# User Guide

# PAML:

# Phylogenetic Analysis by Maximum Likelihood

**Version 4.4 (June 2010)**

**Ziheng Yang**

Suggested citations:

Yang, Z. 1997. PAML: a program package for phylogenetic analysis by maximum likelihood *Computer Applications in BioSciences* **13**:555-556.

Yang, Z. 2007. PAML 4: a program package for phylogenetic analysis by maximum likelihood. *Molecular Biology and Evolution* **24**: 1586-1591 (`http://abacus.gene.ucl.ac.uk/software/paml.html`).

Recent changes and bug fixes are documented in the file doc/pamlHistory.txt.

The author can be reached at the following address. However, emails are not welcome: please post your questions and comments at the PAML discussion site: http://gsf.gc.ucdavis.edu/.

Ziheng Yang

Department of Biology
University College London
Gower Street
London WC1E 6BT
England

Fax: +44 (20) 7679 7096

# Table of Contents

# 1  Overview

PAML (for Phylogenetic Analysis by Maximum Likelihood) is a package of programs for phylogenetic analyses of DNA and protein sequences using maximum likelihood.

## PAML documentation

Besides this manual, please note the following resources:

- PAML web site: http://abacus.gene.ucl.ac.uk/software/PAML.html has information about downloading and compiling the programs.

- PAML FAQ page: http://abacus.gene.ucl.ac.uk/software/pamlFAQs.pdf

- PAML discussion group at http://www.rannala.org/phpBB2/, where you can post bug reports and questions.

## What PAML programs can do

The PAML package currently includes the following programs: baseml, basemlg, codeml, evolver, pamp, yn00, mcmctree, and chi2. A brief overview of the most commonly used models and methods implemented in PAML is provided by Yang (2007). The book (Yang 2006) describes the statistical and computational details. Examples of analyses that can be performed using the package include

- Comparison and tests of phylogenetic trees (baseml and codeml);

- Estimation of parameters in sophisticated substitution models, including models of variable rates among sites and models for combined analysis of multiple genes or site partitions (baseml and codeml);

- Likelihood ratio tests of hypotheses through comparison of implemented models (baseml, codeml, chi2);

- Estimation of divergence times under global and local clock models (baseml and codeml);

- Likelihood (Empirical Bayes) reconstruction of ancestral sequences using nucleotide, amino acid and codon models (baseml and codeml);

- Generation of datasets of nucleotide, codon, and amino acid sequence by Monte Carlo simulation (evolver);

- Estimation of synonymous and nonsynonymous substitution rates and detection of positive selection in protein-coding DNA sequences (yn00 and codeml).

- Bayesian estimation of species divergence times incorporating uncertainties in fossil calibrations (mcmctree).

The strength of PAML is its collection of sophisticated substitution models. Tree search algorithms implemented in baseml and codeml are rather primitive, so except for very small data sets with say, <10 species, you are better off to use another package, such as phylip, paup, or mrBayes, to infer the tree topology. You can get a collection of trees from other programs and evaluate them using baseml or codeml as user trees.

baseml and codeml. The program baseml is for maximum likelihood analysis of nucleotide sequences. The program codeml is formed by merging two old programs: codonml, which implements the codon substitution model of Goldman and Yang (1994) for protein-coding DNA sequences, and aaml, which implements models for amino acid sequences. These two

are now distinguished by the variable seqtype in the control file codeml.ctl, with 1 for codon sequences and 2 for amino acid sequences.  In this document I use codonml and aaml to mean codeml with seqtype = 1 and 2, respectively.  The programs baseml, codonml, and aaml use similar algorithms to fit models by maximum likelihood, the main difference being that the unit of evolution in the Markov model, referred to as a "site" in the sequence, is a nucleotide, a codon, or an amino acid for the three programs, respectively.  Markov process models are used to describe substitutions between nucleotides, codons or amino acids, with substitution rates assumed to be either constant or variable among sites.

evolver.  This program can be used to simulate sequences under nucleotide, codon and amino acid substitution models.  It also has some other options such as generating random trees, and calculating the partition distances (Robinson and Foulds 1981) between trees.

basemlg.  This program implements the (continuous) gamma model of Yang (1993).  It is very slow and unfeasible for data of more than 6 or 7 species.  Instead the discrete-gamma model in baseml should be used.

mcmctree.  This implements the Bayesian MCMC algorithm of Yang and Rannala (2006) and Rannala and Yang (2007) for estimating species divergence times.

pamp.  This implements the parsimony-based analysis of Yang and Kumar (1996).

yn00.  This implements the method of Yang and Nielsen (2000) for estimating synonymous and nonsynonymous substitution rates ($d_S$ and $d_N$) in pairwise comparisons of protein-coding DNA sequences.

chi2.  This is for conducting likelihood ratio tests.  It calculates the chi square critical values, which you can compare with your test statistic calculated from the real data to determine whether the test is significant at the 5% or 1% levels.  Run the program by typing the program name "chi2".  The program can also calculate the *P* value when you input the test statistic and the d.f.  Run the program by typing "chi2 p".

## What PAML programs cannot do

There are many things that you might well expect a phylogenetics package should do but PAML cannot.  Here is a partial list, provided in the hope that it might help you avoid wasting time.

- Sequence alignment.  You should use some other programs such as Clustal or TreeAlign to align the sequences automatically or do a manual alignment, perhaps with assistance from programs such as BioEdit and GeneDoc.  Manual adjustment does not seem to have reached the mature stage to be entirely trustable so you should always do manual adjustment if you can.  If you are constructing thousands of alignments in genome-wide analysis, you should implement some quality control, and, say, calculate some measure of sequence divergence as an indication of the unreliability of the alignment.  For coding sequences, you might align the protein sequences and construct the DNA alignment based on the protein alignment.  Note that alignment gaps are treated as missing data in baseml and codeml (if `cleandata` = 1).  If `cleandata` = 1, all sites with ambiguity characters and alignment gaps are removed.

- Gene prediction.  The codon-based analysis implemented in codonml (codeml for codons with seqtype = 1) assumes that the sequences are pre-aligned exons, the sequence length is an exact multiple of 3, and the first nucleotide in the sequence is codon position 1.  Introns, spacers and other noncoding regions must be removed and the coding sequences must be aligned before running the program.  The program cannot process sequences downloaded

directly from GenBank, even though the CDS information is there. It cannot predict coding regions either.

- Tree search in large data sets. As mentioned earlier, you should use another program to get a tree or some candidate trees and use them as user trees to fit models that might not be available in other packages.

# 2  Compiling and punning PAML programs

PAML programs use the old simple command-line interface. You download the archive from the PAML web site, typically named PAML*.*.tar.gz, and unpack the files onto your hard disk. This is one file for all platforms. Executables for windows are included, while for UNIX or MAC OS X, you need compile the programs before running them.

## Windows

The executables for Windows are included in the package.

1. Go to the PAML web site http://abacus.gene.ucl.ac.uk/software/paml.html and download the latest archive and save it on your hard disk. Unpack, say, using WinZip, the archive into a folder, say D:\software\paml\. Remember the name of the folder.

2. Start a "Command Prompt". Go to "Start – Programs – Accessories". Alternatively, choose "Start – Run" and type the command **cmd** and hit OK. You can right click on the title bar to change the font, colour, size etc. of the window.

3. Change directory to the paml folder. For example you type one of the following.

   ```
   d:
   cd \software\paml
   dir
   ```

4. Note that Windows commands and file names are case-insensitive. The folder src\ contains the source files. The examples\ contains various example files, and bin\ contains Windows executables. You can use Windows Explorer to look at the files. To run the program baseml using the default control file baseml.ctl in the current folder, you can a command somewhat like the following.

   ```
   bin\baseml

   D:\software\paml4\bin\baseml
   ```

   This causes baseml to read the default control file baseml.ctl in the current folder and do the analysis according to its specifications. Now you can print out a copy of baseml.ctl, and open a text editor to view the relevant sequence and tree files.
   Similarly you can run codeml and look at the control file codeml.ctl.

Next you can prepare your own sequence data files and tree files. Control files and other input files are all plain text files. A common problem occurs due to differences in the way UNIX and Windows deal with carriage return or line breaks. If you use MS Word to prepare the input files, you should save them as "Text with line breaks" or "Text without line breaks". Sometimes only one of those two works. Do not save the file as a Word document. I have collected some notes in the section "Overcoming Windows Annoyances" in Appendix B.

If you insist on double-clicking, you can start Widows Explorer, and copy the executables to the folder that contains the control file, and then double-click on the executables.

## UNIX

UNIX executables are not provided in the package, so you will have to compile them using the source files included in the package, in the src/folder. Note that UNIX commands and file or folder names are case-sensitive. The following assumes that you are at the UNIX prompt.

1. Go to the PAML web site http://abacus.gene.ucl.ac.uk/software/paml.html and download the lat

2. est archive and save on your hard disk.  Unpack it using gzip, with a command like the following (replace the version numbers and use the correct name for the archive file)

```
gzip –d paml4.tar.gz
tar xf paml4.tar
```

3. You can use ls to look at the files in the folder.  Delete the Windows executables (.exe files) in the bin folder. Then cd to the src/ folder to compile using make.

```
ls -lF bin  (this should list the .exe files in the bin folder)
rm -r bin/*.exe
cd src
make
ls -lF
rm *.o
mv baseml basemlg codeml pamp evolver yn00 chi2 ../bin
cd ..
bin/codeml
```

4. Those commands compile the programs and generate executables called baseml, basemlg, codeml, pamp, evolver, yn00, and chi2, which you can see with the ls command.  Then remove (rm) the intermediate object files *.o, and move (mv) the compiled executables into bin/ folder in the PAML main folder (that is, ../bin from paml/src/).  Then cd  to the PAML main folder and run codeml, using the default control file codeml.ctl.  You can then print out a copy of codeml.ctl and look at it (and the main result file mlc).

If the compilation (the make command) is unsuccessful, you might have to open and edit the file Makefile before issuing the make command.  For example, you can change cc to gcc and -fast to -O3 or -O4.  If that none of these works, look at the file readme.txt in the src/ folder for compiling instructions.  You can copy the compiling commands onto the command line.  For example

```
cc –o baseml baseml.c tools.c –lm
cc –o codeml codeml.c tools.c –lm
```

would compile baseml and codeml using the C compiler cc.  However, in this case code optimization is not turned on.  You should use compiler switches to optimize the code, say,

```
cc –o codeml –O3 codeml.c tools.c –lm
```

Finally, if your current folder is not on your search path, you will have to add ./ in front of the executable file name even if the executable is in your current working folder; that is, use ./codeml instead of codeml to run codeml.

## Mac OS X

Since Mac OSX is UNIX, you should follow the instructions for UNIX above.  Open a command terminal (Applications-Utilities-Terminal) and then compile and run the programs from the terminal. You cd to the paml/src/ folder and look at the readme.txt or Makefile files.  See above.  If you type commands gcc or make and get a "Command not found" error, you will have to download the Apple Developer's Toolkit at the Apple web site http://developer.apple.com/tools/.  There are some notes about running programs on MAC OS X or UNIX at the FAQ page.

I have stopped distributing executables for old MACs running OS 9 or earlier.

## Running a program

As indicated above, you run a program by typing its name from the command line.  You should know which folder your sequence file, tree file, and control file are, relative to your working folder. If inexperienced, you may copy the executables to the folder containing your data files.  Depending

on the model used, `codeml` may need a data file such as `grantham.dat`, `dayhoff.dat`, `jones.dat`, `wag.dat`, `mtREV24.dat`, or `mtmam.dat`, so you should copy these files as well.

The programs produce result files, with names such as `rub`, `lnf`, `rst`, or `rates`. You should not use these names for your own files as otherwise they will be overwritten.

## *Example data sets*

The `examples/` folder contains many example data sets. They were used in the original papers to test the new methods, and I included them so that you could duplicate our results in the papers. Sequence alignments, control files, and detailed readme files are included. They are intended to help you get familiar with the input data formats and with interpretation of the results, and also to help you discover bugs in the program. If you are interested in a particular analysis, get a copy of the paper that described the method and analyze the example dataset to duplicate the published results. This is particularly important because the manual, as it is written, describes the meanings of the control variables used by the programs but does not clearly explain how to set up the control file to conduct a particular analysis.

**examples/HIVNSsites/:** This folder contains example data files for the HIV-1 env V3 region analyzed in Yang *et al.* (2000b). The data set is for demonstrating the NSsites models described in that paper, that is, models of variable ω ratios among amino acid sites. Those models are called the "random-sites" models by Yang & Swanson (2002) since *a priori* we do not know which sites might be highly conserved and which under positive selection. They are also known as "fishing-expedition" models. The included data set is the 10th data set analyzed by Yang *et al.* (2000b) and the results are in table 12 of that paper. Look at the readme file in that folder.

**examples/lysin/:** This folder contains the sperm lysin genes from 25 abalone species analyzed by Yang, Swanson & Vacquier (2000a) and Yang and Swanson (2002). The data set is for demonstrating both the "random-sites" models (as in Yang, Swanson & Vacquier (2000a)) and the "fixed-sites" models (as in (Yang and Swanson 2002)). In the latter paper, we used structural information to partition amino acid sites in the lysin into the "buried" and "exposed" classes and assigned and estimated different ω ratios for the two partitions. The hypothesis is that the sites exposed on the surface are likely to be under positive selection. Look at the readme file in that folder.

**examples/lysozyme/:** This folder contains the primate lysozyme *c* genes of Messier and Stewart (1997), re-analyzed by Yang (1998). This is for demonstrating codon models that assign different ω ratios for different branches in the tree, useful for testing positive selection along lineages. Those models are sometimes called branch models or branch-specific models. Both the "large" and the "small" data sets in Yang (1998) are included. Those models require the user to label branches in the tree, and the readme file and included tree file explain the format in great detail. See also the section "Tree file and representations of tree topology" later about specifying branch/node labels.

The lysozyme data set was also used by Yang and Nielsen (2002) to implement the so-callled "branch-site" models, which allow the ω ratio to vary both among lineages and among sites. Look at the readme file to learn how to run those models.

**examples/MouseLemurs/:** This folder includes the mtDNA alignment that Yang and Yoder (2003) analyzed to estimate divergence dates in mouse lemurs. The data set is for demonstrating maximum likelihood estimation of divergence dates under models of global and local clocks. The most sophisticated model described in that paper uses multiple

calibration nodes simultaneously, analyzes multiple genes (or site partitions) while accounting for their differences, and also account for variable rates among branch groups. The readme file explains the input data format as well as model specification in detail. The readme2 file explains the ad hoc rate smoothing procedure of Yang (2004).

**examples/mtCDNA/:** This folder includes the alignment of the 12 protein-coding genes on the same strand of the mitochondrial genome from seven ape species analyzed by Yang, Nielsen, & Hasegawa (1998) under a number of codon and amino acid substitution models. The data set is the "small" data set referred to in that paper, and was used to fit both the "mechanistic" and empirical models of amino acid substitution as well as the "mechanistic" models of codon substitution. The model can be used, for example, to test whether the rates of conserved and radical amino acid substitutions are equal. See the readme file for details.

**examples/TipDate/:** This folder includes the example data file used by Rambaut (2000) in his description of his TipDate models, for viral sequences with known dates of sequence determination. The readme file explains how to use baseml to fit the TipDate model, a global clock but with sequences determined at different dates. Local clock models can be applied as well. See the examples/MouseLemurs/ folder for how to do this. Note that I use the symbol @ in the sequence name to prefix the date of sequence determination. The file here is readable by Rambaut's TipDate program, but the file in his package requires some editing (by inserting the @ symbol) before it can be read by baseml.

Some other data files are included in the package as well. The details follow.

**brown.nuc** and **brown.trees**: the 895-bp mtDNA data of Brown *et al.* (1982), used in Yang *et al.* (1994) and Yang (1994b) to test models of variable rates among sites.

**mtprim9.nuc** and **9s.trees**: mitochondrial segment consisting of 888 aligned sites from 9 primate species (Hayasaka et al. 1988), used by Yang (1994a) to test the discrete-gamma model and Yang (1995) to test the auto-discrete-gamma models.

**abglobin.nuc** and **abglobin.trees**: the concatenated α- and β-globin genes, used by Goldman and Yang (1994) in their description of the codon model. abglobin.aa is the alignment of the translated amino acid sequences.

**stewart.aa** and **stewart.trees**: lysozyme protein sequences of six mammals (Stewart et al. 1987), used by Yang *et al.* (1995a) to test methods for reconstructing ancestral amino acid sequences.

# 3  Data file formats

## Sequence data file format

Have a look at some of the example data files in the package (.nuc, .aa, and .nex). As long as you get your data file into one of the formats, PAML programs should be able to read it. The "native" format is the PHYLIP format used in Joe Felsenstein's PHYLIP package (Felsenstein 2005) (but see below). PAML has limited support for the NEXUS file format used by PAUP and MacClade. Only the sequence data or trees are read, and command blocks are ignored. PAML does not deal with comment blocks in the sequence data block, so please avoid them.

### Sequential and interleaved formats

Below is an example of the PHYLIP format (Felsenstein 2005). The first line contains the number of species and the sequence length (possibly followed by option characters). *For codon sequences (`codeml` with `seqtype = 1`), the sequence length in the sequence file refers to the number of nucleotides rather than the number of codons.* The only options allowed in the sequence file are I, S, P, C, and G. The sequences may be in either *interleaved* format (option I, example data file `abglobin.nuc`), or *sequential* format (option S, example data file `brown.nuc`). The default option is S, so you don't have to specify it. Option G is used for combined analysis of multiple gene data and is explained below. The following is an example data set in the sequential format. It has 4 sequences each of 60 nucleotides (or 20 codons).

```
   4 60
sequence 1
AAGCTTCACCGGCGCAGTCATTCTCATAAT
CGCCCACGGACTTACATCCTCATTACTATT
sequence 2
AAGCTTCACCGGCGCAATTATCCTCATAAT
CGCCCACGGACTTACATCCTCATTATTATT
sequence 3
AAGCTTCACCGGCGCAGTTGTTCTTATAAT
TGCCCACGGACTTACATCATCATTATTATT
sequence 4
AAGCTTCACCGGCGCAACCACCCTCATGAT
TGCCCATGGACTCACATCCTCCCTACTGTT
```

*Species/sequence names.* Do not use the following special symbols in a species/sequence name: ", : # ( ) $ =" in a species name as they are used for special purposes and may confuse the programs. The symbol @ can be used as part and end of the sequence name to specify the date of determination of that sequence, for example, virus1@1984. The @ symbol is considered part of the name and the sequence was determined in 1984. The maximum number of characters in a species name (LSPNAME) is specified at the beginning of the main programs `baseml.c` and `codeml.c`. In PHYLIP, exactly 10 characters are used for a species name, which I often found to be too restrictive. So I use a default value of 30. To make this discrepancy less a problem, PAML considers two consecutive spaces as the end of a species name, so that the species name does not have to have exactly 30 (or 10) characters. To make this rule work, you should not have two consecutive spaces *within* a species name. For example the above data set can have the following format too.

```
   4 60
sequence 1   AAGCTTCACCGGCGCAGTCATTCTCATAAT
CGCCCACGGACTTACATCCTCATTACTATT
sequence 2   AAGCTTCACCGGCGCAATTATCCTCATAAT
CGCCCACGGACTTACATCCTCATTATTATT
sequence 3   AAGCTTCACC GGCGCAGTTG TTCTTATAAT
TGCCCACGGACTTACATCATCATTATTATT
sequence 4   AAGCTTCACCGGCGCAACCACCCTCATGAT
TGCCCATGGACTCACATCCTCCCTACTGTT
```

If you want the file to be readable by both PHYLIP and PAML, you should limit the number of characters in the name to 10 and separate the name and the sequence by at least two spaces.

In a sequence, T, C, A, G, U, t, c, a, g, u are recognized as nucleotides (for `baseml`, `basemlg` and `codonml`), while the standard one-letter codes (A, R, N, D, C, Q, E, G, H, I, L, K, M, F, P, S, T, W, Y, V or their lowercase equivalents) are recognized as amino acids. Ambiguity characters (undetermined nucleotides or amino acids) are allowed as well. Three special characters ".", "-", and "?" are interpreted like this: a dot means the same character as in the first sequence, a dash means an alignment gap, and a question mark means an undetermined nucleotide or amino acid. Non-alphabetic symbols such as ><!'"£$%&^[](){}0123456789 inside a sequence are simply ignored and can be freely used as signposts. Lines do not have to be equally long and you can put the whole sequence on one line.

The way that ambiguity characters and alignment gaps are treated in `baseml` and `codeml` depends on the variable `cleandata` in the control file. In the maximum likelihood analysis, sites at which at least one sequence involves an ambiguity character are removed from all sequences before analysis if `cleandata` = 1, while if `cleandata` = 0, both ambiguity characters and alignment gaps are treated as ambiguity characters. In the pairwise distance calculation (the lower-diagonal distance matrix in the output), `cleandata` = 1 means "complete deletion", with all sites involving ambiguity characters and alignment gaps removed from all sequences, while `cleandata` = 0 means "pairwise deletion", with only sites which have missing characters in the pair removed.

There are no models for insertions and deletions in the PAML programs. So an alignment gap is treated as an ambiguity (that is, a question mark ?). Note also that for codon sequences, removal of any nucleotide means removal of the whole codon.

Notes may be placed at the end of the sequence file and will be ignored by the programs.

*Option G:* This option is for combined analyses of heterogeneous data sets such as data of multiple genes or data of the three codon positions. The sequences must be concatenated and the option is used to specify which gene or codon position each site is from.

There are three formats with this option. The first format is illustrated by an excerpt of a sequence file listed below. The example data of Brown *et al.* (1982) are an 895-bp segment from the mitochondrial genome, which codes for parts of two proteins (ND4 and ND5) at the two ends and three tRNAs in the middle. Sites in the sequence fall naturally into 4 classes: the three codon positions and the tRNA coding region. The first line of the file contains the option character G. The second line begins with a G at the first column, followed by the number of site classes. The following lines contain the site marks, one for each site in the sequence (or each codon in the case of `codonml`). The site mark specifies which class each site is from. If there are *g* classes, the marks should be 1, 2, ..., *g*, and if *g* > 9, the marks need to be separated by spaces. The total number of marks must be equal to the total number of sites in each sequence.

```
  5 895 G
G 4
3
123123123123123123123123123123123123123123123123123123123
123123123123123123123123123123123123123123123123123123123
123123123123123123123123123123123123123123123123123123123
123123123123123123123123123123123123123123123123123123123
123123123123123123123123123123123123123123123123123123123
123123123123123123123123123123123123123123123123123123123
123123123123123123123123123123123123123123123123123123123
123123123123123123123123123123123231231231231
44444444444444444444444444444444444444444444444444444444444
```

```
4444444444444444444444444444444444444444444444444444444444
4444444444444444444444444444444444444444444444444444444444
44444444444444444
123123123123123123123123123123123123123123123123123123123
123123123123123123123123123123123123123123123123123123123
123123123123123123123123123123123123123123123123123123123
12312312312312312312312312312312312312312312312312312312
Human
AAGCTTCACCGGCGCAGTCATTCTCATAATCGCCCACGGACTTACATCCTCATTACTATT
CTGCCTAGCAAACTCAAACTACGAACGCACTCACAGTCGCATCATAATC........
Chimpanzee
.........
```

The second format is useful if the data are concatenated sequences of multiple genes, shown below for an example data set. This sequence has 1000 nucleotides from 4 genes, obtained from concatenating four genes with 100, 200, 300, and 400 nucleotides from genes 1, 2, 3, and 4, respectively. The "lengths" for the genes must be on the line that starts with G, *i.e.*, on the second line of the sequence file. (This requirement allows the program to determine which of the two formats is being used.) The sum of the lengths for the genes should be equal to the number of nucleotides, amino acids, or codons in the combined sequence for `baseml` (or `basemlg`), `aaml`, and `codonml`, respectively.

```
5 1000 G
G 4 100 200 300 400
Sequence 1
TCGATAGATAGGTTTTAGGGGGGGGGGGTAAAAAAAAA.......
```

The third format applies to protein-coding DNA sequences only (for `baseml`). You use option characters GC on the first line instead of G alone. The program will then treat the three codon positions differently in the nucleotide-based analysis. It is assumed that the sequence length is an exact multiple of three.

```
 5  855  GC
human      GTG CTG TCT CCT ...
```

*Option G for codon sequences (codeml with seqtype = 1).* The format is similar to the same option for baseml, but note that the sequence length is in number of nucleotides while the gene lengths are in number of codons. This has been a source of confusion. Below is an example:

```
 5  300  G
G2   40  60
```

This data set has 5 sequences, each of 300 nucleotides (100 codons), which are partitioned into two genes, with the first gene having 40 codons and the second gene 60 codons.

## Site pattern counts

The sequence alignment can also be input in the form of site patterns and counts of sites having those site patterns. This format is specified by the option "P" on the first line of the input data file, as illustrated by the following example. Here there are 3 sequences, 8 site patterns, with "P" indicating that the data are site patterns and not sites. The "P" option is used in the same way as options "I" for interleaved format and "S" for sequential format (default). The 8 numbers below the alignment are the numbers of sites having the 8 patterns above. For example, at 100 sites, all three species has G, and at 200 sites all three species has T, and so on. In total there are 100 + 200 + 40 + … + 14 = 440 sites.

```
   3  8  P

human   GTACTGCC
rabbit  GTACTACT
rat     GTACAGAC

100 200 40 50 11 12 13 14
```

This example applies to baseml and basemlg, program for nucleotide-based analysis. To specify multiple genes (site partitions), one may use option G together with option P.

```
    3  10  PG
G 2     4 6

human   GTTA CATGTC
rabbit  GTCA CATATT
rat     GTTA CAAGTC

100 200 40 50 120 61 12 13 54 12
```

Here there are 10 site patterns and 2 genes (site partitions). The first 4 patterns are for the first gene while the next 6 patterns are for the second gene, with a total of 10 site patterns. In partition 1 there are 40 sites having the data AAA (nucleotide A in all three species), and while in partition 2 there are 61 such sites.

The same format applies to protein sequences (codeml with seqtype = 2), with amino acids replacing nucleotides in the examples above.

For codon sequences (codeml with seqtype = 1), the format is as follows. There are 3 species, and 9 site patterns, with 6 sites having the first site pattern (which has the codon GTG in all three species). Note that 27 = 9*3. The program requires that you use 3 times the number of codon site patterns here. This is strange but consistent with the sequential or interleaved sequence format, where the sequence length is specified in the number of nucleotides rather than number of codons. (Initially I did this so that the same file can be read by both baseml for nucleotide based analysis and codonml for codon based analysis.)

```
    3   27   P G

human     GTG CTG TCT CCT GCC GAC AAG ACC
rabbit    ... ... ... G.C ... ... ... T..
rat       ... ... ... ..C ..T ... ... ...

    6 1 1 1 1 4 3 1 1
```

To specify multiple genes for codon site patterns, see the following example.

```
    3   27   P G
G 2      4    5

human     GTG CTG TCT CCT GCC GAC AAG ACC
rabbit    ... ... ... G.C ... ... ... T..
rat       ... ... ... ..C ..T ... ... ...

    6 1 1 1 1 4 3 1 1
```

Here there are again 9 codon site patterns in total, with the first 4 patterns for gene 1 and the next 5 patterns for gene 2.

Furthermore, option variable P can be used together with option variable I or S. PI means that the site patterns are listed using the interleaved format while PS means that the site patterns are listed using the sequential format. P without I or S uses the default sequential format. Having the whole sequence of all site patterns on one line conforms with both the I and S formats, so there is no need to specify I or S.

If you run baseml and codeml to read the sequential or interleaved formats of sequences, the output will include a print-out in this partitioned format. Look for the line "Printing out site pattern counts". You can move this block into a new file and later on read that file instead, if it takes a long time to pack sites into patterns. Note the restrictions with the P format below.

Here are some restrictions to this option. Some outputs are disabled for this option, including ancestral sequence reconstruction and posterior estimates of rates for sites (or site patterns), that you can get for sequences by using RateAncestor = 1. Second, some of the calculations require the sequence length, which I set to the sum of the site pattern frequencies. If the site pattern frequencies are not counts of sites but are instead site pattern probabilities, calculations involving sequence length will not be correct. Such calculations include the SEs for MLEs, the numbers of sites $S$ and $N$ in codonml, for example.

*Possible uses of this option.* Sometimes I use evolver to simulate very long sequences (with >1M sites) and it can take minutes or hours to collapse sites into patterns, which is irritating when the maximum likelihood iteration takes a few seconds and I want to use the same data to run multiple models. A similar case is analysis of large genomic data of long sequences with >100Mb sites. In this case you can run baseml or codeml once, and then copy the pattern counts from the output file into a data file. Next time, you run the program you can read the new file. This way the program skips the step of counting site patterns. Another situation is to calculate the site pattern probabilities under model and then read the probabilities for analysis using a wrong to see whether the correct tree is still recovered. This way, you can check whether the tree reconstruction method is still consistent. See Debry (1992) and Yang (1994c) for such analysis. (I need to enable the code for printing site pattern probabilities.)

## Tree file format and representations of tree topology

A *tree structure file* is used when `runmode` = 0 or 1. The file name is specified in the appropriate control file. The tree topology is typically specified using the parenthesis notation, although it is possible to use a branch representation, as described below.

*Parenthesis notation:* The first is the familiar parenthesis representation, used in most phylogenetic software. The species can be represented using either their names or their indexes corresponding to the order of their occurrences in the sequence data file. If species names are used, they have to match exactly those in the sequence data file (including spaces or strange characters). Branch lengths are allowed. The following is a possible tree structure file for a data set of four species (human, chimpanzee, gorilla, and orangutan, occurring in this order in the data file). The first tree is a star tree, while the next four trees are the same.

```
    4 5 // 4 species, 5 trees
(1,2,3,4); // the star tree
((1,2),3,4); // species 1 and 2 are clustered together
((1,2),3,4); // Commas are needed with more than 9 species
((human,chimpanzee),gorilla,orangutan);
((human:.1,chimpanzee:.2):.05,gorilla:.3,orangutan:.5);
```

If the tree has branch lengths, baseml and codeml allow you to use the branch lengths in the tree as starting values for maximum likelihood iteration.

Whether you should use rooted or unrooted trees depends on the model, for example, on whether a molecular clock is assumed. Without the clock (clock = 0), unrooted trees should be used, such as ((1,2),3,4) or (1,2,(3,4)). With the clock or local-clock models, the trees should be rooted and these two trees are different and both are different from (((1,2),3),4). In PAML, a rooted tree has a bifurcation at the root, while an unrooted tree has a trifurcation or multifurcation at the root.

*Tree files produced by PAUP and MacClade.* PAML programs have only limited compatibility with the tree file generated by PAUP or MacClade. First the "[&U]" notation for specifying an unrooted tree is ignored. For the tree to be accepted as an unrooted tree by PAML, you have to manually modify the tree file so that there is a trifurcation at the root, for example, by changing "(((1,2),3),4)" into "((1,2),3,4)". Second, the "Translate" keyword is ignored by PAML as well, and it is assumed

that the ordering of the sequences in the tree file is exactly the same as the ordering of the sequences in the sequence data file.

*Branch or node labels.* Some models implemented in `baseml` and `codeml` allow several groups of branches on the tree, which are assigned different parameters of interest. For example, in the local clock models (`clock = 2` or 3) in baseml or codeml, you can have, say, 3 branch rate groups, with low, medium, and high rates respectively. Also the branch-specific codon models (`model = 2` or 3 for `codonml`) allow different branch groups to have different $\omega$s, leading to so called "two-ratios" and "three-ratios" models. All those models require branches or nodes in the tree to be labeled. Branch *labels* are specified in the same way as branch *lengths* except that the symbol "#" is used rather than ":". The branch labels are consecutive integers starting from 0, which is the default and does not have to be specified. For example, the following tree

```
((Hsa_Human, Hla_gibbon) #1, ((Cgu/Can_colobus, Pne_langur), Mmu_rhesus), (Ssc_squirrelM, Cja_marmoset));
```

is from the tree file `examples/lysozyme/lysozyme.trees`, with a branch label for fitting models of different $\omega$ ratios for branches. The internal branch ancestral to human and gibbon has the ratio $\omega_1$, while all other branches (with the default label #0) have the background ratio $\omega_0$. This fits the model in table 1C for the small data set of lysozyme genes in Yang (1998). See the readme file in the `examples/lysozyme/` folder.

On a big tree, you might want to label all branches within a clade. For this purpose, you can use the clade label $. $ is for $\Delta$, which looks like a good clade symbol but is missing on most keyboards. So (clade) $2 is equivalent to labeling all nodes/branches within the clade with #2. The following two trees are thus equivalent.

```
(((rabbit, rat) $1, human), goat_cow, marsupial);
(((rabbit #1, rat #1) #1, human), goat_cow, marsupial);
```

Here are the rules concerning nested clade labels. The symbol # takes precedence over the symbol $, and clade labels close to the tips take precedence over clade labels for ancestral nodes close to the root. So the following two trees are equivalent. In the first tree below, $1 is first applied to the whole clade of placental mammals (except for the human lineage), and then $2 is applied to the rabbit-rat clade.

```
((((rabbit, rat) $2, human #3), goat_cow) $1, marsupial);
((((rabbit #2, rat #2) #2, human #3) #1, goat_cow #1) #1, marsupial);
```

Note that with this rule, it may make a difference whether or not you include a label $0. For example

```
((a, b) $0, (c, d)) $1;
```

labels the three branches on the left side (to a, to b, and to both a and b) as #0, while the other branches are #1. However the following would label all branches in the tree as #1.

```
((a, b), (c, d)) $1;
```

I have found it convenient to create the tree file with labels and read the tree using Rod page's (1996) [TreeView](#) to check that the tree and labels are right. The example trees above should be readable by TreeView. For TreeView X, however, you may have to put the labels inside single quotation marks, like the following.

```
((((rabbit '#2', rat '#2') '#2', human '#3') '#1', goat_cow '#1') '#1', marsupial);
```

This way, the tree is readable by both TreeView and TreeView X (and by baseml/codeml as well). Note that TreeView and TreeView X do not accept labels for tips or tip branches, and may interpret

the labels as part of the sequence name.  Another program that you can use to create and/or view branch or node labels is Andrew Rambaut's TreeEdit, available for the MAC.  I have no experiencing of using it.

*Divergence date symbol* @.  Fossil calibration information is specified using the symbol @.  This is used for the clock and local clock models in baseml and codeml.  See the readme file in the examples/MouseLemurs/ folder.  So in the following example, the human-chimpanzee divergence is fixed at 5MY.

```
((gorilla, (human, chimpanzee) '@0.05'), orangutan);
```

The Bayesian dating program `mcmctree` calibration information.  See descriptions later.

**Branch representation of tree topology:**  A second way of representing the tree topology used in PAML is by enumerating its branches, each of which is represented by its starting and ending nodes. This representation is also used in the result files for outputting the estimated branch lengths, but you can also use it in the tree file. For example, the tree ((1,2),3,4) can be specified by enumerating its 5 branches:

```
   5
 5 6    6 1    6 2    5 3    5 4
```

The nodes in the tree are indexed by consecutive natural numbers, with 1, 2, ..., *s* representing the *s* known sequences in the data, in the same order as in the data.  A number larger than *s* labels an internal node, at which the sequence is unknown.  So in the above tree, node 5 is ancestral to nodes 6, 3, and 4, while node 6 is ancestral to nodes 1 and 2.

This notation is convenient to specify a tree in which some sequences in the data are direct ancestors to some others.  For example, the following tree for 5 sequences has 4 branches, with sequence 5 to be the common ancestor of sequences 1, 2, 3, and 4:

```
   4
 5 1    5 2    5 3    5 4
```

☠*Warning.*  I did not try to make this tree representation work with all models implemented in baseml and codeml.  If you use this representation, you should test the program carefully.  If it does not work, you can let me know so that I will try to fix it.

# 4 baseml

## Nucleotide substitution models

For detailed descriptions of Markov models of nucleotide substitution, see Whelan *et al.* (2001), Felsenstein (2004) or Yang (2006: Chapter 1).

Models used in PAML include JC69 (Jukes and Cantor 1969), K80 (Kimura 1980), F81 (Felsenstein 1981), F84 (Felsenstein, DNAML program since 1984, PHYLIP Version 2.6), HKY85 (Hasegawa et al. 1984; Hasegawa et al. 1985), Tamura (1992), Tamura and Nei (1993), and REV, also known as GTR for general-time-reversible (Yang 1994b; Zharkikh 1994). The rate matrices are parametrized as follows.

JC69 :
$$Q = \begin{pmatrix} . & 1 & 1 & 1 \\ 1 & . & 1 & 1 \\ 1 & 1 & . & 1 \\ 1 & 1 & 1 & . \end{pmatrix}$$

K80 :
$$Q = \begin{pmatrix} . & \kappa & 1 & 1 \\ \kappa & . & 1 & 1 \\ 1 & 1 & . & \kappa \\ 1 & 1 & \kappa & . \end{pmatrix}$$

F81 :
$$Q = \begin{pmatrix} . & \pi_C & \pi_A & \pi_G \\ \pi_T & . & \pi_A & \pi_G \\ \pi_T & \pi_C & . & \pi_G \\ \pi_T & \pi_C & \pi_A & . \end{pmatrix}$$

F84:
$$Q = \begin{pmatrix} . & (1+\kappa/\pi_Y)\pi_C & \pi_A & \pi_G \\ (1+\kappa/\pi_Y)\pi_T & . & \pi_A & \pi_G \\ \pi_T & \pi_C & . & (1+\kappa/\pi_R)\pi_G \\ \pi_T & \pi_C & (1+\kappa/\pi_R)\pi_A & . \end{pmatrix}$$

with $\pi_Y = \pi_T + \pi_C$ and $\pi_R = \pi_A + \pi_G$.

HKY85:
$$Q = \begin{pmatrix} . & \kappa\pi_C & \pi_A & \pi_G \\ \kappa\pi_T & . & \pi_A & \pi_G \\ \pi_T & \pi_C & . & \kappa\pi_G \\ \pi_T & \pi_C & \kappa\pi_A & . \end{pmatrix}$$

$$
\text{T92:} \qquad Q = \begin{pmatrix}
. & \kappa(1-\pi_{GC})/2 & \pi_{GC}/2 & \pi_{GC}/2 \\
\kappa(1-\pi_{GC})/2 & . & \pi_{GC}/2 & \pi_{GC}/2 \\
(1-\pi_{GC})/2 & (1-\pi_{GC})/2 & . & \kappa\pi_{GC}/2 \\
(1-\pi_{GC})/2 & (1-\pi_{GC})/2 & \kappa\pi_{GC}/2 & .
\end{pmatrix}
$$

$$
\text{TN93:} \qquad Q = \begin{pmatrix}
. & \kappa_1\pi_C & \pi_A & \pi_G \\
\kappa_1\pi_T & . & \pi_A & \pi_G \\
\pi_T & \pi_C & . & \kappa_2\pi_G \\
\pi_T & \pi_C & \kappa_2\pi_A & .
\end{pmatrix}
$$

$$
\text{REV (GTR):} \qquad Q = \begin{pmatrix}
. & a\pi_C & b\pi_A & c\pi_G \\
a\pi_T & . & d\pi_A & e\pi_G \\
b\pi_T & d\pi_C & . & \pi_G \\
c\pi_T & e\pi_C & \pi_A & .
\end{pmatrix}
$$

$$
\text{UNREST} \qquad Q = \begin{pmatrix}
. & q_{TC} & q_{TA} & q_{TG} \\
q_{CT} & . & q_{CA} & q_{CG} \\
q_{AT} & q_{AC} & . & q_{AG} \\
q_{GT} & q_{GC} & q_{GA} & .
\end{pmatrix} = \begin{pmatrix}
. & a & b & c \\
d & . & e & f \\
g & h & . & i \\
j & k & l & .
\end{pmatrix}.
$$

## The control file

The default control file for baseml is baseml.ctl, and an example is shown below. Note that spaces are required on both sides of the equal sign, and blank lines or lines beginning with "*" are treated as comments. Options not used can be deleted from the control file. The order of the variables is unimportant.

```
      seqfile = brown.nuc * sequence data file name
      outfile = mlb * main result file
     treefile = brown.trees * tree structure file name

        noisy = 3    * 0,1,2,3: how much rubbish on the screen
      verbose = 0    * 1: detailed output, 0: concise output
      runmode = 0    * 0: user tree;  1: semi-automatic;  2: automatic
                     * 3: StepwiseAddition; (4,5):PerturbationNNI

        model = 5    * 0:JC69, 1:K80, 2:F81, 3:F84, 4:HKY85
                     * 5:T92, 6:TN93, 7:REV, 8:UNREST, 9:REVu; 10:UNRESTu
        Mgene = 0    * 0:rates, 1:separate; 2:diff pi, 3:diff kapa, 4:all diff

*       ndata = 1    * number of data sets
        clock = 0    * 0:no clock, 1:clock; 2:local clock; 3:CombinedAnalysis
    fix_kappa = 0  * 0: estimate kappa; 1: fix kappa at value below
        kappa = 2.5   * initial or fixed kappa

    fix_alpha = 1    * 0: estimate alpha; 1: fix alpha at value below
        alpha = 0.   * initial or fixed alpha, 0:infinity (constant rate)
       Malpha = 0    * 1: different alpha's for genes, 0: one alpha
        ncatG = 5    * # of categories in the dG, AdG, or nparK models of rates

      fix_rho = 1    * 0: estimate rho; 1: fix rho at value below
          rho = 0.   * initial or fixed rho,   0:no correlation
        nparK = 0    * rate-class models. 1:rK, 2:rK&fK, 3:rK&MK(1/K), 4:rK&MK

        nhomo = 0    * 0 & 1: homogeneous, 2: kappa for branches, 3: N1, 4: N2
        getSE = 0    * 0: don't want them, 1: want S.E.s of estimates
  RateAncestor = 0   * (0,1,2): rates (alpha>0) or ancestral states
```

```
   Small_Diff = 1e-6
*    cleandata = 1    * remove sites with ambiguity data (1:yes, 0:no)?
*        icode = 0    * (RateAncestor=1 for coding genes, "GC" in data)
* fix_blength = 0    * 0: ignore, -1: random, 1: initial, 2: fixed
       method = 0    * 0: simultaneous; 1: one branch at a time
```

The control variables are described below.

**seqfile**, **outfile**, and **treefile** specifies the names of the sequence data file, main result file, and the tree structure file, respectively. You should not have spaces inside a file name. In general try to avoid special characters in a file name as they might have special meanings under the OS.

**noisy** controls how much output you want on the screen. If the model being fitted involves much computation, you can choose a large number for noisy to avoid loneliness. verbose controls how much output in the result file.

**runmode** = 0 means evaluation of the tree topologies specified in the tree structure file, and runmode = 1 or 2 means heuristic tree search by the star-decomposition algorithm. With runmode = 2, the algorithm starts from the star tree, while if runmode = 1, the program will read a multifurcating tree from the tree structure file and try to estimate the best bifurcating tree compatible with it. runmode = 3 means stepwise addition. runmode = 4 means NNI perturbation with the starting tree obtained by a parsimony algorithm, while runmode = 5 means NNI perturbation with the starting tree read from the tree structure file. The tree search options do not work well, and so use runmode = 0 as much as you can. For relatively small data set, the stepwise addition algorithm seems usable.

**model** specifies the model of nucleotide substitution. Models 0, 1, …, 8 represent models JC69, K80, F81, F84, HKY85, T92, TN93, REV (also known as GTR), and UNREST, respectively. Check Yang (1994 JME 39:105-111) for notation. Two more models are implemented recently. model = 9 are special cases of the REV model, while model = 10 are special cases of the unrestricted model. The format is shown in the following examples and should be self-explanatory. Basically you include extra information on the same line that specifies the model when model = 9 or 10. The number in the brackets [] are the number of free rate parameters. For example, this should be 5 for REV and 11 for UNREST. Following that number are equal number of parenthesis pairs (). The rate parameters in the output file will follow this order here. The pairs that are not mentioned will have rate 1. When model = 9, you specify TC or CT, but not both. When model = 10, TC and CT are different. See the following examples and Yang (1994a) for notation.

```
model = 10  [0]  /* JC69 */
model = 10  [1 (TC CT AG GA)]  /* K80 */
model = 10  [11 (TA) (TG) (CT) (CA) (CG) (AT) (AC) (AG) (GT) (GC) (GA) ]  /* unrest */
model = 10  [5 (AC CA) (AG GA) (AT TA) (CG GC) (CT TC)]  /* SYM */
model = 9   [2 (TA TG CA CG) (AG)]    /* TN93 */
```

**Mgene** is used in combination with option G in the sequence data file, for combined analysis of data from multiple genes or multiple site partitions (such as the three codon positions) (Yang 1996a; pages 116-8 in Yang 2006). Such models are also called partition models. Choose 0 if option G is not used in the data file.

The description here refers to multiple genes, but apply to any strategy of partitioning sites, such as by codon position. Similar partition models are implemented in codeml for analyzing codon or amino acid sequences.

The models are summarized in table 1. The simplest model which assumes complete homogeneity among genes can be fitted by concatenating different genes into one sequence without using the option G (and by specifying `Mgene` = 0 in the control file). The most general model is equavilent to a separate analysis, and can be done by fitting the same model to each data set (each gene), but can also be done by specifying `Mgene` = 1 with the option G in the combined data file. The sum of the log-likelihood values over different genes is then the log likelihood of the most general model considered here. Models accounting for some aspects of the heterogeneity of multiple genes are fitted by specifying `Mgene` in combination with the option G in the sequence data file. `Mgene` = 0 means a model that asumes different substitution rates but the same pattern of nucleotide substitution for different genes. `Mgene` = 2 means different frequency parameters for different genes but the same rate ratio parameters ($\kappa$ in the K80, F84, HKY85 models or the rate parameters in the TN93 and REV models). `Mgene` = 3 means different rate ratio parameters and the same frequency parameters. `Mgene` = 4 means both different rate ratio parameters and different frequency parameters for different genes. Parameters and assumptions made in these models are summarized in the following table, with the HKY85 model used as an example. When substitution rates are assumed to vary from site to site, the control variable `Malpha` specifies whether one gamma distribution will be applied across all sites (`Malpha` = 0) or a different gamma distribution is used for each gene (or codon position).

Table 1. Setups of partition models of nucleotide substitution

| Sequence file | Control file | Parameters across genes |
|---|---|---|
| No G | `Mgene` = 0 | everything equal |
| Option G | `Mgene` = 0 | the same $\kappa$ and $\pi$, but different $c$s (proportional branch lengths) |
| Option G | `Mgene` = 2 | the same $\kappa$, but different $\pi$s and $c$s |
| Option G | `Mgene` = 3 | the same $\pi$, but different $\kappa$s and $c$s |
| Option G | `Mgene` = 4 | different $\kappa$, $\pi$s, and $c$s |
| Option G | `Mgene` = 1 | different $\kappa$, $\pi$s, and different (unproportional) branch lengths |

The different $c$s for different genes mean that branch lengths estimated for different genes are proportional. Parameters $\pi$ represent the equilibrium nucleotide frequencies, which are estimated using the observed frequencies (`nhomo` = 0). The transition/transversion rate ratio $\kappa$ in HKY85 can be replaced by the two or five rate ratio parameters under the TN93 or REV models, respectively. The likelihood ratio test can be used to compare these models, using an approach called the analysis of deviance (McCullagh and Nelder 1989), which is very similar to the more familiar analysis of variance.

**ndata**: specifies the number of separate data sets in the file. This variable is useful for simulation. You can use `evolver` to generate 200 replicate data sets, and then set `ndata` = 200 to use `baseml` to analyze them.

**clock** specifies models concerning rate constancy or variation among lineages. `clock` = 0 means no clock and rates are entirely free to vary from branch to branch. An unrooted tree should be used under this model. For a binary tree with $n$ species (sequences), this model has ($2n$ – 3) parameters (branch lengths). `clock` = 1 means the global clock, and all branches have the same rate. This model has ($n$ – 1) parameters corresponding to the ($n$ – 1) internal nodes in the binary tree. So a test of the molecular clock assumption, which compares those two models, should have d.f. = $n$ – 2.

clock = 2 specifies the local clock models(Yoder and Yang 2000; Yang and Yoder 2003). Most branches in the phylogeny conform with the clock assumption and has the default rate ($r_0 = 1$), but some pre-defined branches may have different rates. Rates for branches are specified using branch labels (# and $) in the tree file. For example, the tree (((1,2) #1, 3), 4) specifies rate $r_1$ for the branch ancestral to species 1 and 2 while all other branches have the default rate $r_0$, which does not have to be specified. The user need to specify which branch has which rate, and the program estimates the unknown rates (such as $r_1$ in the above example; $r_0 = 1$ is the default rate). You need to be careful when specifying rates for branches to make sure that all rates can be estimated by the model; if you specify too many rate parameters, especially for branches around the root, it may not be possible to estimate all of them and you will have a problem with identifiability. The number of parameters for a binary tree in the local clock model is ($n - 1$) plus the number of extra rate parameters for branches. In the above tree of 4 species, you have only one extra rate parameter $r_1$, and so the local clock model has ($n - 1$) + 1 = $n$ = 4 parameters. The no-clock model has 5 parameters while the global clock has 3 parameters for that tree.

The option clock = 3 is for combined analysis of multiple-gene or multiple-partition data, allowing the branch rates to vary in different ways among the data partitions (Yang and Yoder 2003). To account for differences in the evolutionary process among data partitions, you have to use the option G in the sequence file as well as the control variable Mgene in the control file (baseml.ctl or codeml.ctl). Read Yang and Yoder (2003) and the readme file in the examples/MouseLemurs/ folder to duplicate the analysis of that paper. Also the variable (= 5 or 6) is used to implement the ad hoc rate smoothing procedure of Yang (2004). See the file readme2.txt for instructions and the paper for details of the model.

For clock = 1, 2, or 3, a rooted tree should be used. If fossil calibration information is specified in the tree file using the symbol @ or =, the absolute rate will be calculated. Multiple calibration points can be specified this way, but only point calibrations (where a node age is assumed to be known without error) are accepted and bounds are not accepted. See instructions about the mcmctree program, which accepts bounds or other distributions as calibrations. If sequences have dates, this option will fit Andrew Rambaut's TipDate model.

**fix_kappa** specifies whether κ in K80, F84, or HKY85 is given at a fixed value or is to be estimated by iteration from the data. If fix_kappa = 1, the value of another variable, kappa, is the given value, and otherwise the value of kappa is used as the initial estimate for iteration. The variables fix_kappa and kappa have no effect with JC69 or F81 which does not involve such a parameter, or with TN93 and REV which have two and five rate parameters respectively, when all of them are estimated from the data.

**fix_alpha** and **alpha** work in a similar way, where alpha refers to the shape parameter α of the gamma distribution for variable substitution rates across sites (Yang 1994a). The model of a single rate for all sites is specified as fix_alpha = 1 and alpha = 0 (0 means infinity), while the (discrete-) gamma model is specified by a positive value for alpha, and ncatG is then the number of categories for the discrete-gamma model (baseml). Values such as 5, 4, 8, or 10 are reasonable. Note that the discrete gamma model has one parameter ($\alpha$), like the continuous gamma model, and the number of categories is not a parameter.

The mdel of invariable sites is not implemented in PAML programs. I don't like the model as it generates a strong correlation between the proportion of invariable sites and the gamma shape parameter. It is implemented in PAUP and MrBayes, for example.

To infer rates at individual sites, use RateAncestor = 1. See below. Using a large number of categories (say, ncatG = 40) may be helpful if you are interested in calculating such rates.

For detailed descriptions of those models, see Yang (1996b), Chapter 1 of Yang (2006) and chapters 13 and 16 of Felsenstein (2004).

**fix_rho** and rho work in a similar way and concern independence or correlation of rates at adjacent sites, where $\rho$ (rho) is the correlation parameter of the auto-discrete-gamma model (Yang 1995). The model of independent rates for sites is specified as fix_rho = 1 and rho = 0; choosing alpha = 0 further means a constant rate for all sites. The auto-discrete-gamma model is specified by positive values for both alpha and rho. The model of a constant rate for sites is a special case of the (discrete) gamma model with $\alpha = \infty$ (alpha = 0), and the model of independent rates for sites is a special case of the auto-discrete-gamma model with $\rho = 0$ (rho = 0).

**nparK** specifies nonparametric models for variable and Markov-dependent rates across sites: nparK = 1 or 2 means several (ncatG) categories of independent rates for sites, while nparK = 3 or 4 means the rates are Markov-dependent at adjacent sites; nparK = 1 and 3 have the restriction that each rate category has equal probability while nparK = 2 and 4 do not have this restriction (Yang and Roberts 1995). The variable nparK takes precedence over alpha or rho.

**nhomo** is for baseml only, and concerns the frequency parameters in some of the substitution models. The option nhomo = 1 fits a homogeneous model, but estimates the frequency parameters ($\pi_T$, $\pi_C$ and $\pi_A$; $\pi_G$ is not a free parameter as the frequencies sum to 1) by maximum likelihood iteration. This applies to F81, F84, HKY85, T92 (in which case only $\pi_{GC}$ is a parameter), TN93, or REV models. Normally (nhomo = 0) these are estimated by the averages of the observed frequencies. In both cases (nhomo = 0 and 1), you should count 3 (or 1 for T92) free parameters for the base frequencies.

Options nhomo = 3, 4, and 5 work with F84, HKY85, or T92 only. They fit the nonhomogeneous models of Yang and Roberts (1995) and Galtier and Gouy (1998). The nucleotide substitution is specified by the variable model and is one of F84, HKY85 or T92, but with different frequency parameters used in the rate matrix for different branches in the tree, to allow for unequal base frequencies in different sequences. The position of the root then makes a difference to the likelihood, and rooted trees are used. Because of the parameter richness, the model may only be used with small trees except that you have extremely long sequences. Yang and Roberts (1995) used the HKY85 or F84 models, and so three independent frequency parameters are used to describe the substitution pattern, while Galtier and Gouy (1998) used the T92 substitution model and uses the GC content $\pi_{GC}$ only, with the base frequencies give as $\pi_T = \pi_A = (1 - \pi_{GC})/2$ and $\pi_C = \pi_G = \pi_{GC}/2$. The option nhomo = 4 assigns one set of frequency parameters for the root, which are the initial base frequencies at the root, and one set for each branch in the tree. This is model N2 in Yang and Roberts (1995) if the substitution model is F84 or HKY85 or the model of Galtier and Gouy (1998) if the substitution model is T92. Option nhomo = 3 uses one set of base frequencies for each tip branch, one set for all internal branches in the tree, and one set for the root. This specifies model N1 in Yang and Roberts (1995).

The option nhomo = 5 lets the user specify how many sets of frequency parameters should be used and which node (branch) should use which set. The set for the root specifies the initial base frequencies at the root while the set for any other node is for parameters in the

substitution matrix along the branch leading to the node. You use branch (node) labels in the tree file (see the subsection "Tree file and representations of tree topology" above) to tell the program which set each branch should use. There is no need to specify the default set (0). So for example `nhomo` = 5 and the following tree in the tree file species sets 1, 2, 3, 4, and 5 for the tip branches, set 6 for the root, while all the internal branches (nodes) will have the default set 0. This is equivalent to `nhomo` = 3.

        ((((1 #1, 2: #2), 3 #3), 4 #4), 5 #5) #6;

The output for `nhomo` = 3, 4, 5 is under the heading "base frequency parameters (4 sets) for branches, and frequencies at nodes". Two sets of frequencies are listed for each node. The first set are the parameters (used in the substitution rate matrix for the branch leading to the node), and the second set are the expected base frequencies at the node, calculated from the model ((Yang and Roberts 1995); page 456 column top). If the node is the root, the same set of frequencies are printed twice.

Note that the use of the variable `fix_kappa` here with `nhomo` = 3, 4 or 5 is unusual. `fix_kappa` = 1 means one common $\kappa$ is assumed and estimated for all branches, while `fix_kappa` = 0 means one $\kappa$ is estimated for each branch.

`nhomo` = 2 uses one transition/transversion rate ratio ($\kappa$) for each branch in the tree for the K80, F84, and HKY85 models (Yang 1994b; Yang and Yoder 1999).

**getSE** = 0, 1, or 2 tells whether we want estimates of the standard errors of estimated parameters. These are crude estimates, calculated by the curvature method, *i.e.*, by inverting the matrix of second derivatives of the log-likelihood with respect to parameters. The second derivatives are calculated by the difference method, and are not always reliable. Even if this approximation is reliable, tests relying on the SE's should be taken with caution, as such tests rely on the normal approximation to the maximum likelihood estimates. The likelihood ratio test should always be preferred. The option is not available and choose `getSE = 0` when tree-search is performed.

**RateAncestor** = 0 or 1. Usually use 0. The value 1 forces the program to do two additional analyses, which you can ignore if you don't need the results. First under a model of variable rates across sites such as the gamma, `RateAncestor` = 1 forces the program to calculate rates for individual sites along the sequence (output in the file `rates`), using the empirical Bayes procedure (Yang and Wang 1995).

Second `RateAncestor` = 1 forces the program to perform the empirical Bayesian reconstruction of ancestral sequences (Yang et al. 1995a; Koshi and Goldstein 1996; Yang 2006 pages 119-124). Ancestral state reconstruction by parsimony (Fitch 1971; Hartigan 1973) is well known (implemented in the program pamp in PAML). It can also be achieved using the likelihood/empirical Bayes approach. Often the two approaches produce similar results, but the likelihood-based reconstruction has two advantages over parsimony: it uses information from the branch lengths and the relative substitution rates between characters (nucleotides), and it provides a measure of uncertainties in the form of posterior probabilities for reconstructed ancestral states.

The outputs are listed, by site, in the file `rst`. You can also use the variable `verbose` to control the amount of output. If `verbose` = 0, only the best nucleotide (the one with the highest posterior proobability) at each node at each site is listed, while with `verbose` = 1 (try 2 if 1 does not work), the full posterior probability distribution from the marginal reconstruction is listed. If the model assumes one rate for all sites, both the joint and

marginal ancestral reconstructions will be calculated. If the model assumes variable rates among sites like the gamma model, only the marginal reconstructions are calculated.

*Marginal* and *joint* reconstructions. Marginal reconstruction considers character assignments to one single interior node and the character with the highest posterior probability is the best reconstruction (eq. 4 in Yang et al. 1995a; or Eq. 4.15 in Yang 2006). The algorithm for marginal reconstruction implemented in PAML works under both the model of a constant rate for all sites and the gamma model of rates at sites. Joint reconstruction considers all ancestral nodes at the same time and the reconstruction (the set of characters at a site assigned to all interior nodes) with the highest posterior probability is the best reconstruction (eq. 2 in Yang et al. 1995a; or Eq. 4.16 in Yang 2006). The algorithm for joint reconstruction implemented in PAML is based on that of Pupko *et al.* (2000), which gives the best reconstruction at each site and its posterior probability. This works under the model of one rate for all sites only. (It works under the partition models.) The marginal and joint approaches use slightly different criteria, and expected to produce consistent results; that is, the most probable joint reconstruction for a site should almost always consist of characters that are also the best in the marginal reconstruction. Conflicts may arise in borderline cases where two or more reconstructions have similar posterior probabilities.

A good use of ancestral sequence reconstruction is to synthesize the inferred ancestral proteins and measure their biochemical properties in the lab (Pauling and Zuckerkandl 1963; Chang and Donoghue 2000; Thornton 2004). It is also very popular to use reconstructed ancestral sequences as if they were real observed data to perform further analysis. You should resist this irresistible temptation and use full likelihood methods if they are available (e.g., Yang 2002). See section 4.4.4 in Yang (2006) for a discussion of systematic biases in ancestral reconstruction.

For nucleotide based (`baseml`) analysis of protein coding DNA sequences (option GC in the sequence data file), the program also calculates the posterior probabilities of ancestral amino acids. In this analysis, branch lengths and other parameters are estimated under a nucleotide substitution model, but the reconstructed nucleotide triplets are treated as a codon to infer the most likely amino acid encoded. Posterior probabilities for stop codons are small and reset to zero to scale the posterior probabilities for amino acids. To use this option, you should add the control variable `icode` in the control file `baseml.ctl`. This is not listed in the above. The variable `icode` can take a value out of 0, 1, ..., 11, corresponding to the 12 genetic codes included in PAML (See the control file `codeml.ctl` for the definition of different genetic codes). A nucleotide substitution model that is very close to a codon-substitution model can be specified as follows. You add the option characters GC at the end of the first line in the data file and choose model = 4 (HKY85) and `Mgene` = 4. The model then assumes different substitution rates, different base frequencies, and different transition/transversion rate ratio (kappa) for the three codon positions. Ancestral reconstruction from such a nucleotide substitution should be very similar to codon-based reconstruction.

**Small_Diff** is a small value used in the difference approximation of derivatives.

**cleandata** = 1 means sites involving ambiguity characters (undetermined nucleotides such as N, ?, W, R, Y, etc. anything other than the four nucleotides) or alignment gaps are removed from all sequences. This leads to faster calculation. cleaddata = 0 (default) uses those sites.

**method:** This variable controls the iteration algorithm for estimating branch lengths under a model of no clock. method = 0 implements the old algorithm in PAML, which updates all

parameters including branch lengths simultaneously. `method` = 1 specifies an algorithm newly implemented in PAML, which updates branch lengths one by one. `method` = 1 does not work under the clock models (`clock` = 1, 2, 3).

**icode**: This specifies the genetic code to be used for ancestral reconstruction of protein-coding DNA sequences. This is implemented to compare results of ancestral reconstruction with codon-based analysis. For example the F3×4 codon model of Goldman and Yang (1994) is very similar to the nucleotide model HKY85 with different substitution rates and base frequencies for the three codon positions. The latter is implemented by using use options GC in the sequence data file and `model` = 4 and `Mgene` = 4. To use the option `icode`, you have to choose `RateAncestor` = 1.

**fix_blength**: This tells the program what to do if the tree has branch lengths. Use 0 if you want to ignore the branch lengths. Use –1 if you want the program to start from random starting points. This might be useful if there are multiple local optima. Use 1 if you want to use the branch lengths as initial values for the ML iteration. Try to avoid using the "branch lengths" from a parsimony analysis from PAUP, as those are numbers of changes for the entire sequence (rather than per site) and are very poor initial values. Use 2 if you want the branch lengths to be fixed at those given in the tree file (rather than estimating them by ML). In this case, you should make sure that the branch lengths are sensible; for example, if two sequences in the data file are different, but the branch lengths connecting the two sequences in the tree are all zero, the data and tree will be in conflict and the program will crash.

**Output**: The output should be self-explanatory. Descriptive statistics are always listed. The observed site patterns and their frequencies are listed, together with the proportions of constant patterns. Nucleotide frequencies for each species (and for each gene in case of multiple gene data) are counted and listed. lmax = ln(Lmax) is the upper limit of the log likelihood and may be compared with the likelihood for the best (or true) tree under the substitution model to test the model's goodness of fit to data (Goldman 1993; Yang et al. 1995b). You can ignore it if you don't know what it means. The pairwise sequence distances are included in the output as well, and also in a separate file called `2base.t`. This is a lower-diagonal distance matrice, readable by the `NEIGHBOR` program in Felesenstein's PHYLIP package (Felsenstein 2005). For models JC69, K80, F81, F84, the appropriate distance formulas are used, while for more complex models, the TN93 formula is used. baseml is mainly a maximum likelihood program, and the distance matrix is printed out for convenience and really has nothing to do with the later likelihood calculation.

With getSE = 1, the S.E.s are calculated as the square roots of the large sample variances and listed exactly below the parameter estimates. Zeros on this line mean errors, either caused by divergence of the algorithm or zero branch lengths. The S.Es of the common parameters measure the reliability of the estimates. For example, $(\kappa - 1)/SE(\kappa)$, when $\kappa$ is estimated under K80, can be compared with a normal distribution to see whether there is real difference between K80 and JC69. The test can be more reliably performed by comparing the log-likelihood values under the two models, using the likelihood ratio test. It has to be stressed that the S.E.'s of the estimated branch lengths should not be misinterpreted as an evaluation of the reliability of the estimated tree topology (Yang 1994c).

If the tree file has more than one tree, the programs `baseml` and `codeml` will calculate the bootstrap proportions using the RELL method (Kishino and Hasegawa 1989), as well as the method of Shimodaira and Hasegawa (1999) with a correction for multiple comparison. The bootstrap resampling accounts for possible data partitions (option G in the sequence data file).

# 5  basemlg

`basemlg` uses the same control file `baseml.ctl`, as `baseml.` Tree-search or the assumption of a molecular clock are not allowed and so choose `runmode` = 0 and `clock` = 0. Substitution models available for `basemlg` are JC69, F81, K80, F84 and HKY85, and a continuous gamma is always assumed for rates at sites. The variables `ncatG`, `given_rho`, `rho`, `nhomo`  have no effect.  The S.E.'s of parameter estimates are always printed out because they are calculated during the iteration, and so getSE has no effect.

Because of the intensive computation required by `basemlg`, the discrete-gamma model implemented in `baseml` is recommended for data analysis. If you choose to use `basemlg`, you should run `baseml` first, and then run `basemlg.` This allows `baseml` to collect initial values into a file named `in.basemlg`, for use by `basemlg`. Note that `basemlg` implements only a subset of models in `baseml`.

# 6  codeml (codonml and aaml)

## Codon substitution models

There is now a large collection of codon substitution models.  See Yang and Bielawski (2000), Yang (2002) and Yang (2006: Chapter 8) for detailed discussions.

The basic model in common use is a simplified version of the model of Goldman and Yang (1994) and specifies the substitution rate from codon *i* to codon *j* as

$$
q_{ij} = \begin{cases}
0, & \text{if the two codons differ at more than one position,} \\
\pi_j, & \text{for synonymous transversion,} \\
\kappa\pi_j, & \text{for synonymous transition,} \\
\omega\pi_j, & \text{for nonsynonymous transversion,} \\
\omega\kappa\pi_j, & \text{for nonsynonymous transition,}
\end{cases}
$$

(Yang et al. 1998).  The equilibrium frequency of codon *j* ($\pi_j$) can be considered a free parameter, but can also be calculated from the nucleotide frequencies at the three codon positions (control variable CodonFreq).  Under this model, the relationship holds that $\omega = d_N/d_S$, the ratio of nonsynonymous/synonymous substitution rates.  This basic model is fitted by specifying model = 0 NSsites = 0, in the control file codeml.ctl.

The $\omega$ ratio is a measure of natural selection acting on the protein.  Simplistically, values of $\omega < 1$, = 1, and > 1 means negative purifying selection, neutral evolution, and positive selection.  However, the ratio averaged over all sites and all lineages is almost never > 1, since positive selection is unlikely to affect all sites over prolonged time.  Thus interest has been focused on detecting positive selection that affects only some lineages or some sites.

The *branch model*s allow the $\omega$ ratio to vary among branches in the phylogeny and are useful for detecting positive selection acting on particular lineages (Yang 1998; Yang and Nielsen 1998). They are specified using the variable model.  model = 1 fits the so-called *free-ratios* model, which assumes an independent $\omega$ ratio for each branch.  This model is very parameter-rich and its use is discouraged.  model = 2 allows you to have several $\omega$ ratios.  You have to specify how many ratios and which branches should have which rates in the tree file by using branch labels.  See "Branch or node labels" in the section "Tree file format" in Chapter 4.  The lysozyme example data files are included in the examples/lysozyme/ folder; check the readme file.

Table 2. Parameters in the site models

| Model | NSsites | #p | Parameters |
|---|---|---|---|
| M0 (one ratio) | 0 | 1 | $\omega$ |
| M1a (neutral) | 1 | 2 | $p_0$ ($p_1 = 1 - p_0$), |
| | | | $\omega_0 < 1$, $\omega_1 = 1$ |
| M2a (selection) | 2 | 4 | $p_0, p_1$ ($p_2 = 1 - p_0 - p_1$), |
| | | | $\omega_0 < 1$, $\omega_1 = 1$, $\omega_2 > 1$ |
| M3 (discrete) | 3 | 5 | $p_0, p_1$ ($p_2 = 1 - p_0 - p_1$) |
| | | | $\omega_0, \omega_1, \omega_2$ |
| M7 (beta) | 7 | 2 | $p, q$ |
| M8 (beta&$\omega$) | 8 | 4 | $p_0$ ($p_1 = 1 - p_0$), |
| | | | $p, q, \omega_s > 1$ |

NOTE.— #p is the number of free parameters in the $\omega$ distribution. Parameters in parentheses are not free and should not be counted: for example, in M1a, $p_1$ is not a free parameter as $p_1 = 1 - p_0$. In both likelihood ratio tests comparing M1a against M2a and M7 against M8, df = 2. The site models are specified using NSsites.

The *site models* allow the $\omega$ ratio to vary among sites (among codons or amino acids in the protein) (Nielsen and Yang 1998; Yang et al. 2000b). A number of such models are implemented in codeml using the variable Nssites (and model = 0). You can run several Nssites models in one go, by specifying several values for NSsites. For example, NSsites = 0 1 2 7 8 will fit 5 models to the same data in one go. The site models have been used in real data analyses and evaluated in computer simulation studies (Anisimova et al. 2001; Anisimova et al. 2002; Anisimova et al. 2003; Wong et al. 2004). Two pairs of models appear to be particularly useful, forming two likelihood ratio tests of positive selection. The first compares M1a (NearlyNeutral) and M2a (PositiveSelection), while the second compares M7 (beta) and M8 (beta&$\omega$). M1a (NearlyNeutral) and M2a (PositiveSelection) are slight modifications of models M1 (neutral) and M2 (selection) in (Nielsen and Yang 1998). In both test, df = 2 should be used. The M1a-M2a comparison appears to be more robust (or less powerful) than the M7-M8 comparison. See the table below. The old models M1 and M2 fix $\omega_0 = 1$ and $\omega_1 = 1$, and are unrealistic as they do not account for sites with $0 < \omega < 1$. In the new models M1a and M2a, described in Wong *et al.* (2004) and Yang *et al.* (2005) and implemented since version 3.14 (2004), $0 < \omega_0 < 1$ is estimated from the data while $\omega_1 = 1$ is fixed. Also the BEB procedure for identifying positively selected sites is implemented since version 3.14.

A third test compares the null hypothesis M8a (beta&$\omega_s$ =1) and M8 (Swanson et al. 2003; Wong et al. 2004). M8a is specified using NSsites = 8, fix_omega = 1, omega = 1. The null distribution is the 50:50 mixture of point mass 0 and $\chi_1^2$ (Self and Liang 1987). The critical values are 2.71 at 5% and 5.41 at 1% (as opposed to 3.84 for 5% and 6.63 for 1% for $\chi_1^2$). Note that the *p* value for a 50:50 mixture of $\chi_j^2$ and $\chi_k^2$ is just the average of the two *p* values from the two distributions, in the case of M8a-M8 comparison, you get the *p* value from $\chi_1^2$ and then half it to get the *p* value for the mixture distribution. You can also use $\chi_1^2$ (Wong et al. 2004).

We suggest that The M0-M3 comparison should be used as a test of variable $\omega$ among sites rather than a test of positive selection. However, the model of a single $\omega$ for all sites is probably wrong in every functional protein, so there is little point of testing.

The naïve empirical Bayes (NEB) (Nielsen and Yang 1998; Yang et al. 2000b) and the Bayes empirical Bayes (BEB) (Yang et al. 2005) are available for calculating the posterior probabilities for site classes, and can be used to identify sites under positive selection if the likelihood ratio test is significant.  NEB uses the MLEs of parameters (such as the proportions and $\omega$ ratios) but do not account for their sampling errors, while BEB deals with the sampling errors by applying a Bayesian prior.  BEB is implemented under models M2a and M8 only.  We suggest that you ignore the NEB output and use the BEB results only.

---

The BEB output has the following format:

Prob(w>1) mean w

135 K 0.983* 4.615 +- 1.329

Interpretation: 4.615 is the approximate mean of the posterior distribution for w, and 1.329 is the square root of the variance in the posterior distribution for w.   The program prints out an * if the posterior probability is >95%, and ** if the probability is > 99%.

---

*Suzuki and Gojobori's (1999) method for detecting sites under positive selection.*  In the terminology used here, The SG99 method tests whether the $\omega$ ratio for each site is >1 or <1.  It uses parsimony to reconstruct ancestral sequences, and then for each site, counts the numbers of synonymous and nonsynonymous differences and the numbers of synonymous and nonsynonymous sits.  It then test whether the $\omega$ ratio at the site is significantly different from 1.  Errors in the ancestral sequence reconstruction are ignored.  Suzuki has a program called AdaptSite that implements the test.  This is an intuitive test, and is known to lack power.

In codeml, a test of this kind is implemented as a by-product of ancestral sequence reconstruction in codeml and baseml, which use ML to reconstruct ancestral sequences.  Use RateAncestor = 1.  The choice of baseml versus codeml and also the choice of substitution model for each program affects ancestral sequence reconstruction only.  The later steps are the same, and follow Suzuki & Gojobori (1999).  For codeml, you can use M0 (NSsites = 0 and model = 0).  If you want, you can try some other models, such as NSsites = 2 or 8.  The models for ancestral reconstruction typically make little difference.  For baseml, you should have "GC" on the first line of the sequence data file to indicate that the sequences are protein coding.  Use icode (= 0 for the universal code and 1 for vertebrate mitochondrial code) in the control file to specify the genetic code, as in codeml.  The following "multiple-gene" model is close to M0: model = 4 Mgene = 4 (see (Yang 1996a) and the section titled "Models for combined analysis of partitioned data").

The *branch-site* models allow $\omega$ to vary both among sites in the protein and across branches on the tree and aim to detect positive selection affecting a few sites along particular lineages (called foreground branches).  Initially Yang and Nielsen {, 2002 #1657; } implemented model A (model = 2 NSsites = 2) and model B (model = 2 NSsites = 3).  The tests did not work well in simulations (Zhang 2004), so a change was introduced to model A (table 3) (Yang et al. 2005; Zhang et al. 2005), with two tests constructed.  Test 2, also known the branch-site test of positive selection, is the recommended test.  This compares the modified model A with the corresponding null model with $\omega_2$ = 1 fixed (fix_omega = 1 and omega = 1).  The null distribution is the 50:50 mixture of point mass 0 and $\chi_1^2$, with critical values 2.71 at 5% and 5.41 at 1%.  We recommend instead $\chi_1^2$ (with critical values 3.84 and 5.99) to guide against violations of model assumptions.

Similarly both the NEB and BEB methods for calculating posterior probabilities for site classes are implemented for the modified branch-site model A (not for model B).  You should use model A in combination with the BEB procedure and ignore the NEB output.

Table 3. Parameters in the branch-site models

| Site class | Proportion | Old model A (np = 3) | | New model A (np = 4) | |
|---|---|---|---|---|---|
| | | Background | Foreground | Background | Foreground |
| 0 | $p_0$ | $\omega_0 = 0$ | $\omega_0 = 0$ | $0 < \omega_0 < 1$ | $0 < \omega_0 < 1$ |
| 1 | $p_1$ | $\omega_1 = 1$ | $\omega_1 = 1$ | $\omega_1 = 1$ | $\omega_1 = 1$ |
| 2a | $(1 - p_0 - p_1)\, p_0/(p_0 + p_1)$ | $\omega_0 = 0$ | $\omega_2 \geq 1$ | $0 < \omega_0 < 1$ | $\omega_2 \geq 1$ |
| 2b | $(1 - p_0 - p_1)\, p_1/(p_0 + p_1)$ | $\omega_1 = 1$ | $\omega_2 \geq 1$ | $\omega_1 = 1$ | $\omega_2 \geq 1$ |

NOTE.— Branch-site model A is specified using model = 2 NSsites = 2.  This is the alternative model in the branch-site test of positive selection.  The null model  fixes $\omega_2 = 1$.  The likelihood ratio test has df = 1 (see text).

*Clade model* C is specified by model = 3 Nssites = 2 while clade model D is specified by model = 3 NSsites = 3 using ncatG to specify the number of site classes (Bielawski and Yang 2004;  see also Forsberg and Christiansen 2003).  Clade model C is changed, in a similar way to branch-site model A.  The new model C replaces $\omega_0 = 0$ by $0 < \omega_0 < 1$ and has 5 parameters in the $\omega$ distribution: $p_0$, $p_1$, $\omega_0$, $\omega_2$, and $\omega_3$.  The new model C can be compared with the new M1a (NearlyNeutral), which has 2 parameters, with d.f. $\approx 3$.

Table 4. Parameters in clade model C (old vs. new)

| Site class | Proportion | Old model C (np = 4) | | New model C (np = 5) | |
|---|---|---|---|---|---|
| | | Clade 1 | Clade 2 | Clade 1 | Clade 2 |
| 0 | $p_0$ | $\omega_0 = 0$ | $\omega_0 = 0$ | $0 < \omega_0 < 1$ | $0 < \omega_0 < 1$ |
| 1 | $p_1$ | $\omega_1 = 1$ | $\omega_1 = 1$ | $\omega_1 = 1$ | $\omega_1 = 1$ |
| 2 | $p_2 = 1 - p_0 - p_1$ | $\omega_2$ | $\omega_3$ | $\omega_2$ | $\omega_3$ |

Clade model D can work with ncatG = 3 or 2.  The option variable ncatG is ignored when you specify branch-site models A and B, and clade model C, since the number of categories is fixed in the model.

The BEB procedure is implemented for clade model C but not for model D.  You should use model C in combination with the BEB procedure.  Ignore the NEB output.

An extension has been made to clade models C and D to allow for more than two clades or branch types.  The branch types are specified using labels in the tree file.  If you have four branch types (labelled using #0, #1, #2, #3), the clade model will look like the following.  Here $\omega_2$, $\omega_3$, $\omega_4$, $\omega_5$ are independent parameters, optimized in the range $(0, \infty)$.  The BEB calculation under clade model C is very expensive (with each additional branch type increasing the computation by 10 folds), so that the model should be used with just a few branch types.

Table 5. Parameters in clade models C or D, with more than two branch types

| Site class | Proportion | Clade 1 | Clade 2 | Clade 3 | Clade 4 |
|---|---|---|---|---|---|
| 0 | $p_0$ | $0 < \omega_0 < 1$ | $0 < \omega_0 < 1$ | $0 < \omega_0 < 1$ | $0 < \omega_0 < 1$ |
| 1 | $p_1$ | $\omega_1$ | $\omega_1$ | $\omega_1$ | $\omega_1$ |
| 2 | $p_2 = 1 - p_0 - p_1$ | $\omega_2$ | $\omega_3$ | $\omega_4$ | $\omega_5$ |

In clade model C, $\omega_1 = 1$ is fixed, while in clade model D, $\omega_1$ is estimated as a free parameter.

The mutation-selection models of Yang and Nielsen (2008) are implemented using the control variables

```
    CodonFreq = 7  * 0:1/61 each, 1:F1X4, 2:F3X4, 3:codon table
                   * 4:F1x4MG, 5:F3x4MG, 6:FMutSel0, 7:FMutSel
      estFreq = 1
```

where CodonFreq = 6 specifies FMutSel0 and 7 specifies FmutSel. If estFreq = 1, the frequency/fitness parameters are estimated by ML from the data, while if estFreq = 0, they are calculated using the observed frequencies in the data. See Yang and Nielsen (2008) for details of the model. Look at the README file in the examples/mtCDNAape/ folder, to see how to duplicate results of table 1 in that paper.

## Amino acid substitution models

I made a distinction between empirical and mechanistic models of amino acid substitution (Yang et al. 1998; 2006: Chapter 2). Empirical models implemented in codeml include Dayhoff (Dayhoff et al. 1978), JTT (Jones et al. 1992), WAG (Whelan and Goldman 2001), mtMam (Yang et al. 1998), mtREV (Adachi and Hasegawa 1996a), etc. These are estimates of substitution rate parameters under the general time reversible model from real datasets.

Mechanistic models are formulated by considering the mutational distance between the amino acids as determined by the locations of their encoding codons in the genetic code, and the filtering of mutations by natural seletion operating on the protein level (Yang et al. 1998). The program aaml implements a few such models, specified by the variable aaDist.

### The control file

Since the codon based analysis and the amino acid based analysis use different models, and some of the control variables have different meanings, it may be a good idea to use different control files for codon and amino acid sequences. The default control file for codeml is codeml.ctl, as shown below.

```
      seqfile = stewart.aa * sequence data file name
      outfile = mlc * main result file name
     treefile = stewart.trees * tree structure file name

        noisy = 9  * 0,1,2,3,9: how much rubbish on the screen
      verbose = 0  * 1: detailed output, 0: concise output
      runmode = 0  * 0: user tree;  1: semi-automatic;  2: automatic
                   * 3: StepwiseAddition; (4,5):PerturbationNNI; -2: pairwise

      seqtype = 2  * 1:codons; 2:AAs; 3:codons-->AAs
    CodonFreq = 2  * 0:1/61 each, 1:F1X4, 2:F3X4, 3:codon table
*       ndata = 10
        clock = 0   * 0:no clock, 1:clock; 2:local clock; 3:TipDate

       aaDist = 0  * 0:equal, +:geometric; -:linear, 1-6:G1974,Miyata,c,p,v,a
                   * 7:AAClasses
   aaRatefile = wag.dat * only used for aa seqs with model=empirical(_F)
                   * dayhoff.dat, jones.dat, wag.dat, mtmam.dat, or your own

        model = 2
                   * models for codons:
                       * 0:one, 1:b, 2:2 or more dN/dS ratios for branches
                   * models for AAs or codon-translated AAs:
                       * 0:poisson, 1:proportional,2:Empirical,3:Empirical+F
                       * 6:FromCodon, 8:REVaa_0, 9:REVaa(nr=189)

      NSsites = 0  * 0:one w;1:neutral;2:selection; 3:discrete;4:freqs;
                   * 5:gamma;6:2gamma;7:beta;8:beta&w;9:beta&gamma;
                   * 10:beta&gamma+1; 11:beta&normal>1; 12:0&2normal>1;
                   * 13:3normal>0

        icode = 0  * 0:universal code; 1:mammalian mt; 2-11:see below
        Mgene = 0  * 0:rates, 1:separate;

    fix_kappa = 0  * 1: kappa fixed, 0: kappa to be estimated
        kappa = 2  * initial or fixed kappa
    fix_omega = 0  * 1: omega or omega_1 fixed, 0: estimate
        omega = .4 * initial or fixed omega, for codons or codon-based AAs
```

```
    fix_alpha = 1  * 0: estimate gamma shape parameter; 1: fix it at alpha
        alpha = 0. * initial or fixed alpha, 0:infinity (constant rate)
       Malpha = 0  * different alphas for genes
        ncatG = 3  * # of categories in dG of NSsites models

      fix_rho = 1  * 0: estimate rho; 1: fix it at rho
          rho = 0. * initial or fixed rho,   0:no correlation

        getSE = 0  * 0: don't want them, 1: want S.E.s of estimates
 RateAncestor = 0  * (0,1,2): rates (alpha>0) or ancestral states (1 or 2)

   Small_Diff = .5e-6
*    cleandata = 0  * remove sites with ambiguity data (1:yes, 0:no)?
* fix_blength = 0   * 0: ignore, -1: random, 1: initial, 2: fixed
       method = 0   * 0: simultaneous; 1: one branch at a time
```

The variables `seqfile`, `outfile`, `treefile`, `noisy`, `Mgene`, `fix_alpha`, `alpha`, `Malpha`, `fix_rho`, `rho`, `clock`, `getSE`, `RateAncestor`, `Small_Diff`, `cleandata`, `ndata`, `fix_blength`, and `method` are used in the same way as in `baseml.ctl` and are described in the previous section. The variable `seqtype` specifies the type of sequences in the data; `seqtype = 1` means codon sequences (the program is then `codonml`); 2 means amino acid sequences (the program is then `aaml`); and 3 means codon sequences which are to be translated into proteins for analysis.

## *Codon sequences (`seqtype = 1`)*

**`CodonFreq`** specifies the equilibrium codon frequencies in codon substitution model. These frequencies can be assumed to be equal (1/61 each for the standard genetic code, `CodonFreq = 0`), calculated from the average nucleotide frequencies (`CodonFreq = 1`), from the average nucleotide frequencies at the three codon positions (`CodonFreq = 2`), or used as free parameters (`CodonFreq = 3`). The number of parameters involved in those models of codon frequencies is 0, 3, 9, and 60 (for the universal code), for `CodonFreq = 0`, 1, 2, and 3 respectively.

**`clock`** See the notes for the baseml control file.

**`aaDist`** specifies whether equal amino acid distances are assumed (= 0) or Grantham's matrix is used (= 1) (Yang et al. 1998). The example mitochondrial data set analyzed in that paper is included in the example/mtdna folder in the package. aaDist = 7 (AAClasses), which is implemented for both codon and amino acid sequences, allow you to have several types of amino acid substitutions and let the program estimate their different rates. The model was implemented in Yang *et al.* (1998). The number of substitution types and which pair of amino acid changes belong which type is specified in a file called OmegaAA.dat. You can use the model to fit different $\omega$ ratios for "conserved" and "charged" amino acid substitutions. The folder examples/mtCDNA contain example files for this model; check the readme file in that folder.

**`runmode = -2`** performs ML estimation of $d_S$ and $d_N$ in pairwise comparisons of protein-coding sequences (seqtype = 1). The program will collect estimates of $d_S$ and $d_N$ into the files `2ML.dS` and `2ML.dN`. Since many users seem interested in looking at $d_N/d_S$ ratios among lineages, examination of the tree shapes indicated by branch lengths calculated from the two rates may be interesting although the analysis is *ad hoc*. If your species names have no more than 10 characters, you can use the output distance matrices as input to Phylip programs such as `neighbor` without any change. Otherwise you need to edit the files to cut the names short. For amino acid sequences (seqtype = 2), option runmode = -2 lets the program calculate ML distances under the substitution model by numerical iteration, either under the model of one rate for all sites (alpha = 0) or under the gamma model of rates for

sites (alpha ≠ 0).  In the latter case, the continuous gamma is used and the variable ncatG is ignored.  (With runmode = 0, the discrete gamma is used.)

**model** specifies the *branch* models (Yang 1998; Yang and Nielsen 1998).  model = 0 means one $\omega$ ratio for all branches; 1 means one ratio for each branch (the free-ratio model); and 2 means an arbitrary number of ratios (such as the 2-ratios or 3-ratios models).  When model = 2, you have to group branches on the tree into branch groups using the symbols # or $ in the tree.  See the section above about specifying branch/node labels.

With model = 2, the variable fix_omega fixes the last $\omega$ ratio ($\omega_{k-1}$ if you have $k$ ratios in total) at the value of omega specified in the file.  This option is used to test whether the ratio for the foreground branch is significantly greater than one.  See the examples/lysozyme/ folder to duplicate the results of Yang (1998).

**NSsites** specifies the site models, with NSsites = $m$ corresponds to model M$m$ in Yang *et al.* (2000b).  The variable ncatG is used to specify the number of categories in the $\omega$ distribution under some models.  In Yang *et al.* (2000b), this is 3 for M3 (discrete), 5 for M4 (freq), 10 for the continuous distributions (M5: gamma, M6: 2gamma, M7: beta, M8:beta&w, M9:beta&gamma, M10: beta&gamma+1, M11:beta&normal>1, and M12:0&2normal>1, M13:3normal>0).  For example, M8 has 11 site classes (10 from the beta distribution plus 1 additional class for positive selection with $\omega_s \geq 1$).  See the section Codon substittion models above about the changes to M1 and M2 introduced in PAML version 3.14.

You can run several Nssites models in one batch, as follows, in which case the default values of ncatG, as in Yang *et al.* (2000b), are used.

```
NSsites = 0 1 2 3 7 8
```

The posterior probabilities for site classes as well as the expected $\omega$ values for sites are listed in the file rst, which may be useful to identify sites under positive selection.

Look at the examples/hivNSsites/ and examples/lysine/ folders for example of analysis under the site models.

The *branch-site* model A (see the section Codon substitution models above) is specified by using both variables model and NSsites.

```
Model A: model = 2, NSsites = 2,  fix_omega = 0
```

This is the alternative model for the branch-site test of positive selection.  The null model is also the branch-site model A but with $\omega_2 = 1$ fixed, specified by

```
Model A1: model = 2, NSsites = 2,  fix_omega = 1, omega = 1
```

Here are some notes about two old tests that we do not recommend.  The old branch-site model B (Yang and Nielsen 2002) is specified by

```
Model B: model = 2, NSsites = 3
```

The null model for the old test B is the NSsites model 3 (discrete) with 2 site classes:

```
Site model 3: model = 0, NSsites = 3,  ncatG = 2
```

Use d.f. = 2. The large lysozyme data set analyzed by Yang and Nielesn (2002) is in the examples/lysozyme folder.

Also branch-site test 1 described by Yang *et al.* (2005) and Zhang *et al.* (2005) uses the modified branch-site model A as the alternative hypothesis, while the null hypothesis is the new site model M1a (NearlyNeutral), with d.f. = 2. This test can be significant when the foreground branches are either under relaxed selective constraint or under positive selection. The advice is that you use test 2 instead, which is also known as the branch-site test of positive selection.

The *clade* models C and D of Bielawski and Yang (2004) are specified by

```
Model C: model = 3, NSsites = 2
Model D: model = 3, NSsites = 3  ncatG = 2
```

See that paper for details. Similarly model C is modified and the BEB procedure is implemented for model C only; see above.

**icode** specifies the genetic code. Eleven genetic code tables are implemented using `icode` = 0 to 10 corresponding to transl_table 1 to 11 in GenBank. These are 0 for the universal code; 1 for the mammalian mitochondrial code; 3 for mold mt., 4 for invertebrate mt.; 5 for ciliate nuclear code; 6 for echinoderm mt.; 7 for euplotid mt.; 8 for alternative yeast nuclear; 9 for ascidian mt.; and 10 for blepharisma nuclear. There is also an additional code, called Yang's *regularized code*, specified by `icode` = 11. In this code, there are 16 amino acids, all differences at the first and second codon positions are nonsynonymous and all differences at the third codon positions are synonymous; that is, all codons are 4-fold degenerate. There is yet no report of any organisms using this code.

**Mgene**, in combination with option G in the sequence data file, specifies partition models (Yang and Swanson 2002), as summarized in table 6. The lysin data set used in that paper is included in the examples/ folder of the package. The analysis separates the buried and exposed amino acids in the lysin into two partitions ("genes"), and heterogeneity between the partitions are accounted for in the analysis. You can read the readme file and try to duplicate the results in table 6 of Yang & Swanson (2002).

Table 6. Setups of partition models of codon substitution

| Sequence file | Control file | Parameters across genes |
|---|---|---|
| No G | Mgene = 0 | everything equal |
| Option G | Mgene = 0 | the same ($\kappa$, $\omega$) and $\pi$, but different $c$s (proportional branch lengths) |
| Option G | Mgene = 2 | the same ($\kappa$,$\omega$), but different $\pi$s and $c$s |
| Option G | Mgene = 3 | the same $\pi$, but different ($\kappa$, $\omega$) and $c$s |
| Option G | Mgene = 4 | different ($\kappa$, $\omega$), $\pi$s, and $c$s |
| Option G | Mgene = 1 | separate analysis |

**fix_alpha, alpha**: For codon models, the pair fix_alpha and alpha specify the model of gamma rates for sites, in which the relative rate for a site varies among codons according to the gamma distribution, but the $\omega$ ratio stays the same over all sites. This is a lazy extension of the gamma model of rates for nucleotides and amino acids. I don't like this model and suggest that you use the NSsites models instead (which is specified using the NSsites variable, with fix_alpha = 1, alpha = 0). The program should abort if you specify both NSsites and alpha.

**RateAncestor**: See descriptions for the baseml control file.

**Output** for codon sequences (`seqtype` = 1): The codon frequencies in each sequence are counted and listed in a genetic code table, together with their sums across species. Each table contains six or fewer species. For data of multiple genes (option G in the sequence file), codon frequencies in each gene (summed over species) are also listed. The nucleotide distributions at the three codon positions are also listed. The method of Nei and Gojobori (1986) is used to calculate the number of synonymous substitutions per synonymous site ($d_S$) and the number of nonsynonymous substitutions per nonsynonymous site ($d_N$) and their ratio ($d_N/d_S$). These are used to construct initial estimates of branch lengths for the likelihood analysis but are not MLEs themselves.

Results of ancestral reconstructions (`RateAncestor` = 1) are collected in the file `rst`. Under models of variable $d_N/d_S$ ratios among sites (`NSsites` models), the posterior probabilities for site classes as well as positively selected sites are listed in `rst`.

### *Amino acid sequences (*`seqtype` *= 2)*

**model** specifies the model of amino acid substitution: 0 for the Poisson model assuming equal rates for any amino acid substitutions (Bishop and Friday 1987); 1 for the proportional model in which the rate of change to an amino acid is proportional to the frequency of that amino acid. Model = 2 specifies a class of empirical models, and the empirical amino acid substitution rate matrix is given in the file specified by `aaRatefile`. Files included in the package are for the empirical models of Dayhoff (`dayhoff.dat`), JTT, WAG (wag.dat), mtMAM (`mtmam.dat`), mtREV24 (`mtREV24.dat`), etc.

If you want to specify your own substitution rate matrix, have a look at one of those files, which has notes about the file structure. Other options for amino acid substitution models should be ignored. To summarize, the variables `model`, `aaDist`, `CodonFreq`, `NSsites`, and `icode` are used for codon sequences (`seqtype` = 1), while `model`, `alpha`, and `aaRatefile` are used for amino acid sequences.

**Mgene**, in combination with option G in the sequence data file, specifies partition models (Yang and Swanson 2002), as summarized in table 6. The lysin data set used in that paper is included in the examples/ folder of the package. The analysis separates the buried and exposed amino acids in the lysin into two partitions ("genes"), and heterogeneity between the partitions are accounted for in the analysis. You can read the readme file and try to duplicate the results in table 6 of Yang & Swanson (2002).

Table 7. Setups of partition models of amino acid substitution

| Sequence file | Control file | Parameters across genes |
|---|---|---|
| No G | Mgene = 0 | everything equal |
| Option G | Mgene = 0 | the same $\pi$, but different $c$s (proportional branch lengths) |
| Option G | Mgene = 2 | different $\pi$s and $c$s |
| Option G | Mgene = 1 | separate analysis |

**runmode** also works in the same way as in `baseml`.ctl. Specifying `runmode` = –2 will forces the program to calculate the ML distances in pairwise comparisons. You can change the following variables in the control file `codeml.ctl`: `aaRatefile`, `model`, and `alpha`.

If you do pairwise ML comparison (`runmode` = -2) and the data contain ambiguity characters or alignment gaps, the program will remove all sites which have such characters from all sequences before the pairwise comparison if `cleandata` = 1. This is known as

"complete deletion". It will remove alignment gaps and ambiguity characters in each pairwise comparsion ("pairwise" deletion) if `cleandata` = 0. {{This does not seem to be true.  The program currently removes all sites with any ambiguities if `runmode` = -2.  Need checking.  Note by Ziheng 31/08/04.}} Note that in a likelihood analysis of multiple sequences on a phylogeny, alignment gaps are treated as ambiguity characters if `cleandata` = 0, and both alignment gaps and ambiguity characters are deleted if `cleandata` = 1.  Note that removing alignment gaps and treating them as ambiguity characters both underestimate sequence divergences. Ambiguity characters in the data (`cleandata` = 0) make the likelihood calculation slower.

**Output** for amino acid sequences (`seqtype` = 2):  The output file is self-explanatory and very similar to the result files for the nucleotide- and codon-based analyses.  The empirical models of amino acid substitution (specified by dayhoff.dat, jones.dat, wag.dat, mtmam.dat, or mtREV24.dat) do not involve any parameters in the substitution rate matrix.  When `RateAncestor` = 1, results for ancestral reconstruction are in the file `rst`. Calculated substitution rates for sites under models of variable rates for sites are in `rates`.

# 7  evolver

This program generates a naïve menu, like the following.

```
(1) Get random UNROOTED trees?
(2) Get random ROOTED trees?
(3) List all UNROOTED trees into file trees?
(4) List all ROOTED trees into file trees?
(5) Simulate nucleotide data sets (use MCbase.dat)?
(6) Simulate codon data sets      (use MCcodon.dat)?
(7) Simulate amino acid data sets (use MCaa.dat)?
(8) Calculate identical bi-partitions between trees?
(9) Calculate clade support values (read 2 treefiles)?
(0) Quit?
```

**Options 1, 2, 3, 4.**  The program can be used to generate a random tree, either unrooted or rooted, either with or without branch lengths.  It can also list all the trees for a fixed number of species.  Of course, you should do this for a small number of species only as otherwise your hard drive will be filled by useless trees.  Option 8 is for reading many trees from a tree file and then calculating bi-partition distances either between the first and all the remaining trees or between every pair.

**Option 9 (Clade support values)** can be used to summarize bootstrap or Bayesian analyses.  This reads two tree files.  The first file should include one tree, say, the maximum likelihood tree.  You should have the number of species and the number of tree (should be 1) at the beginning of this file.  The second tree file should include a collection of trees, such as 1000 maximum likelihood trees estimated from 1000 bootstrap pseudo-samples.  This option will then calculate the bootstrap support value for each clade on the ML tree in the first tree file, that is, the proportion of trees in the second file that contain the node or clade in the tree in the first file.  The second tree file does not have to have the numbers of species and trees on the first line.  If you run MrBayes, you can move the maximum likelihood tree or maximum *a posteriori* tree into the first file, and the second tree file can be the .t file generated by MrBayes, with no change necessary.  Right now species are represented by numbers only in the tree files, I think.  You can choose this option by running evolver, then option 9.  The program will then ask you to input two file names.  An alternative way, which bypasses the naïve menu, is to put the option and two file names at the command line:
     evolver 9 <MasterTreeFile> <TreesFile>

**Options 5, 6, 7 (Simulations).** The program evolver simulates nucleotide, codon, and amino acid sequences with user-specified tree topology and branch lengths.  The user specifies the substitution model and parameters in a control file; see below.  The program generates multiple data sets in one file in either PAML (output mc.paml) or PAUP* (output mc.paup) format.  If you choose the PAUP* format, the program will look for files with the following names: paupstart (which the program copies to the start of the data file), paupblock (which the program copies to the end of each simulated data set), and paupend (which the program incorporates at the end of the file).  This makes it possible to use PAUP* to analyze all data sets in one run.  Parameters for simulation are specified in three files: MCbase.dat, MCcodon.dat, and MCaa.dat for simulating nucleotide, codon, and amino acid sequences, respectively.  Run the default options while watching out for screen output.  Then have a look at the appropriate .dat files.  As an example, the MCbase.dat file is reproduced below.  Note that the first block of the file has the inputs for evolver, while the rest are notes.  The tree length is the expected number of substitutions per site along all branches in the phylogeny, calculated as the sum of the branch lengths.  This variable was introduced when I was doing simulations to evaluate the effect of sequence divergence while keeping the shape of the tree fixed.  evolver will scale the tree so that the branch lengths sum up to the specified tree length.  If you use –1 for the tree length, the program will use the branch lengths

given in the tree without the re-scaling.  Also note that the base frequencies have to be in a fixed order; this is the same for the amino acid and codon frequencies in `MCaa.dat` and `MCcodon.dat`.

```
0        * 0: paml format (mc.paml); 1:paup format (mc.nex)
367891      * random number seed (odd number)
5 1000000 1 * <# seqs>  <# nucleotide sites>  <# replicates>
-1           * <tree length, use -1 if tree has absolute branch lengths>
(((A :0.1, B :0.2) :0.12, C :0.3) :0.123, D :0.4, E :0.5) ;

3             * model: 0:JC69, 1:K80, 2:F81, 3:F84, 4:HKY85, 5:T92, 6:TN93, 7:REV
5             * kappa or rate parameters in model
0  0          * <alpha>  <#categories for discrete gamma>

0.1 0.2 0.3 0.4    * base frequencies
  T   C   A   G
```

The simulation options (5, 6, 7) of evolver can be run using a command line format, bypassing the naïve menu.  So here are all the possible ways of running evolver:

```
evolver
evolver 5 MyMCbaseFile
evolver 6 MyMCcodonFile
evolver 7 MyMCaaFile
```

The model of codon substitution used by option 6 here assumes the same $\omega$ ratio for all branches in the phylogeny and for all sites in the gene.  This is known as model M0 (one-ratio).  To simulate under the site models with variable $\omega$'s among sites, or the branch models with different $\omega$s among branches, or the branch-site models with $\omega$ varying both among sites and among branches, please read the file CodonSimulation.txt in the paml/Technical/Simulation/Codon/ folder.

The first variable controls the sequence data format to be used: with 0 meaning the paml/phylip format, 1 site pattern counts and 2 the nexus format.  The site pattern counts may be read by baseml or codeml later, and is useful if you have large data sets with many ($>10^6$) sites.  (See the section on sequence data file format above.)

evolver also can simulate data using a random tree with random branch lengths for each simulated data set.  You will have to change the source code and recompile.  Open evolver.c and find fixtree=1 in the routine Simulate() and change the value 1 into 0.  Then recompile and name the program as evolverRandomTree, say.

```
cc -o evolverRandomTree -O2 evolver.c tools.c –lm
evolver 5 MCbaseRandomTree.dat
```

The control data file such as MCbase.dat has to be changed as well.  An example file named MCbaseRandomTree.dat is included in the package.  This has the lines for "tree length" and tree topology replaced by a line for the birth rate $\lambda$, death rate $\mu$, sampling fraction $\rho$, and the tree height (the expected number of substitutions per site from the root to the tips).  The trees are generated using the birth-death process with species sampling (Yang and Rannala 1997).  The clock is assumed.  You will have to change the source code to relax the clock.  If you choose 0 (paml) for the file format, the random trees are printed out in the file ancestral.txt (?); this you can read from within Rod Page's TreeView.  If you choose 2 (nexus) for file format, the program prints out the tree in a tree block in the sequence data file.

The evolver program also has a few options for listing all trees for a specified small number of species, and for generating random trees from a model of cladogenesis, the birth-death process with species sampling (Yang and Rannala 1997).  It also has an option for calculating the partition distance between tree topologies.

***Monte Carlo simulation algorithm used in evolver.***  You can read about more details in the section "Models and Analyses".  See also Chapter 9 in Yang (2006).  Here are some brief notes.  Evolver simulates data sets by "evolving" sequences along the tree.  First, a sequence is generated for the

root using the equilibrium nucleotide, amino acid, or codon frequencies specified by the model and/or the data file (`MCbase.dat`, `MCcodon.dat`, and `MCaa.dat`, respectively). Then each site evolves along the branches of the tree according to the branch lengths, parameters in the substitution model etc. When the sites in the sequence evolve according to the same process, the transition probability matrix is calculated only once for all sites for each branch. For so called site-class models (such as the gamma, and the NSsites codon models), different sites might have different transition probability matrices. Given the character at the start of the branch, the character at the end of the branch is sampled from a multinomial distribution specified by the transition probabilities from the source character. Sequences at the ancestral nodes are generated during the simulation and printed out in the file ancestral.txt.

Some people wanted to specify the sequence at the root rather than letting the program generate a random sequence. This can be achieved by putting a sequence in the file RootSeq.txt. The sequence cannot have ambiguities or gaps or stop codons. In almost all simulations, it is simply wrong to fix the root sequence, so you should resist the temptation of making the mistake. If you want the simulation to reflect your particular gene, you may estimate parameters under a model from that gene and then simulate data sets using the parameter estimates.

# 8  yn00

The program `yn00` implements the method of Yang and Nielsen (2000) for estimating synonymous and nonsynonymous substitution rates between two sequences ($d_S$ and $d_N$). The method of Nei and Gojobori (1986) is also included. The ad hoc method implemented in the program accounts for the transition/transversion rate bias and codon usage bias, and is an approximation to the ML method accounting for the transition/transversion rate ratio and assuming the F3x4 codon frequency model. We recommend that you use the ML method (`runmode= -2`, `CodonFreq = 2` in `codeml.ctl`) as much as possible even for pairwise sequence comparison.

```
    seqfile = abglobin.nuc * sequence data file name
    outfile = yn          * main result file
    verbose = 0     * 1: detailed output (list sequences), 0: concise output

      icode = 0  * 0:universal code; 1:mammalian mt; 2-10:see below
  weighting = 0  * weighting pathways between codons (0/1)?
  commonf3x4 = 0  * use one set of codon freqs for all pairs (0/1)?
```

The control file `yn00.ctl`, an example of which is shown above, specifies the sequence data file name (`seqfile`), output file name (`outfile`), and the genetic code (`icode`). Sites (codons) involving alignment gaps or ambiguity nucleotides in any sequence are removed from all sequences. The variable `weighting` decides whether equal weighting or unequal weighting will be used when counting differences between codons. The two approaches will be different for divergent sequences, and unequal weighting is much slower computationally. The transition/transversion rate ratio $\kappa$ is estimated for all sequences in the data file and used in subsequent pairwise comparisons. `commonf3x4` specifies whether codon frequencies (based on the F3x4 model of `codonml`) should be estimated for each pair or for all sequences in the data. Besides the main result file, the program also generates three distance matrices: `2YN.dS` for synonymous rates, `2YN.dN` for nonsynonymous rates, `2YN.t` for the combined codon rate ($t$ is measured as the number of nucleotide substitutions per codon). Those are lower-diagonal distance matrices and are directly readable by some distance programs such as `NEIGHBOR` in Felesenstein's PHYLIP package.

# 9  mcmctree

## Overview

The program mcmctree may be the first Bayesian phylogenetic program (Yang and Rannala 1997; Rannala and Yang 1996) , but was very slow and decommissioned since the release of MrBayes (Huelsenbeck and Ronquist 2001).

Since PAML version 3.15 (2005), mcmctree implements the MCMC algorithms of Yang and Rannala (2006) and then of Rannala and Yang (2007) for estimating species divergence times on a given rooted tree using multiple fossil calibrations. This is similar to the multidivtime program of Jeff Thorne and Hiro Kishino. The differences between the two programs are discussed by Yang and Rannala (2006) and Yang (2006, Section 7.4); see also below.

Please refer to any book on Bayesian computation, for example, Chapter 5 in Yang (2006) for the basics of MCMC algorithms.

Here are some notes about the program.

- Before starting the program, resize the window to have 100 columns instead of 80. (On Windows XP/Vista, right-click the command prompt window title bar and change Properties - Layout - Window Size - Width.)

- The tree, supplied in the tree file, must be a rooted binary tree: every internal node should have exactly two daughter nodes. You should not use a consensus tree with polytomies for divergence time estimation using MCMCTREE. Instead you should use a bifurcating ML tree or NJ tree or traditional morphology tree. Note that a binary tree has a chance of being correct, while a polytomy tree has none.

- The tree must not have branch lengths. For example, ((a:0.1, b:0.2):0.12, c:0.3) '>0.8<1.0'; does not work, while ((a, b), c) '>0.8<1.0'; is fine.

- Under the relaxed-clock models (clock = 2 or 3) and if there is no calibration on the root, a loose upper bound (maximal age constraint) must be specified in the control file (RootAge). (There should be no need to use RootAge if clock = 1, but the program insists that you have it. I will try to fix this.)

- *Time unit.* The time unit should be chosen such that the node ages are about 0.01-10. If the divergence times are around 100-1000MY, then 100MY may be one time unit. The priors on times and on rates and the fossil calibrations should all be specified based on your choice of the time scale. For example, if one time unit is 10MY, the following
    ```
    rgene_gamma = 2 20        * gamma prior for overall rates for genes
    sigma2_gamma = 10 100     * gamma prior for sigma^2  (for clock=2 or 3)
    ```
    means an overall average rate of $2/20 = 0.1$ substitutions per site per 10MY or $10^{-8}$ substitutions per site per year, which is reasonable for mammalian mitochondrial genes. The gamma prior here has the shape parameter $\alpha = 2$, and is fairly diffuse. If you change the time unit, you should keep the shape parameter fixed and change the scale parameter $\beta$ to have the correct mean. In other words, to use one time unit to represent 100MY, the above should become
    ```
    rgene_gamma = 2 2         * gamma prior for overall rates for genes
    sigma2_gamma = 10 100     * gamma prior for sigma^2  (for clock=2 or 3)
    ```
    Don't change the prior for $\sigma^2$ since $\sigma^2$ is the variance of the log rate: the variance of the logarithm of the rate does not change when you rescale the rate by a constant. The fossil calibrations in the tree file should be changed accordingly. While ideally one would want the

biological results to be unchanged when one changes the time unit, we know that two components of the model are not invariant to the time scale: the log normal distribution for rates and the birth-death model for times. Nevertheless, tests by Mathieu Groussin did suggest that the choice of time scale has very minimal effects on the posterior time and rate estimates.

- It is important that you run the same analysis at least twice to confirm that the different runs produced very similar (although not identical) results. It is critical that you ensure that the acceptance proportions are neither too high nor too low. See below about the variable finetune.

- It is important that you run the program without sequence data (usedata = 0) first to examine the means and CIs of divergence times specified in the prior. In theory, the joint prior distribution of all times should be specified by the user. Nevertheless it is nearly impossible to specify such a complex high-dimensional distribution. Instead the program generates the joint prior by using the calibration distributions and the constaint on the root as well as the birth-death process model to generate the joint prior. This the prior used by the program in the dating analysis. You have to examine it to make sure it is sensible, judged by your knowledge of the species and the relevant fossil record. If necessary, you may have to change the fossil calibrations so that the prior look reasonable.

- The program right now does a simple summary of the MCMC samples, calculating the mean, median and the 95% CIs for the parameters. If you want more sophisticated summaries such as 1-D and 2-D density estimates, you can run a small program ds at the end of the mcmctree run, by typing    ds mcmc.out.

- You can compile the program to use hard bounds, which are implemented by setting the tail probability to $10^{-300}$ instead of 0.025. Copy the command with the flag -DHardBounds from the start of the file mcmctree.c onto the command line to compile an executable named mcmctreeHB.

## Fossil calibration

Fossil calibration information, in the form of statistical distributions of divergence times (or ages of nodes in the species tree), is specified in the tree file. See table 8 for a summary. Here "fossil" means any kind of external calibration data, including geological events. For a sensible analysis, one should have at least one lower bound and at least one upper bound on the tree, even though they may not be on the same node. The gamma, skew normal, and skew *t* distributions can act as both bounds, so one such calibration is enough to anchor the tree to enable a sensible analysis.

Table 8. Calibration distributions

| Calibration | #p | Specification | Density |
|---|---|---|---|
| L (lower or minimum bound) | 4 | '>0.06' or 'L(0.06)' or 'L(0.06, 0.2)' or 'L(0.06, 0.1, 0.5)' or 'L(0.06, 0.1, 0.5, 0.025)' | $L(t_L, p, c, p_L)$ specifies the minimum-age bound $t_L$, with offset $p$, and scale parameter $c$, and left tail probability $p_L$. The default values are $p = 0.1$, $c = 1$, and $p_L = 0.025$, so >0.06 or L(0.06) means L(0.06, 0.1, 1, 0.025), and L(0.06, 0.2) means L(0.06, 0.2, 1, 0.025). If you would like the minimum bound to be hard, use $p_L =$ 1e-300, but do not use $p_L = 0$. |
| U (upper or maximum bound) | 2 | '<0.08' or 'U(0.08)' or 'U(0.08, 0.025)' | Eq. 16 & fig. 2b in YR06. $U(t_U, p_R)$ specifies the maximum-age bound $t_U$, with right tail probability $p_R$. The default value is $p_U = 0.025$, so <0.08 or U(0.08) means U(0.08, 0.025). For example U(0.08, 0.1) means that there is 10% probability that the maximum bound 8MY is violated (i.e., the true age is older than 8MY). |
| B (lower & upper bounds or minimum & maximum bounds) | 4 | '>0.06<0.08' or 'B(0.06, 0.08)' or 'B(0.06, 0.08, 0.025, 0.025)' | Eq. 17 & fig. 2c in YR06 $B(t_L, t_U, p_L, p_U)$ specifies a pair bound, so that the true age is between $t_L$ and $t_U$, with the left and right tail probabilities to be $p_L$ and $p_U$, respectively. The default values are $p_L = p_U = 0.025$. |
| G (Gamma) | 2 | 'G(alpha, beta)' | Eq. 18 & fig. 2d in YR06 |
| SN (skew normal) | 3 | 'SN(location, scale, shape)' | Eq. 2 & plots below |
| ST (skew *t*) | 4 | 'ST(location, scale, shape, df)' | Eq. 4 & plots below |
| S2N (skew 2 normals) | 7 | 'SN2($p_1$, loc1, scale1, shape1, loc2, scale2, shape2)' | $p_1$: $1 - p_1$ mixture of two skew normals. |

Note .— #p is the number of parameters in the distribution, to be supplied by the user. Figure 2 in YR06 (Yang and Rannala 2006) is figure 7.11 in Yang (2006).

**(1) Lower bound** (minimal age) is specified as '>0.06' or 'L(0.06)', meaning that the node age is at least 6MY. Here we assume that one time unit is 100 million years. In PAML version 4.2, the implementation of the minimum bound has changed. Instead of the improper soft flat density of Figure 2a in Yang and Rannala (2006) or figure 7.11a in Yang (2006), a heavy-tailed density based on a truncated Cauchy distribution is now used (Inoue et al. 2010). The Cauchy distribution with location parameter $t_L(1 + p)$ and scale parameter $t_L c$ is truncated at $t_L$, and then made soft by adding 2.5% of density mass left of $t_L$. The resulting distribution has mode at $t_L(1 + p)$, and the 2.5% and 97.5% limits at $t_L$ and $t_L[1 + p + c \tan(\pi(\frac{1}{2} - \frac{0.025A}{0.975}))]$. If the minimum bound $t_L$ is based on good fossil data, the true time of divergence may be close to the minimum bound, so that a small $p$ and small $c$ should be used. It is noted that $c$ has a greater impact thatn $p$ on posterior time estimation. The program uses the default values $p = 0.1$ and $c = 1$. However, you are advised to use different values of $p$ and $c$ for each minimum bound, based on a careful assessment of the fossil data on which the bound is based. Below are a few plots of this density. The minimum bound is fixed at $t_L$

= 1, but one time unit can mean anything like 100Myr or 1000Myr.  For each value of $p$ (0.1 and 0.5), the four curves correspond to $c$ = 0.2, 0.5, 1, 2 (from top to bottom nearthe peak).  The 2.5% limit is at 1, while the 97.5% limits for those values of $c$ are 4.93, 12.12, 24.43, 49.20, respectively, when $p$ = 0.1, and are 4.32, 9.77, 20.65, 44.43 when $p$ = 0.5.  (Note that those values were incorrectly calculated in Inoue et al. 2010)
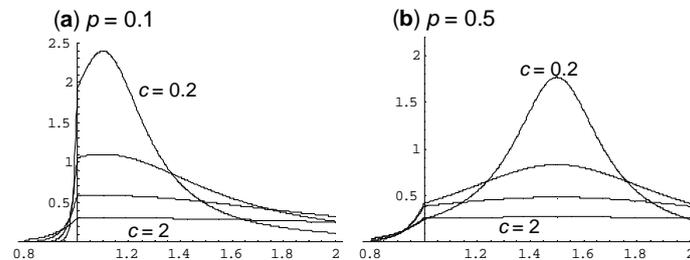


 (2) **Upper bound** (maximal age) is specified as '<0.08' or 'U(0.08)', meaning that the node age is at most 8MY.

(3) **Both lower and upper bounds** on the same node are specified as '>0.06<0.08' or 'B(0.06, 0.08)', meaning that the node age is between 6MY and 8MY.
Note that in all the above three calibrations (L, U, B), the bounds are soft, in that there is a 2.5% probability that the age is beyond the bound (see figure 2 in Yang and Rannala 2006;  or figure 7.11 in Yang 2006).

(4) **The gamma distribution**.  'G(188, 2690)' specifies the gamma distribution with shape parameter $\alpha$ = 188 and scale parameter $\beta$ = 2690.  This has the mean $\alpha/\beta$ = 0.07 and the 2.5 and 97.5 percentiles at ### and ###.  In earlier versions (3.15, 4a & 4b), the gamma was specified as '>.06=0.0693<.08', but this format is not used anymore.

```
((((human, (chimpanzee, bonobo)) 'G(188, 2690)', gorilla), (orangutan, sumatran))
'>.12<.16', gibbon);
```

In the tree above, the human-chimp divergence time has a gamma distribution G(188, 2690), while the orang-utan divergence time has soft bounds between 12MY and 16MY.

The above tree can be read in TreeView, with the calibration information in quotation marks treated as node labels.

You can use the MS Excel function GAMMADIST(X, alpha, beta, 0) to calculate and plot the density function (pdf) of the gamma distribution, and the function GAMMAINV(0.025, alpha, beta) to calculate the 2.5% percentile.  However, note that beta in Excel is $1/\beta$ in MCMCTREE (and other PAML programs).  In other words, the mean is $\alpha/\beta$ in MCMCTREE and $\alpha\beta$ in Excel.

(5) **Skew normal distribution SN(location, scale, shape) or SN($\xi$, $\omega$, $\alpha$)** (Azzalini and Genton 2007).  The basic form of the skew normal distribution has density

$$f(z; \alpha) = 2\phi(z)\Phi(\alpha z), \tag{1}$$

where $\phi()$ and $\Phi()$ are the PDF and CDF of the standard normal distribution respectively.  Then $x = \xi + \omega z$, has the skew normal distribution SN($\xi$, $\omega$, $\alpha$) with location parameter $\xi$, scale parameter $\omega$, and shape parameter $\alpha$.  The density is

$$f_{\text{SN}}(x;\,\xi,\,\omega,\,\alpha) = \frac{1}{\omega} \times \frac{1}{\sqrt{2\pi}}\, e^{-\frac{(x-\xi)^2}{2\omega^2}} \int_{-\infty}^{\alpha\left(\frac{x-\xi}{\omega}\right)} \frac{1}{\sqrt{2\pi}}\, e^{-\frac{u^2}{2}}\, du \,. \tag{2}$$

for $-\infty < \xi < \infty$, $0 < \omega < \infty$, $-\infty < \alpha < \infty$. Let $\delta = \alpha\big/\sqrt{1+\alpha^2}$ . The mean and variance are

$$E(x) = \xi + \omega\delta\sqrt{2/\pi},$$
$$\text{Var}(x) = \omega^2\left(1 - \tfrac{2\delta^2}{\pi}\right). \tag{3}$$

(6) **Skew *t* distribution, ST(location, scale, shape, df) or ST($\xi$, $\omega$, $\alpha$, $\nu$)** (Azzalini and Genton 2007), with location parameter $\xi$, scale parameter $\omega$, shape parameter $\alpha$, and degree of freedom $\nu$, has density

$$f_{\text{ST}}(x;\xi,\omega,\alpha,\nu) = \frac{2}{\omega} t(z;\nu)\, T\!\left(\alpha z\sqrt{(\nu+1)\big/(\nu+z^2)};\nu+1\right), \tag{4}$$

where $z = (x - \xi)/\omega$, $t$ and $T$ are the PDF and CDF of the standard $t$ distribution, respectively. These are defined as follows.

$$t(z;\nu) = \frac{\Gamma\left(\tfrac{1}{2}(\nu+1)\right)}{\sqrt{\pi\nu}\,\Gamma\left(\tfrac{1}{2}\nu\right)}\left[1 + \frac{z^2}{\nu}\right]^{-(\nu+1)/2},$$

$$T(z;\nu) = \begin{cases} \tfrac{1}{2}I_{\nu/(\nu+z^2)}\left(\tfrac{1}{2}\nu, \tfrac{1}{2}\right), & \text{if } z < 0, \\ 1 - T(-z;\nu), & \text{if } z \ge 0. \end{cases} \tag{5}$$

where $\Gamma()$ is the gamma function, and

$$I_p(a,b) = \frac{1}{B(a,b)} \int_0^p u^{a-1}(1-u)^{b-1}\,du \tag{6}$$

is the incomplete beta function ratio, or the CDF of the beta($a$, $b$) distribution, while

$$B(a,b) = \int_0^1 u^{a-1}(1-u)^{b-1}\,du = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)} \tag{7}$$

is the beta function.



SN(2, 1.5, 6)            SN(2, 1.5, –6)

ST(2, 1.5, 6, 2.2)                    ST(2, 1.5, –6, 2.2)



SN(2, 1.5, 10)                        SN(2, 1.5, 50)
ST(2, 1.5, 10, 3)                     ST(2, 1.5, 50, 1)
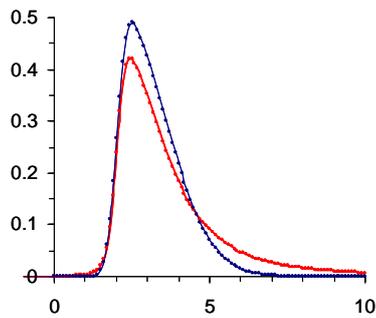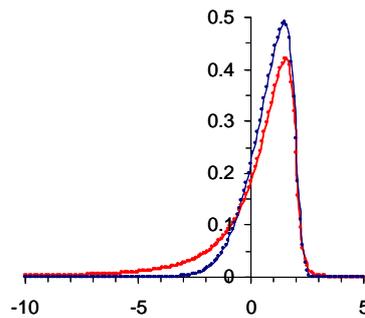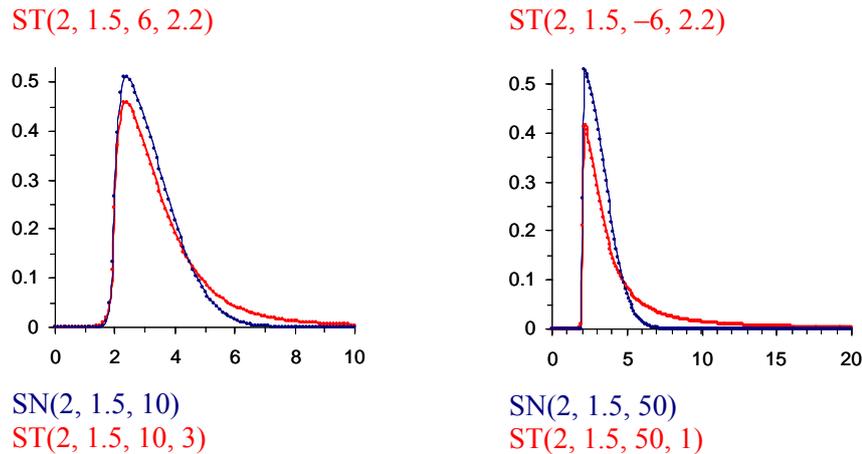
Figure 1. Density functions for skew normal (blue) and skew *t* (red) distributions. Skew *t* has heavier tails than skew normal.

Here are a few notes about the skew normal and skew *t* distributions.

- When the shape parameter $\alpha = 0$, the distributions become the standard (symmetrical) normal and *t* distributions.

- Changing $\alpha$ to $-\alpha$ flips the density at $x = \xi$ (the location parameter). Fossil calibrations should have long right tails, which means $\alpha > 0$.

- A larger $|\alpha|$ means more skewed distributions. When $|\alpha| = \infty$, the distribution is called folded normal or folded *t* distribution, that is, the normal or *t* distribution truncated at $\xi$ from the left ($\alpha = \infty$) or from the right ($\alpha = -\infty$).

- When the degree of freedom $\nu = \infty$, the *t* or skew *t* distribution becomes the normal or skew normal distribution. The smaller $\nu$ is, the heavier the tails are. A small $\nu$ (1-4 , say) with a large shape parameter $\alpha$ in the skew *t* distribution represents virtually hard minimal bound and very uncertain maximal bound. When $\nu = 1$, the *t* distribution is known as Cauchy distribution, which does not have mean or variance.

- Both skew normal and skew *t* distributions go from $-\infty$ to $\infty$. In MCMCTREE, negative values are automatically truncated, so only the positive part is used. If feasible, try to construct the distribution so that the probability for negative values is small (<0.1%, say).

- Please visit the web site http://azzalini.stat.unipd.it/SN/ to plot skew normal and skew *t* distributions. R routines are also available for such plots. The equations above are for my testing and debugging. I think I should remove them later on.

## The control file

You can use the files in the folder examples/SoftBound/ to duplicate the results of Yang and Rannala (2006: table 3) and Rannala and Yang (2007: table 2). Below is a copy of the control file mcmctree.ctl.

```
      seed = -1234567
   seqfile = mtCDNApri123.txt
  treefile = mtCDNApri.trees
   outfile = out

     ndata = 3
   usedata = 1    * 0: no data; 1:seq like; 2:use in.BV; 3: out.BV
     clock = 1    * 1: global clock; 2: independent rates; 3: correlated rates
   RootAge = '>0.8<1.2'

     model = 0    * 0:JC69, 1:K80, 2:F81, 3:F84, 4:HKY85
     alpha = 0    * alpha for gamma rates at sites
     ncatG = 5    * No. categories in discrete gamma

 cleandata = 0    * remove sites with ambiguity data (1:yes, 0:no)?
BlengthMethod = 0  * 0: arithmetic; 1: geometric; 2: Brownian

    BDparas = 1 1 0   * birth, death, sampling
kappa_gamma = 6 2      * gamma prior for kappa
alpha_gamma = 1 1      * gamma prior for alpha

rgene_gamma = 2 2      * gamma prior for rate for genes
sigma2_gamma = 1 1    * gamma prior for sigma^2     (for clock=2 or 3)

   finetune = 0.04 0.02 0.3 0.1 0.3 * time, rate, mixing, paras, RateParas

      print = 1
     burnin = 10000
    sampfreq = 10
    nsample = 10000
```

**seed** should be assigned a negative or positive integer. A negative integer (such as –1) means that the random number seed is determined from the current clock time. Different runs will start from different places and generate different results due to the stochastic nature of the MCMC algorithm. You should use this option and run the program at least twice, to confirm that the results are very similar between runs (identical to 1MY or 0.1MY, depending on the desired precision). If you obtain intolerably different results from different runs, you obviously won't have any confidence in the results. This lack of consistency between runs can be due to many reasons: including slow convergence, poor mixing, insufficient samples taken, or errors in the program. Thus you can check to make sure (i) that the chain is at reasonably good place when it reached 0% (the end of burn-in), indicating that the chain may have converged; (ii) that the acceptance proportion of all proposals used by the algorithm are neither too high nor too low (see below about **finetune**) indicating that the chain is mixing well; (iii) that you have taken enough samples (see **nsample** and **burnin** below). If you give **seed** a positive number, that number will be used as the real seed. Then running the program multiple times will produce exactly the same results. This is useful for debugging the program and should not be the default option for real data analysis.

**ndata** is the number of loci (or site partitions) in a combined analysis. The program allows some species to be missing at some loci. The mt primate data included protein-coding genes, and the three codon positions are treated as three different partitions. In the combined analysis of multiple gene loci, the same substitution model is used, but different parameters are assigned and estimated for each partition.

**usedata.** 0 means that the sequence data will not be used in the MCMC, with the likelihood set to 1, so that the MCMC approximates the prior distribution. This option is useful for testing and debugging the program, and is also useful for generating the prior distribution of the divergence times. The fossil calibrations and the constraints on the root you specify are not the real prior that is implemented in the program; for example, they may not even satisfy the requirement that ancestors

should be older than descendents.  The prior that is used by the program can be generated by running the chain without data.  usedata = 1 means that the sequence data will be used in the MCMC, with the likelihood calculated using the pruning algorithm of Felsenstein (1981), which is exact but very slow except for very small species trees.  This option is available for nucleotide sequences only, and the most complex model available is HKY85+Γ.  usedata = 2 and 3 implement a method of approximate likelihood calculation.  They can be used to analyze nucleotide, amino acid, and codon sequences, using nucleotide, amino acid, and codon substitution models, respectively.

**usedata = 2 inBVfilename** specifies the approximate likelihood calculation, with the input (gradient & Hessian matrix etc.) in the file.

**clock.**  The clock variable is used to implement three models concerning the molecular clock: 1 means global molecular clock, so that the rate is constant across all lineages on the tree (even though the rate may vary among multiple genes); 2 means the independent-rates model, and 3 the auto-correlated rates model.  See Rannala and Yang (2007) and Section §7.4 in Yang (2006) for details.

**RootAge.**  The RootAge variable is used to specify a loose constant on the age of the root, to constrain the root age from above.  It is used if no such constraint is available through a fossil calibration on the root.  Note that fossil calibrations are specified in the tree file.  Two formats are accepted, specifying either a maximum bound (e.g., RootAge = '<1.2') or a pair of minimum and maximum bounds (e.g., RootAge = '>0.8<1.2').  The RootAge variable is ignored if a fossil calibration on the root is specified in the tree file in the form of a maximum bound, a pair of minimum and maximum bounds, or a gamma distribution.  If the fossil calibration in the tree file is a minimum bound on the root (e.g. '>0.9'), and you specify RootAge = '<1.2', then the program implements the pair of bounds, equivalent to specifying the calibration '>0.9<1.2' on the root.

**model, alpha, ncatG** are used to specify the nucleotide substitution model.  These are the same variables as used in baseml.ctl.  If alpha ≠ 0, the program will assume a gamma-rates model, while alpha = 0 means that the model of one rate for all sites will be used.  Those variables have no effect when usedata = 2.

**cleandata** = 0 means that alignment gaps and ambiguity characters will be treated as missing data in the likelihood calculation (see pages 107-108 in Yang 2006).  = 1 means that any sites at which at least one sequence has an alignment gap or ambiguity character will be deleted before analysis.  This variable is used for usedata = 1 and 3 and has no effect if usedata = 2.

**BDparas** = 2 2 .1 specifies the three parameters (birth rate $\lambda$, death rate $\mu$ and sampling fraction $\rho$) in the birth-death process with species sampling (Yang and Rannala 1997), which is used to specify the prior of divergence times (Yang and Rannala 2006).  The node times are order statistics from a kernel density, which is specified by those parameters.  A few kernel densities are shown in figure 2 of Yang and Rannala (1997) or figure 7.12 in Yang (2006).  The Mathematica code for plotting the density for given parameters $\lambda$, $\mu$ and $\rho$ is posted at the web site http://abacus.gene.ucl.ac.uk/ziheng/data.html.  By adjusting parameters $\lambda$, $\mu$ and $\rho$ to generate different tree shapes, one can assess the impact of the prior on posterior divergence time estimation.  Intuitively, the node ages and thus the shape of the tree are determined by the parameters as follows.  There are $s - 1$ internal nodes and thus $s - 1$ node ages in the rooted tree of $s$ species.  The age of the root is fixed, so the $s - 2$ node ages are relative to the root age (they are all between 0 and 1).  We draw $s - 2$ independent random variables from the kernel density and order them.  Those ordered variables will then be the node ages.  Thus if the kernel density has the L shape, all internal nodes tend to be young (relative to the root), and the tree will have long internal branches and short tip branches.  In contrast, if the kernel density has the reverse L shape, the node ages are large and the nodes close to the root, then the tree will be bush-like.  See pages 250-251 in Yang (2006).  (Strictly

speaking the above description is accurate if fossil calibration is available for the root only but not for any other nodes. Otherwise the kernel density specifies the distribution of the ages of non-calibration nodes only, and the impact of the kernel on the joint distribution of all node ages may be complex, depending on the locations of the calibration nodes.)

**kappa_gamma** = 6 2 specifies the shape and scale parameters ($\alpha$ and $\beta$) in the gamma prior for parameter $\kappa$ (the transition/transversion rate ratio) in models such as K80 and HKY85. This has no effect in models such as JC69, which does not have the parameter. Note that the gamma distribution with parameters $\alpha$ and $\beta$ has the mean $\alpha/\beta$ and variance $\alpha/\beta^2$. Those variables have no effect when usedata = 2.

**alpha_gamma** = 1 1 specifies the shape and scale parameters ($\alpha$ and $\beta$) in the gamma prior for the shape parameter for gamma rates among sites in models such as JC69+$\Gamma$, K80+$\Gamma$ etc. The gamma model of rate variation is assumed only if the variable **alpha** is assigned a positive value. This prior has no effect when usedata = 2.

**rgene_gamma** = 2 2 specifies the shape and scale parameters ($\alpha$ and $\beta$) in the gamma prior for the overall rate parameter $\mu$. Under the global-clock model (clock=1), the independent-rates model (clock = 2), and also the correlated-rates model (clock = 3), $\mu$ is the overall rate on the tree specified by the prior. In the example, $\mu$ has the prior mean 2/2 = 1, that is, one change per site per time unit. If one time unit is 100MY, this means an overall average rate of $10^{-8}$ substitutions per site per year. The variance ($2/2^2 = 0.5$ in the example) of this gamma prior specifies how confident you are about the overall rate, or how variable the overall rates are among loci. It is not about how variable the rates are among branches or how wrong the clock is.

You need to adjust this prior to suit your data and the chosen time scale. Don't use the default. A pragmatic way of deriving a rough rate estimate (for use as the prior mean) may be to use baseml or codeml under the global clock model (clock = 1), with point calibrations (as in Yoder and Yang 2000). If the same species are included in every locus, it is quite easy to do this. Otherwise it is a bit more complex, and here is a possible procedure. First run mcmctree with usedata = 3, so that the program generates tmp1.txt, tmp1.ctl, and tmp1.trees. The tree is unrooted. Edit the tree to make it rooted by adding a pair of parentheses. Edit the tree to add one or two point calibrations. If a node has fossil bounds '>0.6<0.8', you can use '=0.7'. Edit tmp1.ctl to add a line   clock = 1, and change the value of getSE from 2 to 0. Then run baseml tmp1.ctl and look at the result file out to get the rate. Do the same thing for another locus. This way you will get an idea about the overall rates among loci and how variable they are among loci, information useful for specifying the prior.

**sigma2_gamma** = 1 1 specifies the shape and scale parameters ($\alpha$ and $\beta$) in the gamma prior for parameter $\sigma^2$, which specifies how variable the rates are across branches. This prior is used for the two variable-rates models (clock = 2 or 3), with a larger $\sigma^2$ indicating more variable rates (Rannala and Yang 2007). If clock = 1, this prior has no effect.

In the independent-rates model (clock = 2), rates for branches are independent variables from a log-normal distribution (Rannala and Yang 2007: equation 9).

$$f(r \mid \mu, \sigma^2) = \frac{1}{r\sqrt{2\pi\sigma^2}} \exp\left\{ -\frac{1}{2\sigma^2}\left[ \log(r/\mu) + \tfrac{1}{2}\sigma^2 \right]^2 \right\}, \quad 0 < r < \infty. \qquad (1)$$

Here $\sigma^2$ is the variance in the logarithm of the rates. The rate $r$ has mean $\mu$ and variance $(e^{\sigma^2} - 1)\mu^2$.

The correlated-rates model (clock = 3) specifies the density of the current rate $r$, given that the ancestral rate time $t$ ago is $r_A$, as

$$f(r \mid r_A, t\sigma^2) = \frac{1}{r\sqrt{2\pi t\sigma^2}} \exp\left\{-\frac{1}{2t\sigma^2}\left(\log(r/r_A) + \tfrac{1}{2}t\sigma^2\right)^2\right\}, \quad 0 < r < \infty \qquad (2)$$

(Rannala and Yang 2007: equation 2). Parameter $\sigma^2$ here is equivalent to $\nu$ in Kishino *et al.* (2001). Thus $r$ has mean $r_A$ and variance $(e^{t\sigma^2} - 1)r_A^2$.

Note that $\sigma^2$ (clock = 2) or $t\sigma^2$ (clock = 3) is not the variance of the rate; it is the variance of the logarithm of the rate.

The log normal distribution is not scale-invariant. When one changes the time unit from 10MY to 100MY (so that the rate, per time unit, is 10 times as large), and changes the priors on times and rates accordingly, one may hope that the posterior estimates stay the same. This is not the case. The log normal may look reasonable on one time scale and not on the other. It is advisable to plot the log-normal density using the prior mean for $\sigma^2$ and make sure it is not unreasonable.

**finetune.** The following line in the control file

```
finetune = 0.04 0.2 0.3 0.1 0.3 * time, rate, mixing, paras, RateParas
```

specifies the step lengths used in the proposals in the MCMC algorithm. These proposals (a) change the divergence times, (b) change the rates, (c) perform the mixing step (page 225 in Yang and Rannala 2006), (d) change parameters in the substitution model (such as $\kappa$ and $\alpha$ in HKY+$\Gamma$), and (e) change parameters in the rate-drift model (for clock = 2 or 3 only). You have to adjust those finetune parameters manually, by looking at the screen output.

```
-20% 0.33 0.01 0.25 0.00 0.00  1.022 0.752 0.252 0.458 0.133 0.843 - 0.074 0.787 -95294.7
-15% 0.33 0.01 0.25 0.00 0.00  1.021 0.751 0.253 0.457 0.130 0.841 - 0.067 0.783 -95295.4
-10% 0.33 0.00 0.26 0.00 0.00  1.022 0.752 0.254 0.458 0.129 0.842 - 0.065 0.781 -95294.6
 -5% 0.33 0.00 0.25 0.00 0.00  1.022 0.751 0.254 0.457 0.128 0.841 - 0.063 0.780 -95292.4
  0% 0.32 0.00 0.25 0.00 0.00  1.022 0.751 0.254 0.457 0.128 0.841 - 0.063 0.780 -95290.2
  2% 0.32 0.00 0.27 0.00 0.00  1.014 0.746 0.253 0.453 0.126 0.833 - 0.059 0.784 -95290.4
```

A few seconds after you start the program, the screen output will look like the above. Scroll back to check that the tree, fossil calibrations and the sequence alignments are read correctly. The output here is generated from a run under the JC model and global clock (clock = 1). The percentage % indicates the progress of the run, with negative values for the burn-in. Then the five ratios (e.g., 0.33 0.01 0.25 0.00 0.00 on the first line) are the acceptance proportions for the corresponding proposals. The optimal acceptance proportions are around 0.3, and you should try to make them fall in the interval (0.2, 0.4) or at least (0.15, 0.7). If the acceptance proportion is too small (say, <0.10), you decrease the corresponding finetune parameter. If the acceptance proportion is too large (say, >0.80), you increase the corresponding finetune parameter. In the example here, the second acceptance proportion, at 0.01 or 0.00, is too small, so you should stop the program (Ctrl-C) and modify the control file to decrease the corresponding finetune parameter (change 0.2 into 0.02, for example). Then run the program again (use the up ↑ and down ↓ arrow keys to retrieve past commands), observe it for 10 seconds or 1 minute and kill it again if the proportions are still not good. Repeat this process a few times until every acceptance proportion is reasonable. This is not so tedious as it may sound.

The finetune parameters in the control file are in a fixed order and always read by the program even if the concerned proposal is not used. In the above example, JC does not involve any substitution parameters, so that the 4th finetune parameter is not used, and the corresponding acceptance proportion is always 0. Also the use of clock = 1 means that there are no parameters in the rate-drift model, which explains why the 5th acceptance proportion is 0 as well.

Note that the impact of the finetune parameters is on the efficiency of the algorithm, or on how precise the results are when the chain is run for a fixed length. Even if the acceptance proportions are too high or too low, reliable results will be obtained in theory if the chain is run sufficiently long. This effect is different from the effect of the prior, which affects the posterior estimates.

**print** = 1 means that samples will be taken in the MCMC and written to disk and the posterior results will be summarized. 0 means that the posterior means will be printed on the monitor but nothing else: this is mainly useful for testing the program.

**burnin** = **2000, sampfreq** = **5, nsample** = **10000.** In the example here, the program will discard the first 2000 iterations as burn-in, and then run the MCMC for 5 × 10000 iterations, sampling (writing to disk) every 5 iterations. The 10000 samples will then be read in and summarized. I think you should take at least 2000 samples.

## Differences between MCMCtree and Multidivtime

Here are a few differences between the two programs.

- Soft bounds are used in mcmctree while hard bounds are in multidivtime.

- MCMCTREE does not use outgroups, and the master species tree supplied by the user should be the rooted tree for the ingroup species only. With multidivtime, one runs estbranches on an unrooted tree for both ingroup and outgroup species and then run multidivtime on the rooted tree for ingroup species only.

- Calibration information is supplied by identifying the node numbers on the rooted tree for ingroup species for multidivtime, while it is supplied as node labels in the rooted tree in mcmctree.

- In multidivtime, the likelihood is calculated using a normal approximation to the branch lengths in the rooted ingroup tree (see Yang 2006, figure 7.10a). In mcmctree, the likelihood is calculated either exactly or approximately for nucleotide sequences, and approximately for amino acid or codon sequences. The approximation in mcmctree is applied to the branch lengths in the unrooted tree of the ingroup species (see Yang 2006, figure 7.10b).

- Both programs use a few gamma priors, but the specification is different, with multidivtime requiring the mean ($m$) and standard deviation ($s$) and mcmctree using the shape ($\alpha$) and scale ($\beta$) parameters. There are related as follows.

$$m = \alpha/\beta, \quad s = \sqrt{\alpha}/\beta,$$

$$\alpha = (m/s)^2, \quad \beta = m/s^2.$$

- The rate-drift model used in multidivtime is the same as the correlated-rates model (clock = 3) in MCMCtree. The following variables are equivalent in the two programs.

| Prior | Multidivtime (`multicntrl.dat`) | MCMCtree (`MCMCtree.ctl`) |
|---|---|---|
| Gamma prior on root rate | `rtrate` & `rtratesd` | `mu_gamma` |
| Gamma prior on $\nu$ or $\sigma^2$ | `brownmean` & `brownsd` | `sigma2_gamma` |
| Gamma prior on root age | `rttm` & `rttmsd` | Absent |

In both programs, a gamma prior is assigned on the root rate (overall rate on the tree), and on parameter $\nu$ or $\sigma^2$. Multidivtime also assigns a gamma prior on the age of the root, which is not used in MCMCtree.

## Approximate likelihood calculation

Thorne *et al.* (1998) suggested the use of the multivariate normal distribution of MLEs of branch lengths to approximate the likelihood function. To implement this approximation, one has to obtain the MLEs of the branch lengths and calculate their variance-covariance matrix or equivalently the matrix of second derivatives of the log likelihood with respect to the branch lengths (this matrix is also called the Hessian matrix). In Thorne's multidivtime package, this is achieved using the program estbranches.

I have implemented this approximation using the option usedata = 3. With this option, mcmctree will prepare three temporary files for each locus and then invoke baseml or codeml to calculate the MLEs of branch lengths and the Hessian matrix. These results are generated in the file rst1 and copied into the file out.BV by mcmctree. The three temporary files for each locus are the control file tmp#.ctl, the sequence alignment tmp#.txt, and the tree file tmp#.trees, where # means the index for the locus. The tree for the locus is generated by mcmctree by pruning the master tree of all species so that only those species present at the locus remain, and by de-rooting the resulting tree. You should not edit this tree file. You can edit the control file tmp#.ctl to use another model implemented in baseml or codeml, and this option should allow you to use amino acid or codon substitution models. The calculation of the Hessian matrix may be sensitive to the step length used in the difference approximation, and it is advisable that you change the variable Small_Diff in the control file tmp#.ctl to see whether the results are stable.

The output file out.BV from usedata = 3 should then be renamed in.BV. This file has one block of results for each locus. If you manually edit the control file tmp#.ctl and then invoke baseml or codeml from the command line (for example, by typing  codeml tmp2.ctl), you will have to manually copy the content of rst1 into in.BV.

With usedata = 2, mcmctree will read the MLEs and Hessian matrix from in.BV and apply the approximate method for calculating the likelihood in the MCMC.

In effect, mcmctree/usedata = 3 performs the function of estbranches and you can manually perform this step by running baseml or codeml externally after the tree file tmp#.ctl is generated. Similarly mcmctree/usedata = 2 performs the function of mlutidivtime.

Models of amino acid or codon substitution are not implemented in the mcmctree program for the exact likelihood calculation. The only way to use those models is through the approximate method (usedata = 3 and 2). If is advisable that you edit the intermediate control file tmp#.ctl to choose the appropriate model of amino acid or codon substitution in the codeml analysis, and then copy the results into the in.BV file. Finally run mcmctree/usedata = 2.

In the description here, a gene or locus means a site partition. For example, since the three codon positions typically have very different rates, different base compositions, etc., you may treat them as separate partitions.

# 10 Miscelaneous notes

Also see the FAQ page, but please note that document is not up to date.

## Analysing large data sets and iteration algorithms

The maximum likelihood method estimates parameters by maximizing the likelihood function. This is multi-dimensional optimisation problem that has to be solved numerically (see Yang 2000a for an exception). PAML implements two iteration algorithms. The first one (`method` = 0) is a general-purpose minimization algorithm that deals with upper and lower bounds for parameters but not general equality or inequality constraints. The algorithm requires first derivatives, which are calculated using the difference approximation, and accumulates information about the curvature (second derivatives) during the iteration using the BFGS updating scheme. At each iteration step, it calculates a search direction, and does a one-dimensional search along that direction to determine how far to go. At the new point, the process is repeated, until there is no improvement in the log-likelihood value, and changes to the parameters are very small. The algorithm updates all parameters including branch lengths simultaneously.

Another algorithm (method = 1) works if an independent rate is assumed for each branch (clock = 0) (Yang 2000b). This algorithm cycles through two phases. Phase I estimates branch lengths with substitution parameters (such as the transition/transversion rate ratio $\kappa$ and the gamma shape parameter $\alpha$) fixed. Phase II estimates substitution parameters using the BFGS algorithm, mentioned above, with branch lengths fixed. The procedure is repeated until the algorithm converges. In phase I of the algorithm, branch lengths are optimized one at a time. The advantage of the algorithm is that when the likelihood is calculated for different values of one single branch length, as is required when that branch length only is optimised, much of likelihood calculations on the phylogeny is the same and can be avoided by storing intermediate results in the computer memory. A cycle is completed after all branch lengths are optimized. As estimates of branch lengths are correlated, several cycles are needed to achieve convergence of all branch lengths in the tree, that is, to complete phase I of the algorithm.

If branch lengths are the only parameters to be estimated, that is, if substitution parameters are fixed, the second algorithm (method = 1) is much more efficient. Thus to perform heuristic tree search using stepwise addition, for example, you are advised to fix substitution parameters (such as $\kappa$ and $\alpha$). The second algorithm is also more efficient if the data contain many sequences so that the tree has many branch lengths.

**Tip**: To get good initial values for large data sets of protein coding DNA sequences, you can use baseml. Add the options characters "GC" at the end of the first line in the sequence data file. Then run the data with baseml. In the result file generated by baseml (say mlb), look for "branch lengths for codon models" and copy the tree with branch lengths into the tree file. Then run codeml and choose "1: initial values" when asked about what to do with the branch lengths in the tree.

## Tree search algorithms

One heuristic tree search algorithm implemented in `baseml`, codonml and `aaml` is a divisive algorithm, called "star-decomposition" by Adachi and Hasegawa (1996b). The algorithm starts from either the star tree (`runmode` = 2) or a multifurcating tree read from the tree structure file (`runmode` = 1). The algorithm joins two taxa to achieve the greatest increase in log-likelihood over the star-like tree. This will reduce the number of OTUs by one. The process is repeated to reduce the number of OTUs by one at each stage, until no multifurcation exists in the tree. This algorithm works either with or without the clock assumption. The stepwise addition algorithm is implemented

with the option `runmode` = 3. Options `runmode` = 4 or 5 are used for nearest neighbor interchanges , with the intial tree determined with stepwise addition under the parsimony criterion (`runmode` = 4) or read from the tree structure file (`runmode` = 5). The results are self-explanatory.

Besides the fact that ML calculations are slow, my implementations of these algorithms are crude. If the data set is small (say, with <20 or 30 species), the stepwise addition algorithm (`runmode` = 3) appears usable. Choose `clock` = 0, and `method` = 1 to use the algorithm that updates one branch at a time, and fix substitution parameters in the model (such as $\kappa$ and $\alpha$) so that only branch lengths are optimized. Parameters $\kappa$ and $\alpha$ can be fixed in the tree search using `fix_kappa` and `fix_alpha` in the control files. Other parameters (such as substitution rates for genes or codon positions or site partitions) cannot be fixed this way; they can instead be specified in the file of initial values (`in.baseml` or `in.codeml`). Suppose you use a candidate tree to estimate branch lengths and substitution parameters with `runmode` = 0. You can then move the substitution parameters (but not the branch lengths) into the file of initial values. You then change the following variables for tree search: `runmode` = 3, `method` = 1. The program will use the substitution parameters as fixed in the tree search, and optimizes branch lengths only. It is important that the substitution parameters are in the right order in the file; so copy-and-paste from PAML output is probably the safest. It is also important that you do not change the parameter specifications in the control file; the control file should indicate that you want to estimate the substitution parameters, but when the program detects the file of initial values, fixed parameter values are used instead.

## Generating bootstrap data sets

To generate bootstrap pseudo-samples from your original data, you should use the control variable `bootstrap` in the control files `baseml.ctl` or `codeml.ctl`, and specify the number of samples, as follows.

```
bootstrap = 100   * generate bootstrap data sets
```

This generates a file named boot.txt. The file name is hard coded in the programs so you might want to rename it. Note that the way bootstrap samples are generated depends on your model, so you use baseml to generate samples for nucleotide-based analysis and codeml for amino acid and codon-based analysis. If the data are partitioned (using option G), the programs use stratified sampling to generate bootstrap samples, preserving the number of sites in each partition.

The bootstrap samples can be analyzed using phylip programs (choose the option for multiple data sets) and `baseml` or `codeml` (using the variable `ndata` in the control file).

## The rub file recording the progress of iteration

If you use a large value for the variable noisy (say >2), the programs `baseml` and `codeml` will log output to the screen, indicating the progress of the iteration process, i.e., the minimization of the negative log-likelihood. They will also print in the `rub` file, the size (norm) of the gradient or search direction (*h*), the negative log likelihood, and the current values of parameters for each round of iteration. A healthy iteration is indicated by the decrease of both *h* and the negative log likelihood, and *h* is particularly sensitive. If you run a complicated model hard to converge or analyzing a large data set with hundreds or thousands of sequences, you may switch on the output. You can check this file to see whether the algorithm has converged. A typical symptom of failure of the algorithm is that estimates of parameters are at the preset boundaries, with values like 2.00000, 5.00000. When `method` = 1, the output in the `rub` file lists the log likelihood and parameter estimates only.

## Specifying initial values

You may change values of parameters in the control file such as kappa, alpha, omega, etc. to start the iteration from different initial values. Initial values for the second and later trees are determined by the program, and so you do not have much control in this way.

You can collect initial values into a file called `in.baseml` if you are running `baseml` or `in.codeml` if you are running `codeml`. This file should contain as many numbers, separated by white spaces, as the number of parameters that are being optimized by the program. So if the program is estimating 56 parameters (say 51 branch lengths, 1 kappa, and 5 other parameters from the ω distribution), you should put 56 numbers in the file. The parameters are ordered internally in the program and you have no control of the ordering. Nevertheless, the order is the same as in the main output (below the lnL line for each tree). One way of generating the in.codeml or in.baseml files is to run a data set, and then copy initial values from the rub file or from the main output file. The `rub` file records the iteration process and has one line for each round of iteration. Each line lists the current parameter values after the symbol x; you can copy those numbers (not the symbol x) into the file of initial values, and if you like, change one or a few of the parameter values too. When you run the program, look at lnL0 printed out on the screen and check that it is the same as recorded in rub.

When the program runs, it checks to see whether a file of initial values exists, and it does, the program will read initial values from it. This may be useful if the iteration is somehow aborted, and then you can collect current values of parameters from the file `rub` into this file of initial values, so that the new iteration can have a better start and may converge faster. The file of initial values may also be useful if you experience problems with convergence. If you have already obtained parameter estimates before and do not want the program to re-estimate them and only want to do some analysis based on those estimates such as reconstructing ancestral sequences, insert -1 before the initial values.

Warning: A complication is that in some models a transformation is applied during the iteration while the printout uses the original variables. Examples of this are the frequency/proportion parameters for base frequencies (nhomo = 1 in baseml), proportions of site classes in the NSsites models (except for models always having only two classes in which case no transformation is applied), and times or node ages in clock models (clock = 1, 2, 3, 5, 6, but not 0). For those models, you can see that the line of output in the main output file looks different from the last line of rub after the iteration finishes. In the file of initial values, if you use -1 at the start, the program assumes the original variables, while if you don't, the program assumes transformed variables.

## Fine-tuning the iteration algorithm

The iteration algorithm uses the difference approximation to calculate derivatives. This method changes the variable ($x$) slightly, say by a small number e, and see how the function value changes. One such formula is $df/dx = [f(x + e) − f(x)]/e$. The small number $e$ should be small to allow accurate approximation but should not be too small to avoid rounding errors. You can change this value by adding a line in the control files `baseml.ctl` or `codeml.ctl`

```
Small_Diff = 1e-6
```

The iteration is rather sensitive to the value of this variable, and reasonable values are between 1e-5 and 1e-7. This variable also affects the calculation of the SE's for parameters, which are much more difficult to approximate than the first derivatives. If the calculated SE's are sensitive to slight change in this variable, they are not reliable.

If you compile the source codes, you can also change the lower and upper bounds for parameters. I have not put these variables into the control files (See below).

## Adjustable variables in the source codes

This section is relevant only if you compile the source codes yourself. The maximum values of certain variables are listed as constants in uppercase at the beginning of the main programs (baseml.c, basemlg.c, codeml.c). These values can be raised without increasing the memory requirement by too much.

>     NS: maximum number of sequences (species)
>     LSPNAME: maximum number of characters in a species name
>     NGENE: maximum number of "genes" in data of multiple genes (option G)
>     NCATG: maximum number of rate categories in the (auto-) discrete-gamma model (baseml.c, codeml.c)

You can change the value of LSPNAME. Other variables that may be changed include the bounds for parameters, specified at the beginning of the function testx or SetxBound in the main programs (baseml.c and codeml.c). For example, these variables are defined in the function SetxBound in codeml.c:

```
double tb[]={.0001,9}, rgeneb[]={0.1,99}, rateb[]={1e-4,999};

double alphab[]={0.005,99}, rhob[]={0.01,0.99}, omegab[]={.001,99};
```

The pairs of variables specify lower and upper bounds for variables (tb for branch lengths, rgeneb for relative rates of genes used in multiple gene analysis, alphab for the gamma shape parameter, rhob for the correlation parameter in the auto-discrete-gamma model, and omegab for the $d_N/d_S$ ratio in codon based analysis.

## Using PAML with other phylogenetic programs

### PHYLIP

**Sequence data file**. There are some incompatibilities between the PHYLIP format used by PAML programs and the PHYLIP format used by the current version of Joe Felsenstein's PHYLIP package. First, in Phylip, the sequence name can have at most 10 characters, while PAML uses 30 characters. This difference exists all the time and is due to my need to use longer names sometimes ago. If you want the sequence data file to be readable by both PHYLIP and PAML, you should limit the number of characters in the name to 10 and also separate the name from the sequence by at least two spaces. Having two spaces at the end of the name will inform PAML programs that the name has finished. Second, the "interleaved" format is specified by toggling the menu in PHYLIP programs while by a letter I on the first line inside the sequence data file. The latter was the option used by earlier versions of PHYLIP. I have not followed the change since in general PAML does not use command-line menus as PHYLIP programs do. If you use the sequential format, the same file can be read by both programs. You can even use sequential format with the whole sequence on one line.

**Tree file**. Many PHYLIP programs output the estimated trees in a file called treefile. This uses the parenthesis notation and the file should be directly useable in PAML. Or you can copy the trees into a file and add the number of trees at the beginning of the file for use in baseml, codeml, or pamp.

**Distance matrices and neighbour**. baseml and codeml produce distance matrices. They are printed into separate files with names like 2ML.t, 2ML.dS, 2NG.dS, etc. Those files use the lower-diagonal format and are directly readable by the neighbour program in PHYLIP, so you can use the program to make a neighbour-joining tree (Saitou and Nei 1987). You can rename the file as infile

or type in the file name when prompted by neighbour. Then type L to tell the program that the matrix if lower-diagonal.

## PAUP, MacClade, and MrBayes

**Sequence data file**. PAUP, MacClade and MrBayes use the so-called NEXUS file format. PAML programs (mainly baseml and codeml) have some limited support for this format and can read the sequence alignment in that format. Only the sequence alignment is read and the command blocks are ignored. Also PAML does not recognise comments inside the sequence data block, so please avoid them.

The program evolver in the paml package can generate data sets both in the PAML/PHYLIP format and in the PAUP/MrBayes nexus format. You can also modify the file paupblock to add blocks of paup or MrBayes command at the end of each simulated data set. See the descriptions for the evolver program in Chapter

**Tree file**. PAML programs have only limited support with the tree file generated by PAUP or MacClade. First the "[&U]" notation for specifying an unrooted tree is not recognised. For a tree to be accepted as an unrooted tree by PAML, you have to manually modify the tree file and delete a pair of parenthesis so that there is a trifurcation at the root; that is, the outmost pair of parentheses groups together three taxa rather than two, so the tree should be in the format (A, B, C). Thus changing "(((1,2),3),4)" into "((1,2),3,4)" will deroot the tree. Perhaps I should let the program to do this automatically. Second, the "Translate" keyword is ignored by PAML, and it is assumed that the ordering of the sequences in the tree file is exactly the same as the ordering of the sequences in the sequence data file. This seems normally the case if the trees are reconstructed from the same sequence file using paup.

## Clustal

 **Sequence data file**. When you save the clustal alignment in the PHYLIP format with extension .phy, clustal output the alignment using the "interleaved" phylip format, truncating sequence names to 10 characters. This file is typically not readable by PAML programs. You need to make two changes. First add the letter I at the end of the first line, after the number of sequences and the number of sites in the sequence. Second, add spaces between the sequence name and the sequence and make sure there are at least two spaces separating the name and the sequence.

## MEGA

**Sequence data file**. The MEGA sequence data format (Kumar et al. 1994) is different and not directly readable by PAML programs. Need to find out about the format and write something here.

## MOLPHY

 **Sequence data file**. It is possible to prepare the same file to be readable by both MOLPHY programs (nucml and protml) and PAML programs. Need to find out about the format and write something here.

The tree file produced by MOLPHY also uses the parenthesis notation and is readable by PAML programs.

## TreeView

The trees with branch lengths calculated from PAML programs should be directly readable by TreeView (Page 1996). You can copy the tree onto the clipboard and paste into TreeView, or save the tree in a file and read the file from within TreeView. Some of the models implemented in

baseml and codeml require the user to label branches or nodes on the tree, and I found TreeView particularly useful for this purpose when the tree is large.  TreeView shows those labels as node labels.  For example, the free-ratios model in codonml (model = 1) estimates one $\omega$ ratio for each branch.  In the output, codeml prints out a tree with the estimated $\omega$ ratio as node/branch labels, with some notes like "Tree for Rod Page's TreeView".  I can copy this tree into tree view.  Similarly the global and local clock models in baseml and codeml estimate an age for each node, and the output tree from those two programs can be copied into TreeView directly.

Andrew Rambaut's TreeEdit, which runs on MACs, has similar functionalities.  However, I don't have MAC and have no experience with the program.

## Windows notes

***Turn on file extensions in Windows Explorer.***  Windows Explorer by default hides file extensions for known file types.  You should go to "Windows Explorer - Tools - Folder options - View" and un-tick "Hide extensions for known file types", so that you can see the full file names from Windows Explorer.

***Using Task Manger to change job priority.***  Start Task Manger (for example, right click on task bar and choose Task Manager).  Click on Processes button.  Locate the big job, say, codeml.  Right click and Set Priority to Low.  Note that the process running the Command Prompt is cmd.  If you change the priority of cmd to low, all jobs started from that window will run at low priority.  You can change View – Update Speed to Low and change View – Select Columns.  Change Options – Minimize on Use.  Then you can minimize rather than close Task Manger.

***All input and output files are plain text files.***  In the Command Prompt box (Start - Programs – Accessaries – Command Prompt), you can use **type** or **more** to view a text file.  If you see strange characters on the screen and perhaps also hear beeps, the file is not a plain text file.  You can also use a text editor to view and edit a plain text file.  If you use Microsoft Word or Wordpad to save a file, make sure that the files are saved as a plain text file.  Use File – Save As and change the file type.  When you do, Word or other programs might automatically add the file extension .txt, and you will have to rename the file if you don't want the .txt.

**How to change your search path to include paml programs.**  You can create a folder called bin under your account.  The folder name should be shown when you move (cd ) to that folder.  Let's say this is D:\bin.  Copy or move the executable files from the paml/bin/ folder into this folder.

Now change the environment variable path.  On Windows Vista or XP, open Control Panel – System – Advanced System Settings – Environment variables.  Under "User variables for Ziheng", find path, and Edit.  Add the folder name (D:\bin in my example) to the end or beginning of the string.  The fields are separated by semi-colons.  Note that Windows/Dos is case-insensitive, so D:\bin is the same as D:\BIN or d:\bin.

## UNIX/Linux/Mac OSX notes

**How to change your search path to include paml programs.**  You can create a folder bin in your root directory and copy paml executables into that folder, and then include the folder in your search path.  This way you can run the program no matter where you are.  If you have several versions of the paml, you may even name the executables baseml3.15, baseml4, etc., to distinguish them.  Similarly you can copy other programs such as mb, dnadist, etc. into that folder.

Type cd   to move to your root directory.  Create a bin/ folder in your root directory.

mkdir bin

Copy executables such as codeml, baseml, and mcmctree into the bin/ folder.

Then modify your path environment variable to include the bin/ folder in the initialization file for the shell. You can use  more /etc/passwd to see which shell you run. Below are notes for the C shell and bash shell. There are other shells, but these two are commonly used.

1)  If you see /bin/csh for your account in the /etc/passwd file, you are running the C shell, and the intialization file is .cshrc in your root folder. You can use  more .cshrc  to see its content if it is present. Use a text editor (such as emacs, vi, SimpleText, etc.) to edit the file, and insert the following line

    set path = ($path . ~/bin)

The different fields are separated by spaces. Here  .  means the current folder, and ~/ means your root folder, and ~/bin means the bin folder you created, and $path is whatever folders are already in the path.
If the file .cshrc does not exist, you create one, and insert the line above.

2)  If you see /bin/bash in the file /etc/passwd for your account, you are running the bash shell, and the initialization file is .bashrc. Use a text editor to open .bashrc and insert the following line

    PATH=$PATH:./:~/bin/

This changes the environment variable PATH. The different fields are separated by colon : and not space. If the file does not exist, create one.

After you have changed and saved the initialization file, every time you start a shell, the path is automatically set for you.

Note that UNIX is case-sensitive.

# 11  References

Adachi, J., and M. Hasegawa. 1996a. Model of amino acid substitution in proteins encoded by mitochondrial DNA. Journal of Molecular Evolution **42**:459-468.

Adachi, J., and M. Hasegawa. 1996b. MOLPHY Version 2.3: Programs for molecular phylogenetics based on maximum likelihood.  Computer Science Monographs, **28**:1-150.  Institute of Statistical Mathematics, Tokyo.

Anisimova, M., J. P. Bielawski, and Z. Yang. 2001. The accuracy and power of likelihood ratio tests to detect positive selection at amino acid sites. Molecular Biology and Evolution **18**:1585-1592.

Anisimova, M., J. P. Bielawski, and Z. Yang. 2002. Accuracy and power of Bayes prediction of amino acid sites under positive selection. Molecular Biology and Evolution **19**:950-958.

Anisimova, M., R. Nielsen, and Z. Yang. 2003. Effect of recombination on the accuracy of the likelihood method for detecting positive selection at amino acid sites. Genetics **164**:1229-1236.

Azzalini, A., and M. G. Genton. 2007. Robust likelihood methods based on the skew-*t* and related distributions. Int. Statist. Rev. **76**:106-129.

Bielawski, J. P., and Z. Yang. 2004. A maximum likelihood method for detecting functional divergence at individual codon sites, with application to gene family evolution. Journal of Molecular Evolution **59**:121-132.

Bishop, M. J., and A. E. Friday. 1987. Tetropad relationships: the molecular evidence. Pp. 123-139 *in* C. Patterson, ed. Molecules and Morphology in Evolution: Conflict or Compromise? Cambridge University Press, Cambridge, England.

Brown, W. M., E. M. Prager, A. Wang, and A. C. Wilson. 1982. Mitochondrial DNA sequences of primates: tempo and mode of evolution. Journal of Molecular Evolution **18**:225-239.

Chang, B. S., and M. J. Donoghue. 2000. Recreating ancestral proteins. Trends Ecol. Evol. **15**:109-114.

Dayhoff, M. O., R. M. Schwartz, and B. C. Orcutt. 1978. A model of evolutionary change in proteins. Pp. 345-352. Atlas of protein sequence and structure, Vol 5, Suppl. 3. National Biomedical Research Foundation, Washington D. C.

DeBry, R. W. 1992. The consistency of several phylogeny-inference methods under varying evolutionary rates. Molecular Biology and Evolution **9**:537-551.

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

Felsenstein, J. 2004. Inferring Phylogenies. Sinauer Associates, Sunderland, Massachusetts.

Felsenstein, J. 2005. Phylip: Phylogenetic inference program, Version 3.6. University of Washington, Seattle.

Fitch, W. M. 1971. Toward defining the course of evolution: minimum change for a specific tree topology. Systematic Zoology **20**:406-416.

Forsberg, R., and F. B. Christiansen. 2003. A codon-based model of host-specific selection in parasites, with an application to the influenza A virus. Molecular Biology and Evolution **20**:1252-1259.

Galtier, N., and M. Gouy. 1998. Inferring pattern and process: maximum-likelihood implementation of a nonhomogeneous model of DNA sequence evolution for phylogenetic analysis. Molecular Biology and Evolution **15**:871-879.

Goldman, N. 1993. Statistical tests of models of DNA substitution. Journal of Molecular Evolution **36**:182-198.

Goldman, N., and Z. Yang. 1994. A codon-based model of nucleotide substitution for protein-coding DNA sequences. Molecular Biology and Evolution **11**:725-736.

Hartigan, J. A. 1973. Minimum evolution fits to a given tree. Biometrics **29**:53-65.

Hasegawa, M., T. Yano, and H. Kishino. 1984. A new molecular clock of mitochondrial DNA and the evolution of Hominoids. Proc. Japan Acad. B. **60**:95-98.

Hasegawa, M., H. Kishino, and T. Yano. 1985. Dating the human-ape splitting by a molecular clock of mitochondrial DNA. Journal of Molecular Evolution **22**:160-174.

Hayasaka, K., T. Gojobori, and S. Horai. 1988. Hayasaka, K., T. Gojobori, and S. Horai. 1988. Molecular phylogeny and evolution of primate mitochondrial DNA. Molecular Biology and Evolution 5:626-644. Molecular Biology and Evolution **5**:626-644.

Huelsenbeck, J. P., and F. Ronquist. 2001. MrBayes: Bayesian inference of phylogenetic trees. Bioinformatics **17**:754-755.

Inoue, J., P. C. H. Donoghue, and Z. Yang. 2010. The impact of the representation of fossil calibrations on Bayesian estimation of species divergence times. Systematic Biology **59**:74-89.

Jones, D. T., W. R. Taylor, and J. M. Thornton. 1992. The rapid generation of mutation data matrices from protein sequences. CABIOS **8**:275-282.

Jukes, T. H., and C. R. Cantor. 1969. Evolution of protein molecules. Pp. 21-123 *in* H. N. Munro, ed. Mammalian Protein Metabolism. Academic Press, New York.

Kimura, M. 1980. A simple method for estimating evolutionary rate of base substitution through comparative studies of nucleotide sequences. Journal of Molecular Evolution **16**:111-120.

Kishino, H., and M. Hasegawa. 1989. Evaluation of the maximum likelihood estimate of the evolutionary tree topologies from DNA sequence data, and the branching order in hominoidea. Journal of Molecular Evolution **29**:170-179.

Kishino, H., J. L. Thorne, and W. J. Bruno. 2001. Performance of a divergence time estimation method under a probabilistic model of rate evolution. Molecular Biology and Evolution **18**:352-361.

Koshi, J. M., and R. A. Goldstein. 1996. Probabilistic reconstruction of ancestral protein sequences. Journal of Molecular Evolution **42**:313-320.

Kumar, S., K. Tamura, and M. Nei. 1994. MEGA: Molecular Evolutionary Genetics Analysis software for microcomputers. Comput Appl Biosci **10**:189-191.

McCullagh, P., and J. A. Nelder. 1989. Generalized linear models. Chapman and Hall, London.

Messier, W., and C.-B. Stewart. 1997. Episodic adaptive evolution of primate lysozymes. Nature **385**:151-154.

Nei, M., and T. Gojobori. 1986. Simple methods for estimating the numbers of synonymous and nonsynonymous nucleotide substitutions. Molecular Biology and Evolution **3**:418-426.

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

Page, R. D. M. 1996. TreeView: An application to display phylogenetic trees on personal computers. Comput. Appl. Biosci. **12**:357-358.

Pauling, L., and E. Zuckerkandl. 1963. Chemical paleogenetics: molecular "restoration studies" of extinct forms of life. Acta Chem. Scand. **17**:S9-S16.

Pupko, T., I. Pe'er, R. Shamir, and D. Graur. 2000. A fast algorithm for joint reconstruction of ancestral amino acid sequences. Molecular Biology and Evolution **17**:890-896.

Rambaut, A. 2000. Estimating the rate of molecular evolution: incorporating non-comptemporaneous sequences into maximum likelihood phylogenetics. Bioinformatics **16**:395-399.

Rannala, B., and Z. Yang. 1996. Probability distribution of molecular evolutionary trees: a new method of phylogenetic inference. Journal of Molecular Evolution **43**:304-311.

Rannala, B., and Z. Yang. 2007. Inferring speciation times under an episodic molecular clock. Systematic Biology **56**:453-466.

Robinson, D. F., and L. R. Foulds. 1981. Comparison of phylogenetic trees. Math. Biosci. **53**:131-147.

Saitou, N., and M. Nei. 1987. The neighbor-joining method: a new method for reconstructing phylogenetic trees. Molecular Biology and Evolution **4**:406-425.

Self, S. G., and K.-Y. Liang. 1987. Asymptotic properties of maximum likelihood estimators and likelihood ratio tests under nonstandard conditions. J. Am. Stat. Assoc. **82**:605-610.

Shimodaira, H., and M. Hasegawa. 1999. Multiple comparisons of log-likelihoods with applications to phylogenetic inference. Molecular Biology and Evolution **16**:1114-1116.

Stewart, C.-B., J. W. Schilling, and A. C. Wilson. 1987. Adaptive evolution in the stomach lysozymes of foregut fermenters. Nature **330**:401-404.

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

Swanson, W. J., R. Nielsen, and Q. Yang. 2003. Pervasive adaptive evolution in mammalian fertilization proteins. Molecular Biology and Evolution **20**:18-20.

Tamura, K. 1992. Estimation of the number of nucleotide substitutions when there are strong transition-transversion and G+C content biases. Molecular Biology and Evolution **9**:678-687.

Tamura, K., and M. Nei. 1993. Estimation of the number of nucleotide substitutions in the control region of mitochondrial DNA in humans and chimpanzees. Molecular Biology and Evolution **10**:512-526.

Thorne, J. L., H. Kishino, and I. S. Painter. 1998. Estimating the rate of evolution of the rate of molecular evolution. Molecular Biology and Evolution **15**:1647-1657.

Thornton, J. 2004. Resurrecting ancient genes: experimental analysis of extinct molecules. Nat. Rev. Genet. **5**:366-375.

Whelan, S., and N. Goldman. 2001. A general empirical model of protein evolution derived from multiple protein families using a maximum likelihood approach. Molecular Biology and Evolution **18**:691-699.

Whelan, S., P. Liò, and N. Goldman. 2001. Molecular phylogenetics: state of the art methods for looking into the past. Trends Genet. **17**:262-272.

Wong, W. S. W., Z. Yang, N. Goldman, and R. Nielsen. 2004. Accuracy and power of statistical methods for detecting adaptive evolution in protein coding sequences and for identifying positively selected sites. Genetics **168**:1041-1051.

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. 1994a. Maximum likelihood phylogenetic estimation from DNA sequences with variable rates over sites: approximate methods. Journal of Molecular Evolution **39**:306-314.

Yang, Z. 1994b. Estimating the pattern of nucleotide substitution. Journal of Molecular Evolution **39**:105-111.

Yang, Z. 1994c. Statistical properties of the maximum likelihood method of phylogenetic estimation and comparison with distance matrix methods. Systematic Biology **43**:329-342.

Yang, Z. 1995. A space-time process model for the evolution of DNA sequences. Genetics **139**:993-1005.

Yang, Z. 1996a. Maximum-likelihood models for combined analyses of multiple sequence data. Journal of Molecular Evolution **42**:587-596.

Yang, Z. 1996b. Among-site rate variation and its impact on phylogenetic analyses. Trends Ecol. Evol. **11**:367-372.

Yang, Z. 1998. Likelihood ratio tests for detecting positive selection and application to primate lysozyme evolution. Molecular Biology and Evolution **15**:568-573.

Yang, Z. 2000a. Complexity of the simplest phylogenetic estimation problem. Proc. R. Soc. B: Biol. Sci. **267**:109-116.

Yang, Z. 2000b. Maximum likelihood estimation on large phylogenies and analysis of adaptive evolution in human influenza virus A. Journal of Molecular Evolution **51**:423-432.

Yang, Z. 2002. Inference of selection from multiple species alignments. Curr. Opinion Genet. Devel. **12**:688-694.

Yang, Z. 2004. A heuristic rate smoothing procedure for maximum likelihood estimation of species divergence times. Acta Zoologica Sinica **50**:645-656.

Yang, Z. 2006. Computational Molecular Evolution. Oxford University Press, Oxford, England.

Yang, Z. 2007. PAML 4: Phylogenetic analysis by maximum likelihood. Molecular Biology and Evolution **24**:1586-1591.

Yang, Z., and D. Roberts. 1995. On the use of nucleic acid sequences to infer early branchings in the tree of life. Molecular Biology and Evolution **12**:451-458.

Yang, Z., and T. Wang. 1995. Mixed model analysis of DNA sequence evolution. Biometrics **51**:552-561.

Yang, Z., and S. Kumar. 1996. Approximate methods for estimating the pattern of nucleotide substitution and the variation of substitution rates among sites. Molecular Biology and Evolution **13**:650-659.

Yang, Z., and B. Rannala. 1997. Bayesian phylogenetic inference using DNA sequences: a Markov chain Monte Carlo Method. Molecular Biology and Evolution **14**:717-724.

Yang, Z., and R. Nielsen. 1998. Synonymous and nonsynonymous rate variation in nuclear genes of mammals. Journal of Molecular Evolution **46**:409-418.

Yang, Z., and A. D. Yoder. 1999. Estimation of the transition/transversion rate bias and species sampling. Journal of Molecular Evolution **48**:274-283.

Yang, Z., and R. Nielsen. 2000. Estimating synonymous and nonsynonymous substitution rates under realistic evolutionary models. Molecular Biology and Evolution **17**:32-43.

Yang, Z., and J. P. Bielawski. 2000. Statistical methods for detecting molecular adaptation. Trends Ecol. Evol. **15**:496-503.

Yang, Z., and W. J. Swanson. 2002. Codon-substitution models to detect adaptive evolution that account for heterogeneous selective pressures among site classes. Molecular Biology and Evolution **19**:49-57.

Yang, Z., and R. Nielsen. 2002. Codon-substitution models for detecting molecular adaptation at individual sites along specific lineages. Molecular Biology and Evolution **19**:908-917.

Yang, Z., and A. D. Yoder. 2003. Comparison of likelihood and Bayesian methods for estimating divergence times using multiple gene loci and calibration points, with application to a radiation of cute-looking mouse lemur species. Systematic Biology **52**:705-716.

Yang, Z., and B. Rannala. 2006. Bayesian estimation of species divergence times under a molecular clock using multiple fossil calibrations with soft bounds. Molecular Biology and Evolution **23**:212-226.

Yang, Z., and R. Nielsen. 2008. Mutation-selection models of codon substitution and their use to estimate selective strengths on codon usage. Molecular Biology and Evolution **25**:568-579.

Yang, Z., N. Goldman, and A. Friday. 1994. Comparison of models for nucleotide substitution used in maximum- likelihood phylogenetic estimation. Molecular Biology and Evolution **11**:316-324.

Yang, Z., S. Kumar, and M. Nei. 1995a. A new method of inference of ancestral nucleotide and amino acid sequences. Genetics **141**:1641-1650.

Yang, Z., N. Goldman, and A. E. Friday. 1995b. Maximum likelihood trees from DNA sequences: a peculiar statistical estimation problem. Systematic Biology **44**:384-399.

Yang, Z., R. Nielsen, and M. Hasegawa. 1998. Models of amino acid substitution and applications to mitochondrial protein evolution. Molecular Biology and Evolution **15**:1600-1611.

Yang, Z., W. J. Swanson, and V. D. Vacquier. 2000a. Maximum likelihood analysis of molecular adaptation in abalone sperm lysin reveals variable selective pressures among lineages and sites. Molecular Biology and Evolution **17**:1446-1455.

Yang, Z., W. S. W. Wong, and R. Nielsen. 2005. Bayes empirical Bayes inference of amino acid sites under positive selection. Molecular Biology and Evolution **22**:1107-1118.

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

Yoder, A. D., and Z. Yang. 2000. Estimation of primate speciation dates using local molecular clocks. Molecular Biology and Evolution **17**:1081-1090.

Zhang, J. 2004. Frequent false detection of positive selection by the likelihood method with branch-site models. Molecular Biology and Evolution **21**:1332-1339.

Zhang, J., R. Nielsen, and Z. Yang. 2005. Evaluation of an improved branch-site likelihood method for detecting positive selection at the molecular level. Molecular Biology and Evolution **22**:2472-2479.

Zharkikh, A. 1994. Estimation of evolutionary distances between nucleotide sequences. Journal of Molecular Evolution **39**:315-329.

# Index