myTree.b.mu;
Optimize (paramValues, theLikFun);
fprintf  (stdout, theLikFun);
lnL0=paramValues[1][0];
df0=paramValues[1][1];

LRT=-2*(lnL0-lnLA);
Pvalue=1-CChi2(LRT,dfA-df0);
fprintf(stdout,"\n\nThe statistic ",LRT," has P-value ",
Pvalue,"\n\n");
```

The unconstrained analysis is of the simple type we have discussed previously. In the constrained analysis, however, we impose the constraint of equal substitution rates between lineages a and b with the command

```
myTree.a.mu := myTree.b.mu;
```

The results from this batchfile when applied to *three.seq* are:

```
 Unconstrained analysis:

Log Likelihood = -523.374642786834;
Tree myTree=(a:0.0313488,b:0.00634291,og:0.11779);


 Constrained analysis:

Log Likelihood = -525.013303516343;
Tree myTree=(a:0.018846,b:0.018846,og:0.116881);

The statistic 3.27732 has P-value 0.0702435
```

Since these models are nested, we can consider the likelihood ratio statistic, $-2(lnL_0 - lnL_A)$ to have an asymptotic chi-squared distribution. In this case, the test statistic has a value of 3.27732. Note in the batchfile how the likelihood values and parameter counts are returned by *Optimize* and stored in *paramValues*. The built-in function *CChi2* is the cumulative distribution function of the chi-squared distribution.

Molecular Clocks

Perhaps the most common molecular evolutionary hypothesis tested is that a set of sequences has evolved according to a molecular clock. It now seems quite clear that a global molecular clock exists for few, if any, gene sequences. In contrast, the existence of local molecular clocks among more closely related species is more probable. *HyPhy* allows for both types of constraints, including the possibility of testing for multiple local clocks for different user-defined clades in the same tree.

Global clocks: molclock.bf.

The batch file *molclock.bf* is a simple example of testing for a global molecular clock. The code should be familiar, except for the new `MolecularClock` statement, which declares that the values of the parameter *mu* should follow a molecular clock on the entire tree *myTree*. An important difference in this batch file is that the `Tree` statement defines a rooted tree. Had an unrooted tree been used, it would be treated as a rooted tree with a multifurcation at the root. When using time reversible models, which can't resolve the placement of the root, global molecular clock when given a rooted tree can be interpreted as: *there is a way to place the root on the root branch as to enforce global molecular clock on specified rates.* The section of code imposing the molecular clock constraint is:

```
fprintf(stdout,"\n\n Molecular Clock Analysis: \n");
MolecularClock(myTree,m);
LikelihoodFunction theLikFun = (myFilter, myTree);
Optimize(results,theLikFun);
```

Local clocks: localclocks.bf.

Particularly when studying data sets consisting of many species spanning a wide level of taxonomic diversity, it may be of interest to assign local molecular clocks to some clades. For instance, in a study of mammalian molecular evolution one might specify that each genus evolves in a clocklike manner, but that different genera evolve at different rates. To allow such analyses, the *MolecularClock* command can be applied to any node on a tree. Unlike the global clock of the previous case, it is not necessary for the *MolecularClock* command to be applied to a rooted tree; the placement of the *MolecularClock* command "roots" the tree, at least locally. To illustrate this feature, we use *localclocks.bf* in conjunction with the file *six.seq*. The relevant new sections of the code are the tree topology definition:

```
myTopology = "(((a,b)n1,(c,(d,e))n2),f)";
```

and the declaration of two local molecular clocks:

```
fprintf(stdout,"\n\n Local Molecular Clock Analysis: \n");
ClearConstraints(myTree);
MolecularClock(myTree.n1,m);
MolecularClock(myTree.n2,m);
LikelihoodFunction theLikFun = (myFilter, myTree);
Optimize(results,theLikFun);
```

The topology string used in *localclocks.bf* takes advantage of *HyPhy*'s extended syntax. Notice how we have named two of the internal nodes *n1* and *n2*. Those names override *HyPhy*'s default (and rather cryptic) node naming convention and allow us to call functions–in this case, *MolecularClock*– on the clades they tag. The syntax of the *MolecularClock* statements is rather C-like. `MolecularClock(myTree.n1,m);` imposes a local clock on the clade rooted at node *n1* in tree *myTree*. The parameter with clocklike behavior is *m*, the only option for the F81 model being used. The results using the datafile *six.seq* are:

```
UNCONSTRAINED ANALYSIS:
Log Likelihood = -685.473598259084;
Tree myTree=((a:0.0296674,b:0.00831723)n1:0.040811,
(c:0.0147138,(d:0.0142457,e:0.0328603)
Node7:0.0309969)n2:0.0130927,f:0.0517146);

 GLOBAL MOLECULAR CLOCK ANALYSIS:
Log Likelihood = -690.857603506283;
Tree myTree=((a:0.0181613,b:0.0181613)n1:0.0350919,
(c:0.0385465,(d:0.0195944,e:0.0195944)
Node7:0.0189521)n2:0.0147067,f:0.053838);

 P-value for Global Molecular Clock Test: 0.0292988


 LOCAL MOLECULAR CLOCK ANALYSIS:
Log Likelihood = -690.761234081996;
Tree myTree=((a:0.0190659,b:0.0190659)n1:0.0386549,
(c:0.0370133,(d:0.0189116,e:0.0189116)
Node7:0.0181017)n2:0.0128865,f:0.0537045);

 P-value for Local Molecular Clock Test: 0.0142589
```

By examining the output one finds that under the local clock model the two subtrees do, indeed, have clocklike branch lengths, yet the tree as a whole is not clocklike. The likelihood ratio test suggests that neither the global nor local clock assumption is correct.

Simulation Tools

The use of simulation in molecular evolutionary analysis has always been important. Simulation allows us to test statistical properties of methods, to assess the validity of theoretical asymptotic distributions of statistics, and to study the robustness of procedures to underlying model assumptions. More recently, methods invoking simulation have seen increased use. These techniques include numerical resampling methods for estimating variances or for computing confidence intervals, as well as parametric bootstrap procedures for estimating p-values of test statistics. *HyPhy* provides both parametric and nonparametric simulation tools, and examples of both are illustrated in the following sections.

The Bootstrap: bootstrap.bf.

The bootsrap provides, among other things, a simple nonparametric approach for estimating variances of parameter estimates. Consider *bootstrap.bf*. The relevant commands from the batch file are as follows. (Some lines of code have been deleted for clarity).

```
Model F81 = (F81RateMatrix, obsFreqs);
Tree myTree = (a,b,og);
LikelihoodFunction theLikFun = (myFilter, myTree);
Optimize (paramValues, theLikFun);

reps = 100;

for (bsCounter = 1; bsCounter<=reps; bsCounter = bsCounter+1)
{
DataSetFilter bsFilter = Bootstrap(myFilter,1);
HarvestFrequencies (bsFreqs, bsFilter, 1, 1, 1);
Model bsModel = (F81RateMatrix, bsFreqs);
Tree bsTree = (a,b,og);
LikelihoodFunction bsLik = (bsFilter, bsTree);
Optimize (bsParamValues, bsLik);
}
```

The first section of code is simply the completion of a typical data analysis, storing and printing results from the analysis of data in *myFilter*. The *for* loop is the heart of the batch file. For each of the *reps* replicates, we generate a new *DataSetFilter* named *bsFilter*. We do this by creating a bootstrap replicate from the existing *DataSetFilter* named *bsFilter*, which was created in the normal fashion. *bsFilter* will contain the same number of columns as *myFilter*. Once the new filter has been created, we recreate a *Model* named *bsModel* and a *Tree* named *bsTree*, which are then used in an appropriate *LikelihoodFunction* command. *Optimize* is used to find MLEs of the parameters. The end result of this batchfile is a table consisting of 100 sets of MLEs,

each from a bootstrap sample from the original data. Notice in the complete batch file (not shown in the code above) how we use the matrix variable *BSRes* to tabulate and report the average of all bootstrap replicates. More complex analyses, such as bootstrap confidence intervals, based on the bootstrap estimates can be programmed within the batchfile, or the results can be saved and imported into a spreadsheet for statistical analyses.

The *Permute* function, with syntax identical to *Bootstrap*, exists for applications where the columns in the existing *DataSetFilter* must appear exactly once in each of the simulated datasets. This feature may be useful for comparisons between the three codon positions or for studies investigating spatial correlations or spatial heterogeneity.

The Parametric Bootstrap: parboot.bf.

Another useful simulation tool is the parametric bootstrap. *HyPhy* provides the *SimulateDataSet* command to provide the type of model-based simulation required. In *parboot.bf* we find the following lines of code. Again, some lines have been deleted for clarity.

```
for (bsCounter = 1; bsCounter<=reps; bsCounter = bsCounter+1)
{
DataSet bsData = SimulateDataSet(theLikFun);
DataSetFilter bsFilter = CreateFilter (bsData,1);
HarvestFrequencies (bsFreqs, bsFilter, 1, 1, 1);
Model bsModel = (F81RateMatrix, bsFreqs);
Tree bsTree = (a,b,og);
LikelihoodFunction bsLik = (bsFilter, bsTree);
Optimize (bsParamValues, bsLik);
}
```

The end result is analagous to that of *bootstrap.bf*: we simulate *reps* datasets, find MLEs, and tabulate results. The fundamental difference is that the datasets are formed by simulating using the tree structure, evolutionary model, and parameters in *theLikFun* via the function *SimulateDataSet*. An important technical difference is that *SimulateDataSet* generates a *DataSet* as opposed to the *DataSetFilter* created by *Bootstrap*. Thus, we must use the *CreateFilter* command to create an appropriate filter.

Again note the use of *BSRes* for tabulating results, and also the use of *fscanf* for acquiring input from the user (see the *Batch Language Command Reference* for details).

Putting It All Together: *positions.bf*

As an example of the type of analysis *HyPhy* was designed to implement, we now describe the batchfile *positions.bf*. This file illustrates some of the features of the *CreateFilter* command by ignoring species *C* in *four.seq* and

by creating separate filters for each of the three codon positions. The HKY85 model is used as the basic substitution model. A *global* transition:transversion ratio, R, is created; its value is allowed to be shared by all three positions. In the "Combined Analysis", the entire data set is analyzed in the normal way, treating all sites identically. A second *LikelihoodFunction* is then created, in which the data are split into three partitions according to codon position. Each of the three partitions is allowed to evolve with a separate rate. However, the transition/transversion ratio is constrained to be the same for all three codon positions as well as for all lineages. The likelihood ratio test statistic comparing these two models is computed, and the statistical significance of the test is reported using both the chi-squared approximation and nonparametric bootstrapping.

The file *positions.bf* is rather complicated, so we will focus only on some of its key features.

Read and filter the data.

It is often the case that molecular data sets have some repeating underlying structure that we would like to exploit or study. For instance, coding regions might be described with the repeating structure $123123123\ldots$. In *positions.bf* we create separate *DataSetFilters* for first, second, and third codon positions. The command:

```
DataSetFilter myFilter1 = CreateFilter (myData,1,"<100>","0,1,3");
```

creates a *DataSetFilter* named *MyData1* that includes only the first nucleotide of each triplet. Likewise, the statement

```
DataSetFilter myFilter3 = CreateFilter (myData,1,"<001>","0,1,3");
```

creates a *DataSetFilter* named *MyData3* that includes only the third nucleotide of every triplet. Had we wished to create a filter consisting of both first and second positions, we would have used a statement like

```
DataSetFilter myFilter12 = CreateFilter (myData,1,"<110>","0,1,3");
```

Define a substitution model for each position.

The next portion of *positions.bf* creates a vector of observed frequencies for each of the filters using standard syntax.

```
HarvestFrequencies (obsFreqs, myFilter, 1, 1, 1);
HarvestFrequencies (obsFreqs1, myFilter1, 1, 1, 1);
HarvestFrequencies (obsFreqs2, myFilter2, 1, 1, 1);
HarvestFrequencies (obsFreqs3, myFilter3, 1, 1, 1);
```

Next, the basic substitution model is defined. We use the HKY85 model, with transversion parameter b and global transition:transversion ratio R. A separate *Model* is created for each partition, since they each use different frequencies:

```
global R;
HKY85RateMatrix = {{*,b,R*b,b}{b,*,b,R*b}{R*b,b,*,b}{b,R*b,b,*}};
Model HKY85 = (HKY85RateMatrix, obsFreqs);
Tree myTree = (a,b,d);
Model HKY851 = (HKY85RateMatrix, obsFreqs1);
Tree myTree1 = (a,b,d);
Model HKY852 = (HKY85RateMatrix, obsFreqs2);
Tree myTree2 = (a,b,d);
Model HKY853 = (HKY85RateMatrix, obsFreqs3);
Tree myTree3 = (a,b,d);
```

Define two likelihood functions.

We are now ready to set up *LikelihoodFunctions* and *Optimize* them. The analysis of the combined data set is routine:

```
LikelihoodFunction theLikFun = (myFilter,myTree);
Optimize (paramValues, theLikFun);
```

We also store some results for later use:

```
lnLik0 = paramValues[1][0];
npar0 = paramValues[1][1]+3;
fprintf  (stdout, theLikFun, "\n\n");
```

The statement `npar0 = paramValues[1][1]+3;` requires some explanation. The *Optimize* function always returns the number of parameters that were optimized as the `[1][1]` element of its returned matrix of results. Typically, we do not optimize over base frequency values, electing instead to simply use observed frequencies, which are usually very close to the maximum likelihood estimates. Since the frequencies are, in fact, estimated from the data, they need to be included in the parameter count. The value of *npar0*, therefore, includes the count of independent substitution parameters in the model (the number of which is returned by *Optimize*) along with the three independent base frequencies estimated from the data.

The *LikelihoodFunction* for the "partitioned" analysis simply uses the extended form of the *LikelihoodFunction* command:

```
LikelihoodFunction theSplitLikFun = (myFilter1,myTree1,
                                      myFilter2,myTree2,
                                      myFilter3,myTree3);
Optimize (paramValues, theSplitLikFun);
lnLik1 = paramValues[1][0];
npar1 = paramValues[1][1]+9;
```

Note the addition of the 9 estimated frequencies to the model's parameter count, 3 for each partition.

Find P-values for hypothesis tests.

Finally, we compute the P-value for the test of the combined analysis (null hypothesis) against the split model (alternative hypothesis). Two approaches are used. First, the normal chi-squared approximation to the LRT statistic:

```
LRT = 2*(lnLik1-lnLik0);
pValueChi2 = 1-CChi2 (LRT, npar1-npar0).
```

One can also estimate the P-value using the parametric bootstrap. The statement for simulating a random dataset based on *theLikFun* is

```
DataSet simData = SimulateDataSet(theLikFun);
```

The remaining part of the loop is basically a copy of the original analysis, with variable names adjusted to indicate that they are coming from simulated data. For each simulated dataset we compute the LRT, named *simLRT*, and compare it to the observed LRT. The estimate of the P-value is the proportion of simulated datasets with a LRT larger than that of the observed data. We keep track of the number of such events using the variable *count*:

```
simLRT = 2*(simlnLik1-simlnLik0);
if (simLRT > LRT)
{
        count = count+1;
}
```

and report the results:

```
fprintf(stdout,"\n\n*** P-value (Parametric BS)= ",count/reps,"\n");
```

The batchfile *positions.bf* provides a good example of the flexibility of *HyPhy*, and many of the same ideas could be used to develop analyses of multiple genes. Of particular importance for multilocus is the ability to mix local and global variables. To our knowledge, the type of modeling and testing flexibility demonstrated in *positions.bf* is unique.

Site-to-site rate heterogeneity.

One of the most important additions to recent models of sequence evolution is the incorporation of site-to-site rate heterogeneity, which allows the highly desirable property of some positions evolving quickly, some slowly, while others have intermediate rates. A variety of approaches have been introduced in the literature (Refs). In the first portion of this chapter, we demonstrated some of *HyPhy*'s basic functionality with regard to rate heterogeneity. We now continue this discussion, demonstrating the "traditional" approaches to modeling rate heterogeneity, as well as some novel features unique to *HyPhy*. We feel that the flexibility in modeling site-to-site rate heterogeneity is one of the strongest aspects of the software package.

The fundamental elements of incorporating site-to-site rate heterogeneity are demonstrated in the file *ratehet.bf*. There one will find an analysis labeled "Variable Rates Model 1", which simply uses the F81 nucleotide model with sites falling into one of four rate classes. The first rate class is an invariant class (ie, rate 0), while rates of the remaining three categories have relative rates of 1, 2, and 4. The frequencies of the four categories are assumed to be equal for illustration. The key section of code is the following:

```
category rateCat = (4, EQUAL, MEAN, , {{0}{1}{2}{4}}, 0, 4);

F81VarRateMatrix = {{*,rateCat*m,rateCat*m,rateCat*m}
                                    {rateCat*m,*,rateCat*m,rateCat*m}
                     {rateCat*m,rateCat*m,*,rateCat*m}
                     {rateCat*m,rateCat*m,rateCat*m,*}};

Model F81Var = (F81VarRateMatrix, obsFreqs);
```

The "category" statement defines a discrete probability distribution for the rates. In this case, there are four possible (relative) rates, 0, 1, 2, and 4, and the categories occur with equal frequencies (see the *HyPhy* documentation and the examples below for further information on the *category* statement). The second and third statements define a variant of the F81 model of nucleotide evolution. Had we left out the "rateCat" multiplier in the rate matrix, the model would be the standard F81 model. With the inclusion of "rateCat", which is defined in the first statement to be a category variable, we have a model declaring that each site evolves according to the F81 model, but that the rates vary from site to site in accordance to the distribution described in the *category* statement. Note that in this case the relative rates are specified by the user, so there is no rate haterogeneity parameter to be estimated from the data.

In the "Variable Rates Model 2" analysis we find an implementation of the slightly more complex (but more well-known) discrete gamma model, first described by Yang (1994). The key element in this analysis is simply a different *category* statement:

```
category rateCat = (4, EQUAL, MEAN, GammaDist(_x_,alpha,alpha),
CGammaDist(_x_,alpha,alpha),0,1e25,CGammaDist(_x_,alpha+1,alpha));
```

We again introduce a discrete distribution with four equiprobable classes, but this time the relative rates of those classes are provided by the gamma distribution. In turn, the arguments in the *category* statement declare

1. Use 4 rate categories
2. Assign equal frequencies to the 4 categories
3. Use the mean of each discretized interval to represent the rate for the corresponding class

4. The density function for the rates is the gamma density (which is a built-in function. Alternatively, the formula for any desired density could be entered.)

5. The cumulative density function is provided by the gamma distribution function (Again, this is a predefined function, and the cdf for any chosen density could be substituted.)

6. The relative rates are limited to the range 0 to 1×10^{25} (to make numerical work simpler)

7. The final argument is optional, and specifies a formula for the mean of each interval. If this argument were not provided, the mean would be evaluated numerically.

With this model, *HyPhy* would estimate the branch lengths for each branch in the tree, along with the shape parameter, α, that is specified in the *category* statement.

The third and final example in *ratehet.bf* allows rates to vary according to an exponential distribution. The *category* statement in this case is essentially the same as for the gamma distribution, but with the density and distribution functions for the exponential distribution used instead:

```
category rateCat = (4, EQUAL, MEAN, alpha*Exp(-alpha*_x_),
1-Exp(-alpha*_x_), 0, 1e25, -_x_*Exp(-alpha*_x) + (1-Exp(-alpha*_x_))/alpha);
```

This fundamental approach can be used to fit any discretized density to data by simply writing an appropriate *category* statement and combining it with any desired substitution matrix. A number of examples are provided in the sample files in the *HyPhy* distribution.

In the file *twocats.bf* we demonstrate a new idea in modeling rate heterogeneity, the possibility of moving beyond the simple idea of each site having its own rate. For illustration, we show that it is simple to define a model that allows each site to have its own transition and transversion rate, but sites with high transition rates need not also have high transversion rates. We demonstrated an application of this approach to codon-based models based on synonymous and nonsynonymous rates in the first half of the chapter. The basic approach is the same as for the previous examples, we will use the *category* statement to define distributions of rate heterogenetiy. However, in this case we will use two *category* statements, one for transitions and one for transversions.

The first analysis in *twocats* is essentially the discrete gamma model found in *ratehet.bf*, but with 16 categories rather than four. The second analysis introduces separate distributions for transitions and transversions. Each type of rate is assumed to come from a (discrete) gamma distribution with four categories, but each distribution has its own parameters. This model leads to a model with $4 \times 4 = 16$ rate categories, and thus has computational complexity equal to the 16-category discrete gamma in the first analysis. The *category* statements have the same basic format as the previous examples:

```
category catTS = (4, EQUAL, MEAN, GammaDist(_x_,alphaS,alphaS),
CGammaDist(_x_,alphaS,alphaS),0,1e25,CGammaDist(_x_,alphaS+1,alphaS));
```

```
category catTV = (4, EQUAL, MEAN, GammaDist(_x_,alphaV,beta),
CGammaDist(_x_,alphaV,beta),0,1e25,CGammaDist(_x_,alphaV+1,beta)*alphaV/beta);
```

An important mathematical fact arises at this point. Traditionally, the gamma distribution in rate analyses has been described only by its "shape" parameter". The gamma distribution in general is described by a shape parameter and a scale parameter. The confounding of rates and times allows for the (arbitrary) determination of one of the two parameters, and for simplicity the two parameters have simply been assumed to be equal. When we move to the case of two gamma distributions, we still have this level of freedom to arbitrarily assign one parameter. In this example, we have maintained the "traditional" style for the transition rates (see the *category* statement for *catTS*), but we must use both the shape and scale parameters for the second distribution. Thus, we end up with three parameters that govern the distributional form for the transition and transversion rates: *alphaS*, the shape parameter for the transition rate distribution, along with *alphaV* and *beta*, the shape and scale parameters for the gamma distribution describing transversion rates.

We must still introduce these category variables into the substitution matrix, and examining the definition of *HKY85TwoVarRateMatrix* we see that transition rates are multiplied by *catTS*, while transversion rates are multipled by *catTV*.

Analyzing codon data.

So far we have considered only nucleotide alignments and evolutionary models as examples. The example included in the file *codon.bf* we will discuss how to read and filter codon data and define substitution models which operate at the level of codons.

Defining codon data filters.

Codon data sets are nucleotide sequences where the unit of evolution is a triplet of nucleotides, and some states (stop codons) are disallowed. The task of making *HyPhy* interpret a nucleotide alignment as codons is handled by supplying a few additional parameters in a call to *CreateFilter*. Consider the following line in *codons.bf*:

```
DataSetFilter codonFilter = CreateFilter(myData,3,"","","TAA,TAG,TGA");
```

The second argument of 3 instructs *HyPhy* to consider triplets of characters in the data set *myData* as units of evolution. If it had been 2, then the filter would consist of di-nucleotides. The empty third and the fourth arguments include all sequences and sites in the filter. The fifth argument is the comma separated

list of **exclusions**, i.e. character states that are not allowed. One can easily recognize that the list includes the three stop codons for the universal genetic code. All sites in the original nucleotide alignment that contained at least one of the excluded states would be omitted from the filter, and a message would be written to *messages.log*, located in the main *HyPhy* directory.

Filter *myFilter* consists of data $4^3 - 3 = 61$ states, i.e. all sense codons in the universal genetic code, therefore any substitution model compatible with this filter must describe a process with 61 states and use a 61×61 rate matrix. Before we proceed with the definition of this matrix, a crucial question must be answered: how does *HyPhy* index codons? For example, which entry in the rate matrix will describe the change from codon ATC to codon TTC? *HyPhy* uses a uniform indexing scheme, which is rather straightforward. The default nucleotide alphabet is ordered as ACGT, and each character was assigned an index in that order: A=0, C=1, G=2, T=3 (note that all indexing starts at 0, as in the programming language C). In previous examples, we used this mapping to define nucleotide rate matrices. For example, the entry in row 2 and column 3 would define the rate of $G \rightarrow T$ substitutions. Analogously, all sense codons all ordered alphabetically: AAA, AAC, AAG, AAT, ACA, ..., TTG, TTT, excluding stop codons, with the corresponding indexing from 0 to 60. It is easy to check that ATC will have the index of 13, whereas TTC is assigned the index of 58. Consequently, the rate of ATC to TTC substitutions should be placed in row 13 and column 58 of the rate matrix.

A 61×61 rate matrix has 3721 entries, hence defining them one by one would be a daunting task indeed, so we need a way to avoid an explicit enumeration. Consider the MG94xHKY85 model (2) explained in section 2.4. Each substitution rate can be classified by determining the following four attributes: (i) is the change one-step or multi-step, (ii) is the change synonymous or non-synonymous, (iii) is the change a transition or a transversion and (iv) what is the equilibrium frequency of the target nucleotide.

A compact way to define the model is to loop through all possible 3721 pairs of codons, answer the four questions above, and assign the appropriate rate to the matrix cell. *HyPhy* has no intrinsic knowledge of how codons are translated to amino-acids, and this information is needed to decide whether a nucleotide substitution is synonymous or non-synonymous. *codons.bf* contains such a map for the universal genetic code in the matrix *UniversalGeneticCode*. For all possible 64 codons there are 21 possible translations (20 amino acids and a 'stop'). Each of the 64 cells of *UniversalGeneticCode* contains an amino acid (or stop) code from 0 to 20, whose meaning is explained in the comments in *codons.bf*. We refer the reader to the code and comments in *codons.bf* for implementation details. The implementation is straightforward, but somewhat obtuse. Once the reader becomes comfortable with referencing codons by their indices and interpreting them, the code should be clear.

The file *codons.bf* illustrates several other useful concepts:

- How to define and call user functions. Function *BuildCodonFrequencies* is employed to compute codon equilibrium frequencies based on observed nucleotide proportions, defined in (3).
- The use of a built-in variable to reference the tree string present in the data file (*DATAFILE_TREE*)
- The use of double underscore operator to substitute numerical values of arguments into formula definitions and avoid unwanted dependencies.

Lastly, the *codons.bf* writes data out for further processing with a standard file from *HyPhy* distribution to perform posterior Bayesian analysis, as discussed in section 2.4.

4 Conclusion

This chapter has provided an overview of the basic features and use of the *HyPhy* system. With a programming language at its core, users may elect to write their own likelihood-based molecular evolutionary analyses. A graphical user interface offers much of the power of the batch language, allowing users to fit complex, customizable models to sequence alignments. The user interface also provides access to the parametric bootstrap features of *HyPhy* for carrying out tests of both nested and non-nested hypotheses. Many features of the package, of course, could not be described in this chapter. For instance, *HyPhy* includes a model editor for describing new stochastic models to be used in analyses, and the graphical user interface provides a mechanism to define arbitrary constraints among parameters for construction of likelihood ratio tests. Its authors continue to develop *HyPhy*, with a goal of providing a flexible, portable, and powerful system for carrying out cutting-edge molecular evolutionary analyses.

References

Akaike, H. 1974. A new look at the statistical model identification. IEEE Transactions on Automatic Control 119:716–723.

Cox, D. R. 1961. Tests of separate families of hypotheses. Pp. 105–123, *in* Proceedings of the 4th berkeley symposium, vol. 1. University of California Press, Los Angeles, CA.

Felsenstein, J. 1981. Evolutionary trees from DNA-sequences - a maximum-likelihood approach. Journal Of Molecular Evolution 17:368–376.

Goldman, N. 1993. Statistical tests of models of DNA substitution. J. Mol. Evol. 36:182–198.

Hasegawa, M., H. Kishino, and T. Yano. 1985. Dating of the human-ape splitting by a molecular clock of mitochondrial dna. Molecular Biology and Evolution 21:160–174.

Jones, D., W. Taylor, and J. Thornton. 1992. The rapid generation of mutation data matrices from protein sequences. Computer Applications in Biosciences 8(3):275–282.

Kosakovsky Pond, S., and S. Muse. 2004. Column sorting: Rapid calculation of the phylogenetic likelihood function. To appear in Systematic Biology .

Muse, S. V., and B. S. Gaut. 1994. A likelihood approach for comparing synonymous and nonsynonymous nucleotide substitution rates, with application to the chloroplast genome. Molecular Biology And Evolution 11:715–724.

Muse, S. V., and B. S. Gaut. 1997. Comparing patterns of nucleotide substitution rates among chloroplast loci using the relative ratio test. Genetics 146:393–399.

Nielsen, R., and Z. H. Yang. 1998. Likelihood models for detecting positively selected amino acid sites and applications to the HIV-1 envelope gene. Genetics 148:929–936.

Suzuki, Y., and T. Gojobori. 1999. A method for detecting positive selection at single amino acid sites. Molecular Biology And Evolution 16:1315–1328.

Yang, Z. 1993. Maximum-likelihood estimation of phylogeny from DNA sequences when substitution rates differ over sites. Molecular Biology and Evolution 10:1396–1401.

Yang, Z. 1994. Maximum likelihood phylogenetic estimation from DNA sequences with variable rates over sites: approximate methods. J. Mol. Evol. 39:105–111.

Yang, Z. H. 1996. Among-site rate variation and its impact on phylogenetic analyses. Trends In Ecology & Evolution 11:367–372.

Yang, Z. H., R. Nielsen, N. Goldman, and A. M. K. Pedersen. 2000. Codon-substitution models for heterogeneous selection pressure at amino acid sites. Genetics 155:431–449.