

User Guide

PAML: Phylogenetic Analysis by Maximum Likelihood

Version 4 (July 2007)

Ziheng Yang

© Copyright 1993 – 2007 by Ziheng Yang.

The software package is provided "as is" without warranty of any kind. In no event shall the author or his employer be held responsible for any damage resulting from the use of this software, including but not limited to the frustration that you may experience in using the package. The program package, including source codes, example data sets, executables, and this documentation, is distributed free of charge for academic use only. Permission is granted to copy and use programs in the package provided no fee is charged for it and provided that this copyright notice is not removed.

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>).

The author can be reached at

Ziheng Yang

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

Fax: +44 (20) 7679 7096

Table of Contents

Table of Contents	2
1 Overview	1
<i>PAML Documentation</i>	<i>1</i>
<i>What PAML Programs Can Do</i>	<i>1</i>
<i>What PAML Programs Cannot Do</i>	<i>2</i>
<i>Organisation of This Manual</i>	<i>3</i>
2 Compiling and Running PAML Programs.....	4
<i>Windows.....</i>	<i>4</i>
<i>UNIX.....</i>	<i>4</i>
<i>Mac OS X.....</i>	<i>5</i>
<i>Files in the Package.....</i>	<i>6</i>
<i>Example Data Sets</i>	<i>7</i>
<i>Which Files Are Needed?</i>	<i>9</i>
3 Input File Formats	10
<i>Sequence Data File Format</i>	<i>10</i>
<i>Sequential and Interleaved Formats</i>	<i>10</i>
<i>Site Pattern Counts</i>	<i>13</i>
<i>Tree File Format and Representations of Tree Topology.....</i>	<i>15</i>
<i>baseml Control File</i>	<i>17</i>
<i>basemlg Control File.....</i>	<i>23</i>
<i>codeml (codonml and aaml) Control File</i>	<i>23</i>
<i>Codon Sequences (seqtype = 1)</i>	<i>24</i>
<i>Amino Acid Sequences (seqtype = 2)</i>	<i>28</i>
<i>evolver.....</i>	<i>29</i>
<i>yn00.....</i>	<i>32</i>
<i>mcmc tree</i>	<i>33</i>
4 Models and Analyses.....	37
<i>General Theory.....</i>	<i>37</i>
<i>Nucleotide Substitution Models</i>	<i>38</i>
<i>Transition/transversion Rate Ratio.....</i>	<i>40</i>
<i>Codon Substitution Models</i>	<i>41</i>
<i>Basic Model</i>	<i>41</i>
<i>Branch Models</i>	<i>42</i>
<i>Site Models</i>	<i>42</i>
<i>Branch-site models.....</i>	<i>46</i>
<i>Clade Models</i>	<i>47</i>
<i>Amino Acid Substitution Models</i>	<i>47</i>
<i>Variable Rates Among Sites.....</i>	<i>48</i>

<i>Models for Combined Analyses of Partitioned Data</i>	50
For Nucleotides (baseml)	50
For Codons (codeml with seqtype = 1)	51
For Amino Acids (codeml with seqtype = 2)	51
<i>Global and Local Clocks, and Sequences With Dates</i>	51
<i>Reconstruction of Ancestral Sequences</i>	52
<i>Analysing Large Data Sets and Iteration Algorithms</i>	53
<i>Tree Search Algorithms</i>	54
<i>Bootstrap Data Sets</i>	54
<i>Simulation</i>	55
5 Technical Notes	56
<i>The rub File Recording the Progress of Iteration</i>	56
<i>Specifying Initial Values</i>	56
<i>Fine-tuning the Iteration Algorithm</i>	57
<i>Adjustable Variables in the Source Codes</i>	57
<i>More Codon Models</i>	58
6 Appendixes	59
<i>Appendix A. Using PAML with Other Phylogenetics Software</i>	59
PHYLIP	59
PAUP, MacClade, and MrBayes	59
Clustal	60
MEGA	60
MOLPHY	60
TreeView	60
<i>Appendix B. Overcoming Windows Annoyances</i>	61
<i>Appendix C. Recent Changes, Since Version 3.15</i>	61
7 References	62
Index	66

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`, with `baseml`, `codeml`, and `evolver` to be the most important ones. 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`).

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.

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".

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.

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.

Organisation of This Manual

Chapter 2 “Compiling and Running PAML programs” explains how to install the programs and how to run the example data sets included in the package to get started. Chapter 3 “Input File Formats” explains the formats of the sequence data file, the tree file. It also goes through the variables in the control files such as `baseml.ctl` and `codeml.ctl`, which you use to specify the model of analysis. Chapter 4 “Models and Analyses” provides background information about the models and analyses implemented. It also mentions the control variables used to implement the models. Chapters 3 and 4 thus constitute the bulk of this manual.

2 Compiling and Running 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 Windows 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 latest 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
```

2. 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
```

3. 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.

Files in the Package

The list is not up to date now, and you probably do not need to read this section. The following is a list of files included in the package, which I prepared some time ago.

Source codes (in the src/ folder):

- baseml.c: various models for nucleotide sequences
- codeml.c: models for codon (seqtype = 1) and amino acid (seqtype = 2) sequences
- pamp.c: parsimony analyses of nucleotide or amino acid sequences
- mcmctree.c: Bayes Markov chain Monte Carlo method on trees
- evolver.c: simulation of sequence data and comparison of trees
- basemlg.c: Nucleotide-based model with (continuous) gamma rates among sites
- yn00.c: Estimation of dN and dS by the method of Yang and Nielsen (2000)
- treesub.c: a few functions
- treespace.c: a few more functions
- tools.c: my toolkit
- tools.h: header file

Compiling commands

- Makefile: make file
- Makefile.UNIX: make file for UNIX/Linux/MAC OSX
- README.txt: compiling commands for GNU gcc, and unix CC compilers

Control files:

- basemlctl: control file for running baseml and basemlg;
- codemlctl: control file for codeml (i.e., codonml and aaml)
- pampctl: control file for pamp
- yn00ctl: control file for yn00
- mcmctreectl: control file for mcmctree

Data files for codeml (see the files for details):

- grantham.dat: amino acid distance matrix (Grantham 1974)
- miyata.dat: amino acid distance matrix (Miyata *et al.* 1980)
- dayhoff.dat: Empirical amino acid substitution matrix of Dayhoff *et al.* (1978)
- jones.dat: Empirical amino acid substitution matrix of Jones *et al.* (1992)
- wag.dat: Empirical amino acid substitution matrix of Whelan and Goldman (in press)
- mtREV24.dat: Empirical amino acid substitution matrix of Adachi and Hasegawa (1996b)
- mtmam.dat: Empirical amino acid substitution matrix for mitochondrial proteins of mammals from Yang *et al.* (1998).

Data files for evolver (see those small files for details):

- MCbase.dat: data file for simulating nucleotide sequences
- MCcodon.dat: data file for simulating codon sequences
- MCaa.dat: data file for simulating amino acid sequences

Example tree files:

`4s.trees`: tree structure file for 4-sequence data

`5s.trees`: tree structure file for 5-sequence data

Documentations:

`readme.txt`: PAML readme file

`PAML.html`: PAML web page, serving also as part of the manual (html file)

`PAMLDOC.pdf`: this document

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.

`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-called “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.

Which Files Are Needed?

You may copy the executables to a directory containing your data files. Please note that the program `codeml` may need some of the data files in the package such as `grantham.dat`, `dayhoff.dat`, `jones.dat`, `wag.dat`, `mtREV24.dat`, or `mtmam.dat`. You should probably copy these files together. Other programs do not need such data files apart from the sequence and tree files you specify in the control file. There should be better ways of managing the multiple files, but I am too lazy and stupid to figure that out.

Note also that 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.

3 Input 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
CGCCACGGACTTACATCCTCATTACTATT
sequence 2
AAGCTTCACCGGCGCAATTATCCTCATAAT
CGCCACGGACTTACATCCTCATTATTATT
sequence 3
AAGCTTCACCGGCGCAGTTGTTCTTATAAT
TGCCACGGACTTACATCATCATTATTATT
sequence 4
AAGCTTCACCGGCGCAACCACCTCATGAT
TGCCATGGACTCACATCCTCCCTACTGTT
```

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.

```

sequence 1  AAGCTTCACCGGCGCAGTCATTCTCATAAT
CGCCACGGACTTACATCCTCATTACTATT
sequence 2  AAGCTTCACCGGCGCAATTATCCTCATAAT
CGCCACGGACTTACATCCTCATTATTATT
sequence 3  AAGCTTCACC GGCGCAGTTG TTCTTATAAT
TGCCACGGACTTACATCATCATTATTATT
sequence 4  AAGCTTCACCGGCGCAACCACCTCATGAT
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 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

[illegible]

5 1000 G
G 4 100 200 300 400
Sequence 1
TCGATAGATAGGTTTTAGGGGGGGGGGTAAAAAAAAA.....

5 855 GC
human GTG CTG TCT CCT ...

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 $100 + 200 + 40 + \dots + 14 = 440$ 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 $27 = 9 \times 3$. 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 ω 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 ω ratios for branches. The internal branch ancestral to human and gibbon has the ratio ω_1 , while all other branches (with the default label #0) have the background ratio ω_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 Δ , 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-rate clade.

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

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. New versions of [TreeView](#) also allow you to add branch labels in the tree-edit window, but even being able to view the labels is a big help. TreeView however does not recognize or allow labels for tips or tip branches. 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. In the `mcmcree` program implementing Bayes MCMC dating methods, I also use symbols < and > to specify soft bounds on fossil calibration nodes ages, while @ is used to represent the most likely age. So in the following example, the human-chimpanzee divergence is most likely at 6MY and quite unlikely to be outside the (4MY, 10MY) interval.

```
((gorilla, (human, chimpanzee) '>.04 @0.06 <.10'), orangutan) '>.12 <.30';
```

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.

baseml Control File

The default control file for baseml is baseml.ct1, 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). More details are given later in the Models and Methods section. Choose 0 if option G is not used in the data file.

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 **clock** = 1, 2, or 3, a rooted tree should be used. **clock** = 1 means the global clock, with all branches having the same rate. If fossil calibration information is specified in the tree file using the symbol @, the

absolute rate will be calculated. Multiple calibration points can be specified this way. If sequences have dates, this option will fit Andrew Rambaut's TipDate model. `clock = 2` implements local clock models of Yoder and Yang (2000) and Yang and Yoder (2003), which assume that branches on the tree can be partitioned into several rate groups. The default is group 0, while all other groups have to be labeled using branch/node labels (symbols # and \$) in the tree. The program will then estimate those rates for branch groups. `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 the section above on "Tree file format" about how to specify fossil calibration information in the tree, how to label branch groups. 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.

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).

fix_rho and `rho` work in a similar way and concern independence or correlation of rates at adjacent sites, where ρ (`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 (π_T , π_C and π_A ; π_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 π_{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 π_{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 κ is assumed and estimated for all branches, while `fix_kappa = 0` means one κ is estimated for each branch.

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

getSE 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 = 1 also works with `runmode = 0` only. For models of variable rates across sites, the program will calculate rates for sites along the sequence (output in the file `rates`) and performs marginal ancestral reconstruction (output in `rst`). For models of one rate for all sites, `RateAncestor = 1` does both marginal and joint ancestral sequence reconstruction (Yang et al. 1995a). The program lists results site by site. You can also use the variable `verbose` to control the amount of output. If you choose `verbose = 0`, the program will list the best nucleotide at each node for the variable sites only and results for constant sites are suppressed. If `verbose = 1`, the program will list all sites for the best nucleotide at each node. If `verbose = 2`, the program also lists the full posterior probability distribution for each site at each ancestral node (for marginal 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.ct1`. 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.ct1` 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 (κ) for the three codon positions. Ancestral reconstruction from such a nucleotide substitution should be very similar to codon-based reconstruction. (Thanks to Belinda Change for many useful suggestions.)

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. $\ln(L_{\max})$ 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 matrix, readable by the NEIGHBOR program in Felsenstein'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.E.s of the common parameters measure the reliability of the estimates. For example, $(\kappa - 1)/SE(\kappa)$, when κ 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 REL 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).

`basemlg` Control File

`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`.

`codeml` (`codonml` and `aaml`) 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:Flx4, 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.

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 ω 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 (α = 0) or under the gamma model of rates for sites ($\alpha \neq 0$). In the latter case, the continuous gamma is used and the variable `ncatG` is ignored. (With `runmode` = 0, the discrete gamma is used.)

model concerns assumptions about the ω ratios among branches (Yang 1998; Yang and Nielsen 1998). `model` = 0 means one ω ratio for all lineages (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 ratio (ω_{k-1} if you have k ratios in total) at the value of `omega` specified in the file. This option can be used to test, for example, whether the ratio for a specific lineage is significantly different from one. See the readme file in the `examples/lysozyme/` folder and try to duplicate the results of Yang (1998).

NSsites specifies models that allow the d_N/d_S ratio (ω) to vary among sites (Nielsen and Yang 1998; Yang et al. 2000b). `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 ω distribution under some models. The values of `ncatG` used to perform analyses in that paper are 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). This means M8 will have 11 site classes (10 from the beta distribution plus 1 additional class). The posterior probabilities for site classes as well as the expected ω values for sites are listed in the file `rst`, which may be useful to pinpoint sites under positive selection, if they exist.

To run several `NSsites` models in one batch, you can specify several models on the same line, as follows:

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

This forces the program to run models M0, M1, M2a, M3, M7, and M8 on the same data set in one go. When more than one `NSsites` model is specified in this way, the number of categories (`ncatG`) used will match those used in Yang *et al.* (2000b), and you do not have any control over it.

The continuous neutral and selection models of Nielsen and Yang (1998) are not implemented in the program.

Version 3.14 introduced some changes to the `NSsites` models M1 and M2. Specifically, the old version of those two models assume a class of conserved sites with $\omega_0 = 0$ while in the modified models, called M1a and M2a, ω_0 is estimated from the data under the constraint $0 < \omega_0 < 1$. Furthermore, the Bayes empirical Bayes (BEB) calculation of posterior probabilities for site classes has been implemented for models M2a (`NSsites` = 2) and M8 (`NSsites` = 8) to replace the old naïve empirical Bayes (NEB) calculation (Yang *et al.* 2005). You are advised to use M1a and M2a to construct an LRT and M7 and M8 to construct an LRT, and use M2a and M8 to identify sites under positive selection. In other words, use CODEML v3.14 or later and do not use earlier versions for site-based analysis. See the section Codon Models in the next Chapter for more details.

Example files for NSsites models: The HIV *env* data set used in Yang *et al.* ((2000b): table 12) is included in the PAML/examples/hivNSsites folder. The abalone sperm lysin data set was analyzed by Yang, Swanson and Vacquier (2000a) using several `NSsites` models. This data set is included in the examples/ folder as well. Also the lysozyme data set, included in the examples/ folder, was analyzed by Yang and Nielsen (2002) using a few `NSsites` models.

The *branch-site* model A (Yang and Nielsen 2002; Yang *et al.* 2005; Zhang *et al.* 2005) is specified by

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

This is the alternative model for the branch-site test of positive selection, or test 2 in Zhang (2005). 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
```

Use d.f. = 1 for the likelihood ratio test, although this tends to make the test conservative. The BEB procedure for calculating probabilities of site classes is implemented for the branch-site model A.

The above is a description of the model and test after modifications in Yang *et al.* (2005) and Zhang *et al.* (2005). Our advice is that you use this test and forget about the old tests.

In case you need the old tests, here are the details. Note that those tests are not recommended. The old branch-site model A fixes $\omega_0 = 0$. This is not available in versions since 3.14. You will have to use an earlier version of the program. The old branch-site model B is still in the program and is specified by

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

The null model is the NSsites model 3 (discrete) with 2 site classes, specified as

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

Use d.f. = 2 degrees of freedom for the test. The (“large”) lysozyme data set analyzed in that paper is included in the examples folder in the package. Look at the readme file.

Also Yang *et al.* (2005) and Zhang *et al.* (2005) described a branch-site test 1, and pointed out that it can be significant when the foreground branches are either under relaxed selective constraint or under positive selection. This test uses the modified branch-site model A as the alternative hypothesis, while the null hypothesis is new site model M1a (NearlyNeutral), with d.f. ≈ 2 . Note that we advise the use of test 2, the branch-site test of positive selection.

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

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

See that paper for details. Similarly model A is modified and the BEB procedure is implemented for model A only. See the next chapter for more details.

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.

RateAncestor: Choose 1 if you want to reconstruct ancestral sequences and 0 to avoid the calculation. The output under codon-based models usually shows the encoded amino acid for each codon. The output under "Prob of best character at each node, listed by site" has two posterior probabilities for each node at each codon (amino acid) site. The first is for the best codon. The second, in parentheses, is for the most likely amino acid

under the codon substitution model. This is a sum of posterior probabilities across synonymous codons. In theory it is possible although rare for the most likely amino acid not to match the most likely codon.

Under gamma models of rates for sites, choosing 1 for this variable will also force the program to estimate the substitution rate at each site.

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 *et al.* (1978) (`dayhoff.dat`), Jones *et al.* 1992 (1992) (see (Kishino *et al.* 1990) for the construction), and Whelan and Goldman (2001) (`wag.dat`). The file `mtmam.dat` has a matrix for mitochondrial proteins estimated by maximum likelihood from a data set of 20 mammals (Yang *et al.* 1998). The mtREV24 model of the MOLPHY package (Adachi and Hasegawa 1996b) is also provided (the file `mtREV24.dat`). These two are similar, and the difference is that the former is derived from proteins from mammals only while the latter came from more-diverse species including chicken, fish, frog, and lamprey. Due to differences in the implementation, you may see small differences in log-likelihood values and branch lengths between `aaml` and `protml` in the MOLPHY package. Such differences are normal and you should use the same program to compare different trees. Under the mtREV24 model, the two programs should give almost identical results.

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.

runmode also works in the same way as in `baseml.ctl`. Specifying `runmode = -2` will force 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 comparison ("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`.

evolver

This program has a small naïve menu, which looks 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          * 0,1:seqs or patterns in paml format (mc.paml); 2: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 ω 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 ω 's among sites, or the branch models with different ω s among branches, or the branch-site models with ω 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 λ , death rate μ , sampling fraction ρ , 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 generating a random sequence using the base, amino acid, or codon frequencies. This can now be achieved by putting a sequence in the file `RootSeq.txt`. The sequence cannot have ambiguities or gaps or stop codons. In most simulations, it is wrong to fix the root sequence. For example, 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. If you want to fix the root sequence, you have to make sure that you know what you are doing.

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 κ is estimated for all sequences in the data file and used in subsequent pairwise comparisons. I hope to add an option to allow κ to be estimated for each pair. `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.

mcmctree

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

Since PAML version 3.15, `mcmctree` implements the MCMC algorithm of Yang and Rannala (2006) and 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). Here are a few of them. First, we use soft bounds for fossil calibrations while `multidivtime` uses hard bounds. Second, `mcmctree` does not use outgroups, and the master species tree supplied by the user should be the rooted tree for the ingroup species only. In `estbranches` (the program to run before `multidivtime`), the master tree should be an unrooted tree for both ingroups and outgroups. Third, calibration information is supplied by identifying the node numbers on the tree in `multidivtime`, while it is supplied as part of the tree structure in `mcmctree`. Fourth, 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 can be calculated either exactly or approximately under nucleotide substitution models, while under amino acid and codon substitution models, only approximate calculation is available. The approximation in `mcmctree` is applied to the branch lengths in the unrooted tree of the ingroup species (see Yang 2006, figure 7.10b).

You can read any book on MCMC algorithms, for example, Chapter 5 in Yang (2006) for the basics of MCMC algorithms.

The example file is in the folder `examples/SoftBound/`, which you can use 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 = mtCDNApril23.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

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 = 2 2 .1 * 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 = .2 0.5 0.5 0.1 .9 * times, rates, mixing, paras, RateParas

print = 1
burnin = 10000
sampfreq = 10

```

```
nsample = 10000
```

Fossil calibrations. The rooted tree topology is fixed and assumed known. It is specified in the tree file and should include all species that have data available at any loci. Note that while Jeff Thorne's multidivtime program requires outgroup species, mcmctree does not require or use outgroup species. The tree has to be a rooted master tree for the ingroup species and has to include all species that is present at any locus.

Fossil calibration information is specified in the tree file. The program accepts four kinds of calibration information, explained here using examples. (1) Lower bound (minimal age) is specified as ">0.06", meaning that the node age is at least 6MY. (2) Upper bound (maximal age) is specified as "<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", meaning that the node age is between 6MY and 8MY. (4) The notation ">.06=0.0693<.08" will cause the program to fit a gamma distribution, with the 2.5% and 97.5% percentiles matching the bounds and the mode of the distribution matching the most likely age following the equal sign. Note that the bounds are all soft bounds, in that there is a 2.5% probability that the age is beyond the bound (see figure 2 in Yang, 2006 #2730).

```
((((human, (chimpanzee, bonobo)) >.06=0.0693<.08, gorilla), (orangutan, sumatran))
>.12=0.139<.16, gibbon);
```

In the tree above, two ancestral nodes have calibration information. The human-chimp divergence is between 6MY and 8MY, with a most likely date of 7MY (or 6.93MY). The program will then fit a gamma distribution to describe the uncertainties in the node age. Similarly the orang-utan divergence time is specified to be between 12MY and 16MY, with a most likely date ~14MY. A gamma distribution is fitted to this information as well.

You can include the calibration information in quotation marks so that the tree is readable in TreeView, as follows. You can then click on the "internal labels" button to make sure that the information is entered correctly.

```
((((human, (chimpanzee, bonobo)) '>.06=0.0693<.08', gorilla), (orangutan, sumatran))
'>.12=0.139<.16', gibbon);
```

For a sensible analysis, one should have at least one lower bound and at least one upper bound on the tree, even though the lower and upper bounds may not be placed on the same node. We also suggest that there should be an upper bound on the root age.

Description of the variables

If **seed** is given a negative number, a random seed is used determined from the current clock time. Different runs will start from different places. If you use a positive number, that number will be used as the seed, so that running the same program multiple times using the same control file will produce the same results.

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 mtprim 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 the sequence data will not be used in the MCMC. This is useful for testing the program or simulating the prior. 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. This can be used to analyze nucleotide, amino acid, and codon sequences, using nucleotide, amino acid, and codon substitution models, respectively. More details are provided in the next subsection: “Approximate likelihood calculation”.

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.

model, alpha, ncatG are used to specify the nucleotide substitution model. These are the same variables as used in `baseml.ctl`. If $\alpha \neq 0$, the program will assume a gamma-rates model, while $\alpha = 0$ means that the model of one rate for all sites will be used.

BDparas = 2 2 .1. This line specifies the three parameters in the prior for divergence times: the per-lineage birth rate λ , the per-lineage death rate μ , and the sampling fraction ρ . The prior is specified by a birth-death process with species sampling (Yang and Rannala 1997). The three parameters affect the shape of the tree and potentially the divergence times (see fig. 7.12 in Yang 2006).

finetune. The 5 finetune parameters are step lengths used in the five proposals in the MCMC; these proposals (a) change the species divergence times, (b) change the rates, (c) performs the mixing step, (d) change substitution parameters (such as κ and α in HKY+ Γ), and (e) change parameters in the rate-drift model (for clock = 2 or 3 only). When you run the program, it prints out four or five percentages on the screen. These are the acceptance proportions for the proposals. You should change the finetune variables so that the corresponding acceptance proportions lie between 10% and 90%, or preferably between 15% to 80%. If the acceptance rate is too small, make the finetune variables smaller.

The variables `rgene_gamma` specifies the gamma prior for the rate for each gene. The gamma distribution is specified by two parameters α and β , so that the distribution has mean α/β variance α/β^2 . Similarly a gamma distribution is assigned to the variance parameter `sigma2_gamma` (σ^2) in the variable-rates models (clock = 2 or 3). The larger σ^2 is, the more variable the rate is among lineages. If clock = 1, the prior on σ^2 has no effect.

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. The specification here means that 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.

Approximate likelihood calculation

Thorne et al. (1998) suggested the use of 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 by 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 to see whether the results are stable.

The output file `out.BV` 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`.

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

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.

In sum, *mcmctree/usedata = 3* is similar to *estbranches*, while *mcmctree/usedata = 2* is similar to *mltidivtime*.

Analysis under amino acid or codon substitution models

As mentioned above, models of amino acid or codon substitution are not implemented in the *mcmctree* program for (exact) likelihood calculation. The only way to use those models is through the approximate method (`usedata = 3` and `2`). Thus to use those models to analyze amino or codon sequence data, you run *mcmctree/usedata = 3*, which calls *codeml* to calculate

the MLEs and variance-covariance matrix of the branch lengths. You may want to edit the intermediate control file to use the appropriate model in the codeml analysis. Afterwards run `mcmctree/usedata = 2` to run the MCMC, and the program will calculate the likelihood using the normal approximation.

4 Models and Analyses

General Theory

This chapter provides some background information about the models implemented in the programs in the PAML package and the kind of analyses that can be performed by the programs. Almost all the models and analyses discussed in this chapter are implemented in the two programs `baseml` and `codeml`. I will briefly mention the control variables used to specify the models under discussion when you run the programs, and you can consult Chapter 3 for more details. The section on simulation is for the program `evolver`.

For a comprehensive and readable account, please see Yang (2006). In summary, `baseml` and `codeml` are maximum likelihood programs. They use numerical optimization algorithms to maximize the log likelihood value under a model you specify to calculate the maximum likelihood estimates of parameters and the corresponding log likelihood. A major use of those models is to test interesting biological hypothesis using the likelihood ratio test.

Maximum likelihood estimates (MLEs) and likelihood ratio tests (LRTs)

MLE: The probability of observing the data X , when viewed as a function of the unknown parameters θ with the data given, is called the *likelihood function*: $L(\theta, X) = f(\theta|X)$. According to the *likelihood principle*, the likelihood function contains all information in the data about parameters θ . The best point estimate of θ is given by the θ that maximizes the likelihood L or the log likelihood $\ell(\theta, X) = \log\{L(\theta, X)\}$. Furthermore, the likelihood curve provides information about the uncertainty in the point estimate.

LRT: Suppose the simpler (null) model has p_0 parameters and the more general (alternative) model has p_1 parameters, and the (optimal) log likelihood values under the two models are ℓ_0 and ℓ_1 . Then twice the log likelihood difference, $2\Delta\ell = 2(\ell_1 - \ell_0)$, has asymptotically a χ^2 distribution with d.f. = $p_1 - p_0$ if the null model is true. So the test statistic $2\Delta\ell$ can be compared with that χ^2 distribution to test whether the null model is rejected against the alternative model.

You can fit two nested models to construct a likelihood ratio test, but this process is not automated, and you have to change the specifications and run the program twice. For example, JC69 and K80 can be compared using a likelihood ratio test, which will be a test of the null

hypothesis that the transition/transversion rate ratio $\kappa = 1$ (JC69) against the alternative $\kappa \neq 1$ (K80). Twice the log likelihood $2\Delta\ell$ should be compared with the χ^2 with d.f. = 1. To conduct this test, you should run `baseml` on the same data set using the same tree once with `model = 0` (for JC69) and another time with `model = 1` (for K80) and retrieve the calculated log likelihood values. You will then do the subtraction and multiplication to get $2(\ell_{K80} - \ell_{JC69})$. You can get the P values by running the program `chi2` included in the package or look up a table of χ^2 critical values.

Markov process models are used to describe substitutions between nucleotides (`baseml` and `basemlg`), codons (`codeml`) or amino acids (`codeml`). The substitution rate can be constant over all sites or assumed to be variable among sites. A *discrete-gamma* model (Yang 1994a) is used in `baseml`, `codonml` and `aaml` to accommodate rate variation among sites, according to which rates for sites come from several (say, four or eight) categories used to approximate the continuous gamma distribution. When rates are variable at sites, the *auto-discrete-gamma* model (Yang 1995) accounts for correlation of rates between adjacent sites.

General assumptions of the models (programs) include the following:

- Substitutions occur independently in different lineages.
- Substitutions occur independently among sites (except for the auto-discrete-gamma model which account for correlated substitution rates at neighboring sites).
- The process of substitution is described by a time-homogeneous Markov process. Further restrictions may be placed on the structure of the *rate matrix* of the process and lead to different substitution models.

The process of substitution is assumed to be stationary. In other words, the frequencies of nucleotides (`baseml`), codons (`codonml`), or amino acids (`aaml`) have remained constant over the time period covered by the data.

The existence of a molecular clock (rate constancy among lineages) is not necessary but can be imposed. Variation (and dependence) of rates at sites is allowed by the discrete-gamma (or auto-discrete-gamma) models implemented in `baseml`, `codonml` and `aaml`.

Nucleotide Substitution Models

Markov process models of nucleotide substitution implemented 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 of these models are given below

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

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

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

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

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

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

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

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

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

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

The element q_{ij} ($i \neq j$) represents the rate of substitution from nucleotide i to j , with the diagonals q_{ii} specified by the mathematical requirement that each row of Q sums to zero. The nucleotides are ordered T, C, A, G. The transition probability matrix over time t is then given as $P(t) = \{p_{ij}(t)\} = \exp(Qt)$, where $p_{ij}(t)$ is the probability that nucleotide i will become

nucleotide j after time t . The sequence data does not permit separation of rate (Q) and time (t), and so Q specifies relative rates only. In the programs, Q is multiplied by a constant so that the average rate of substitution is 1 when the process is in equilibrium. This scaling means that time t , or the branch length in a tree, is measured by the expected number of nucleotide substitutions per site. Q thus represents the pattern of substitution, while the amount of evolution is reflected in time or the branch length. The frequency parameters $\pi_T, \pi_C, \pi_A, \pi_G$ (with the sum to be 1) give the equilibrium distribution of the process for the F81, F84, HKY85, TN93 and REV models. The equilibrium distribution under the JC69 and K80 models has equal frequencies (1/4) for the four nucleotides, while that under T92 is $\pi_T = \pi_A = (1 - \pi_{GC})/2$, $\pi_C = \pi_G = \pi_{GC}/2$, where the GC content π_{GC} is a parameter. Parameters a, b, c, d, e in REV, κ in F84, HKY85 or T92, and κ_1 and κ_2 in TN93 may be termed rate ratio parameters. So the JC69, K80, F81, F84, HKY85, T92, TN93 and REV models contain 0, 1, 0, 1, 1, 1, 2, 5 rate ratio parameters respectively, and 0, 0, 3, 3, 1, 3, 3, 3 frequency parameters respectively. Normally the frequency parameters are estimated using the averages of the observed frequencies, which should be very close to the true maximum likelihood estimates if the assumptions of homogeneity and stationarity are acceptable. Under simple models for a single gene, you can use `nhomo = 1` to estimate the frequency parameters by ML.

Parameter κ in the K80, HKY85 and T92 models is equivalent to α/β in the notation of Kimura (1980) and Hasegawa *et al.* (Hasegawa *et al.* 1985). The present notation is more convenient in a maximum likelihood analysis as the ratio is assumed to be constant for different branches of the tree. F84 is the model implemented in J. Felsenstein's DNAML program. The rate matrix for this model was given by Hasegawa and Kishino (1989), Kishino and Hasegawa (1989), Yang (1994a; 1994b) and Tateno *et al.* (1994). Thorne *et al.* (1992) described the transition probability matrix, and Yang (1994a) and Tateno *et al.* (1994) derived formulae for estimating sequence distances under the model. REV is the general time-reversible process model, also known as GTR (see also Tavaré 1986; Yang 1994b; Zharkikh 1994). It is used in `baseml` only. It seems sufficiently general to enable accurate estimation of the substitution pattern from real data.

For more details about nucleotide substitution models, see review articles by Swofford *et al.* (1996), Lio and Goldman (1998), and Whelan *et al.* (2001).

Transition/transversion Rate Ratio

Unfortunately there are quite a few different definitions of the *transition/transversion rate ratio*. The worst is the ratio of the observed numbers of transitional and transversional differences between two sequences, without correcting for multiple hits, also known as P/Q in Kimura's (1980) notation (see, e.g., (Wakeley 1994)). I suggest that this measure should not be used. The measure used in `baseml` is κ as specified in the above formulas for K80 or HKY95. In Kimura's (1980) notation, $\kappa = \alpha/\beta$. A third measure (R) is the ratio averaged over base frequencies; this is the ratio of the expected number of transitions to the expected number of transversions if one observes the substitution process over time. In Kimura's (1980) notation, $R = \alpha/(2\beta)$. PHYLIP and PAUP* use R and name it the "transition/transversion rate ratio", while κ is referred to in those programs as the "transition/transversion rate parameter".

For a general substitution model without any constraint (the UNREST model in baseml), R is defined as

$$R = \frac{\pi_T q_{TC} + \pi_C q_{CT} + \pi_A q_{AG} + \pi_G q_{GA}}{\pi_T q_{TA} + \pi_T q_{TG} + \pi_C q_{CA} + \pi_C q_{CG} + \pi_A q_{AT} + \pi_A q_{AC} + \pi_G q_{GT} + \pi_G q_{GC}}.$$

Special examples are listed in the following table.

Model	Average transition/transversion rate ratio (R)
JC69 (Jukes and Cantor 1969)	$\frac{1}{2}$
K80 (Kimura 1980)	$\kappa/2$
F81 (Felsenstein 1981)	$(\pi_T \pi_C + \pi_A \pi_G) / (\pi_Y \pi_R)$
F84 (Phylip)	$[\pi_T \pi_C (1 + \kappa / \pi_Y) + \pi_A \pi_G (1 + \kappa / \pi_R)] / (\pi_Y \pi_R)$
HKY85 (Hasegawa et al. 1985)	$(\pi_T \pi_C + \pi_A \pi_G) \kappa / (\pi_Y \pi_R)$
T92 (Tamura 1992)	To be filled in.
TN93 (Tamura and Nei 1993)	$(\pi_T \pi_C \kappa_1 + \pi_A \pi_G \kappa_2) / (\pi_Y \pi_R)$
REV (GTR)	$(\pi_T \pi_C a + \pi_A \pi_G) / (\pi_T \pi_A b + \pi_T \pi_G c + \pi_C \pi_A d + \pi_C \pi_G e)$

Note that the definition of κ under F84 is different than under K80 or HKY85. When transition and transversion rates are equal, $\kappa = 1$ and $R = \frac{1}{2}$ under K80, $\kappa = 1$ and $R = (\pi_T \pi_C + \pi_A \pi_G) / (\pi_Y \pi_R)$ under HKY85, and $\kappa = 0$ and $R = (\pi_T \pi_C + \pi_A \pi_G) / (\pi_Y \pi_R)$ under F84. In general, by forcing R to be identical under HKY85 and F84, one can derive an approximate relationship between κ_{HKY85} and κ_{F84} :

$$\kappa_{\text{HKY}} = 1 + \frac{\pi_T \pi_C / \pi_Y + \pi_A \pi_G / \pi_R}{\pi_T \pi_C + \pi_A \pi_G} \kappa_{\text{F84}}.$$

For K80, both κ and R would serve the same purpose. For F84 and HKY85, κ is easier to use than R ; for example the null hypothesis of equal transition and transversion rates is represented by $\kappa = 0$ under F84 and $\kappa = 1$ under HKY85, while it is rather awkward to specify using R . However, for more complex models such as REV (GTR) or UNREST, it is impossible to define κ while R can be calculated straightforwardly.

Codon Substitution Models

There is now a large collection of codon substitution models. Most of them are discussed in the following review articles (Yang and Bielawski 2000; Yang 2001; Yang 2002). Yang (2001) reviewed the mathematical aspects, while Yang and Bielawski (2000) and Yang (2002) are for biologists. Add some comments about the simulation papers?

Basic Model

Goldman and Yang (1994) suggested a model of codon substitution that is similar to nucleotide-substitution models (especially HKY85) but considers a sense codon as the unit of evolution. The original model used amino acid chemical distances (Grantham 1974) to modify their substitution rate, but the model was found not to fit data well. The commonly used

version now ignores the fact that some amino acids are close to each other chemically while others are very different. This simplified version (Yang et al. 1998) specifies the substitution rate from codon i to codon j as

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

The equilibrium frequency of codon j (π_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`. It forms the basis for more sophisticated models implemented in `codeml`.

The ω 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.

Branch Models

The *branch models* allow the ω 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 ω ratio for each branch. This model is very parameter-rich and its use is discouraged. `model = 2` allows you to have several ω 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.

Site Models

The *site models* allow the ω 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` (use `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 and are recommended for real data analysis. The first pair include M1a (NearlyNeutral) and M2a (PositiveSelection), while the second pair include M7 (beta) and M8 (beta& ω). M1a

(NearlyNeutral) and M2a (PositiveSelection) are slight modifications of models M1 (neutral) and M2 (selection) in (Nielsen and Yang 1998). 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, $0 < \omega_0 < 1$ is estimated from the data while $\omega_1 = 1$ is fixed. The modified models M1a and M2a are described in Wong *et al.* (2004) and Yang *et al.* (2005). In CODEML prior to v3.14, M1 and M2 were implemented, but since 3.14, M1a and M2a have replaced M1 and M2, respectively, so M1 and M2 are not available in v3.14 or later.

Model	NSsites	np	Free parameters
M0 (one ratio)	NSsites = 0	1	ω
M1a (NearlyNeutral): p_0 ($p_1 = 1 - p_0$) $\omega_0 < 1$, $\omega_1 = 1$	NSsites = 1	2	p_0 , $\omega_0 < 1$
M2a (PositiveSelection): p_0 , p_1 ($p_2 = 1 - p_0 - p_1$) $\omega_0 < 1$, $\omega_1 = 1$, $\omega_2 > 1$	NSsites = 2	4	p_0 , p_1 , $\omega_0 < 1$, $\omega_2 > 1$
M3 (discrete): p_0 , p_1 ($p_2 = 1 - p_0 - p_1$) ω_0 , ω_1 , ω_2	NSsites = 3	5	p_0 , p_1 , ω_0 , ω_1 , ω_2
M7 (beta): p , q	NSsites = 7	2	p , q
M8 (beta& ω): p_0 ($p_1 = 1 - p_0$) p , q , $\omega_s > 1$	NSsites = 8	4	p_0 , p , q , $\omega_s > 1$

Also codeml v3.14 applies the constraints $\omega_2 > 1$ for M2a and $\omega_s > 1$ for M8, which seems to remove the problem of multiple local peaks for those two models.

The modifications are based on the following considerations, some of which originated from computer simulation studies. The insistence of a site class with $\omega_1 = 1$ in M2a helps to avoid misclassifying sites under weak purifying selection (with ω slightly less than 1) into the site class of positive selection, as such sites will be lumped into the neutral class. The M1a-M2a pair seems more robust than the M7-M8 pair. If the true null model assumes several classes of conserved sites with $\omega < 1$ as well as neutral sites with $\omega = 1$, the M7-M8 comparison may often be significant, and among half of such cases, the ω_s estimate under M8 will be > 1 , producing false positives. In such cases, the M8a-M8 comparison or M1a-M2a comparison may be more robust. See below on LRTs about description of M8a.

M3 (discrete) is found not to be a good model either for LRT of positive selection or for identification of positive selection sites using NEB (naïve Empirical Bayes; see below). Partly this is because codeml looks at whether an ω estimate is > 1 but not at how much larger it is than 1. So if an estimated ω is 1.12, the program may lump many sites under weak purifying selection (with true ω slightly less than 1) into this class of positively selected sites, producing many false positives. In some simulations, M2a performed better than model M3 even if the data were simulated under M3 (I think there are some examples of this in (Anisimova *et al.* 2002) or (Wong *et al.* 2004).) So even if M3 often fits the data better than all other models, we do not recommend the use of M0-M3 comparison for detecting positive selection nor the use of

M3 and NEB to identify sites under positive selection. Note that the BEB (Bayes Empirical Bayes) procedure for identifying positive selection sites is implemented for M2a and M8 only and not for M3.

You might want to check the original simulation papers cited above for details. Notes, files, and summary programs for such simulations are in the paml release in the folder technical/simulation/codon/. Also look at the small program multiruns useful when you run the same analysis multiple times to guide against convergence problems. We welcome suggestions as to how to break our recommended models (M2a and M8). Note, however, that high rates of intragenic recombination may produce false positives (Anisimova et al. 2003). Strong local variation in the base mutation rate may also be expected to affect these tests.

Testing positive selection using the likelihood ratio test (LRT). We recommend two (almost three) LRTs for testing positive selection. The first test compares M1a against M2a, and the second test compares M7 against M8. See Yang *et al.* (2005) for detailed description of the models. 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 degree of freedom for the chi square in these three tests is unclear. A difficulty is that when the proportion p_2 under M2a or p_1 under M8 is 0, the corresponding ω (ω_2 for M2a or ω_s for M8) is not identifiable. We suggest the use of $df = 2$ for the M1a-M2a and M7-M8 comparisons, which is expected to be too conservative. As pointed out by Swanson *et al.* (2003), the test statistic for the M8a-M8 comparison should be compared with the 50:50 mixture of point mass 0 and χ_1^2 (Self and Liang 1987), so that the critical values are 2.71 at 5% and 5.41 at 1%. Note that the p value for a 50:50 mixture of χ_j^2 and χ_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 χ_1^2 and then half it to get the p value for the mixture distribution. For example, if the statistic is $2\Delta\ell = 2.0$, then $p = 0.157/2 = 0.079$. Wong *et al.* (2004) recommended the use of χ_1^2 based on simulations and also to guide against violations of assumptions.

Identifying sites using the Empirical Bayes method. When the likelihood ratio test is significant, the empirical Bayes method is used to calculate posterior probabilities for site classes. The method we implemented initially (Nielsen and Yang 1998; Yang et al. 2000b) is called the naïve empirical Bayes (NEB). It is naïve as it uses the maximum likelihood estimates of parameters (such as the proportions and ω ratios) but do not account for their sampling errors. This defect is now fixed using a procedure called Bayes empirical Bayes (BEB) implemented for models M2a and M8 (Yang et al. 2005). Both the NEB and BEB calculations are included in the output, and the calculations are automatically performed whenever you specify the site models.

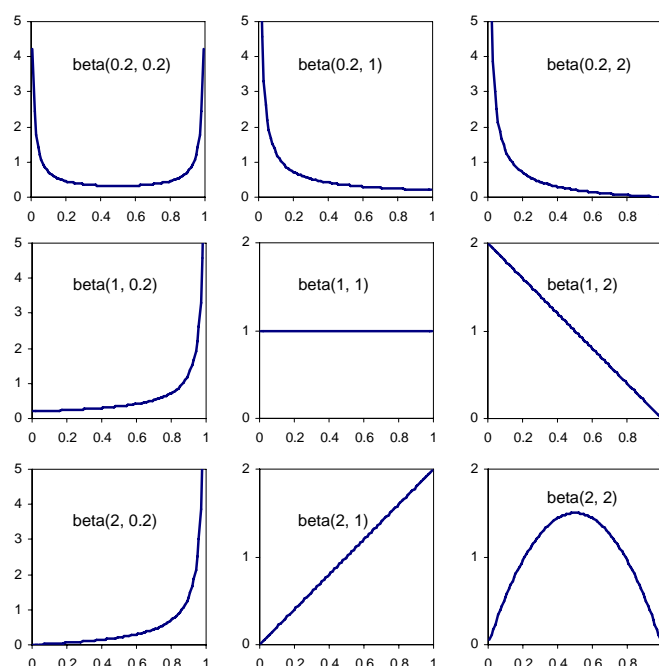
The BEB output has the following format:

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%.

The beta distribution. The beta distribution, $\text{beta}(p, q)$, is a flexible distribution for a variable in the range (0, 1). Below are nine probability densities corresponding to different parameters p and q . The x axis is the dN/dS ratio (ω), and the y axis represents the number or proportion of sites with that dN/dS ratio. So the first plot, for $\text{beta}(0.2, 0.2)$, indicates that most sites are either highly conserved with dN/dS close to 0 or nearly neutral with dN/dS = 1, while few sites are in between. The shape of the beta distribution $\text{beta}(p, q)$ depends on the two parameters p and q . The left side of the curve goes up if $p < 1$ and down if $p > 1$. The right side of the curve goes up if $q < 1$ and down if $q > 1$. Look at the plots below.



Suzuki and Gojobori's (1999) method

Suzuki & Gojobori (1999) proposed a method for testing the effect of selection at individual sites in an alignment of protein coding DNA sequences. With the terminology used here, the method tests whether the ω ratio (dN/dS) is >1 or <1 significantly, which indicates positive and purifying selection, respectively. This method uses parsimony to reconstruct sequences at the ancestral nodes, and then for each site, counts the numbers of synonymous and nonsynonymous differences (Sd and Nd) and the numbers of synonymous and nonsynonymous sites (S and N). It then tests whether the dN/dS 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.

In PAML, a test of this kind is implemented as a by-product of ancestral sequence reconstruction in `codeml` and `baseml`. A difference is that `baseml` and `codeml` use maximum likelihood to reconstruct ancestral sequences (Yang et al. 1995a), while Suzuki & Gojobori used parsimony. Use `RateAncestor = 1`. The output currently goes into the file `rst1`. (I might move it to another place later on.) 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 are noted to 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"). Again the model choice should probably not matter much to ancestral reconstruction, so you can use a simple model such as JC69 (model = 0 Mgene = 0).

I did some test using the data file `abglobin.nuc` in the PAML release. In general, the two implementations produce similar results. At a few sites, the differences may be larger, due to the following reasons.

- `Adaptsite` uses parsimony reconstruction while `codeml` uses a likelihood reconstruction, so the reconstructed ancestral states may be different.
- S&G (1999) uses branch lengths to weight branches when sites (S and N) are counted, with longer branches given higher weights. `adaptsite` uses an NJ algorithm to estimate branch lengths while `codeml` uses a codon model (M0) to do it, so the estimates may be different.
- For codons that are neighbors of stop codons, `adaptsite` and `codeml` count sites differently. For example, for codons TAC and TAT, `adaptsite` counts S = 1 and N = 2, while `codeml` gave 0.429 and 2.571.
- Missing data are handled differently.

Branch-site models

Yang and Nielsen (2002) implemented two models, called A and B, that let the ω ratio vary both among sites and among lineages. The models attempt to detect positive selection that affects only a few sites along a few lineages. The specifications are model = 2 NSsites = 2 for model A and model = 2 NSsites = 3 for model B. There has been a change to model A, as detailed in the table below (Yang et al. 2005; Zhang et al. 2005).

The old model assumes $\omega_0 = 0$ and is unrealistic. This is replaced by $0 < \omega_0 < 1$, estimated from the data. The new model is still called branch-site model A. It can be compared with the new M1a (NearlyNeutral) to form a likelihood ratio test, with d.f. ≈ 2 . This is called test 1. This test can mistake relaxed selective constraint on the foreground branches as positive selection, and so bear in mind that a significant result does not necessarily mean positive selection. Another test, called test 2 or branch-site test of positive selection, uses the same alternative model A but the null model is model A with $\omega_2 = 1$ fixed (use `fix_omega = 1` and `omega = 1` in `codeml.ctl`). The null distribution should be a 50:50 mixture of point mass 0 and χ_1^2 , so that the critical values at the 5% and 1% levels are 2.71 and 5.41, respectively. We recommend use of χ_1^2 (with critical values 3.84 and 5.99) to guide against violations of model assumptions. Test 2 appears to be a robust test of positive selection on the foreground branches and is called the branch-site test of positive selection. We recommend its use. You can forget about the old modes and tests.

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.

Branch site model A: Old and New

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	$(1 - p_0 - p_1) p_0 / (p_0 + p_1)$	$\omega_0 = 0$	$\omega_2 > 1$	$0 < \omega_0 < 1$	$\omega_2 > 1$
2b	$(1 - p_0 - p_1) p_1 / (p_0 + p_1)$	$\omega_1 = 1$	$\omega_2 > 1$	$\omega_1 = 1$	$\omega_2 > 1$

Clade Models

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 (see also Forsberg and Christiansen 2003; Bielawski and Yang 2004). 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 ω distribution: $p_0, p_1, \omega_0, \omega_2$, and ω_3 . The new model C can be compared with the new M1a (NearlyNeutral), which has 2 parameters, with d.f. ≈ 3 .

Clade model C

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	$p_2 = 1 - p_0 - p_1$	ω_2	ω_3	ω_2	ω_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.

Amino Acid Substitution Models

“Empirical” models based on the Dayhoff substitution matrix (model = 2) or its updated version of Jones *et al.* (1992) are constructed using the same strategy. The transition probability matrix over a very short time period such as one PAM, i.e., P(0.01), is used to approximate the matrix of instantaneous rates (Q). The empirical matrices of Dayhoff *et al.* (1978) and Jones *et al.* (1992) were made to satisfy the reversibility condition, that is,

$$\pi_i q_{ij} = \pi_j q_{ji}$$

for any i and j , so that my implementations may be slightly different from that of Kishino *et al.* (1990). These models assume a fixed pattern of amino acid substitution. The package also include an empirical model for globular proteins, the WAG model of Whelan and Goldman

(2001), which is given by the file `wag.dat`, and two similar empirical models for mitochondrial proteins. The first of these is given by the file `mtREV24.dat` and is the mtREV24 model of Adachi and Hasegawa (1996a; 1996b) estimated from a diverse range of species including mammals, chicken, frog, fish, and lamprey. The matrix was estimated by maximum likelihood from real data. The second is given by the file `mtmam.dat` and is estimated from 20 mammalian species using maximum likelihood under the REV model with variable rates among sites (Yang et al. 1998). You can check those files for more details, or if you want to supply your own empirical matrix.

"Mechanistic" models of amino acid substitution requires consideration of both the mutational distance between the amino acids as determined by the locations of their encoding codons in the genetic code table, and the effects that the potential change may have on the structure and function of the protein, which may be related to the physical, chemical and structural differences between amino acids. It seems natural that such a model should be formulated at the level of codons. The program `aaml` implements a few such models, specified by the variable `aaDist`.

Models of variable substitution rates across site (see Yang 1996b for review) are implemented for both nucleotide (`baseml`) and amino acid (`aaml`) sequences. Although the option variables such as `fix_alpha` and `alpha` are also available for codon models (`codonml`), the gamma model for codons is unrealistic as it applies the same gamma rate to both synonymous and nonsynonymous substitutions, with their rate ratio held constant among sites. You are recommended to use the `Nssites` models instead, which assume homogeneous synonymous rates but variable nonsynonymous rates.

Variable Rates Among Sites

Those models assume that the substitution rates are variable among sites according to some statistical distribution. Each site (nucleotide, amino acid, or codon) is assumed to have a rate, which stays constant throughout the tree, so that a fast-evolving site is fast evolving along all lineages and a slowly-evolving site is always slowly-evolving. A commonly used distribution is the gamma distribution (Uzzell and Corbin 1971; Jin and Nei 1990). The gamma distribution has a shape parameter and a scale parameter. To avoid the use of too many parameters, the scale parameter is fixed so that the mean of the distribution is 1 and as a result, the rates for sites are relative rate factors. Thus the shape parameter α measures how variable the rates are among sites. If $\alpha > 1$, the distribution is bell-shaped (\cap), meaning that most sites have rates around 1 while few sites have either very low or very high rates. When $\alpha \rightarrow \infty$, the distribution degenerates into the model of one rate for all sites. When $\alpha \leq 1$, the distribution has a highly skewed L-shape, meaning that most sites have very low rates or are nearly "invariable", but there are some evolutionary "hot spots" with high rates.

Likelihood calculation under the gamma distribution of rates for sites is described in Yang (1993). The algorithm is implemented in the program `baseml`, for nucleotide substitution models JC69 (Jukes and Cantor 1969), K80 (Kimura 1980), F81 (Felsenstein 1981), F84 (Phylip), and HKY85 (Hasegawa et al. 1985). Steel *et al.* (1993) described a way of achieving the same computation using Hadamard matrix transformation. This works for models that are

special cases of Kimura's model of three substitution types (3ST) (1981). There does not seem to be a program implementing this algorithm. The algorithm of Yang (1993) involves very intense computation and baseml is usable for trees of no more than about 6 or 7 sequences. The commonly-used algorithm, called discrete gamma, uses several categories of rates to approximate the continuous gamma (Yang 1994a). This is the version implemented in packages such as PHYLIP/DNAML and PAUP. In baseml and codeml, the model is implemented by specifying a nonzero value for the parameter alpha in the control files: `fix_alpha = 0, alpha = 0.5`, say, means estimating α using maximum likelihood with a starting value of 0.5, while `fix_alpha = 1, alpha = 0.5` means a discrete-gamma model with $\alpha = 0.5$ fixed. The number of categories is specified using `ncatG`. Values such as 5, 4, 8, or 10 are reasonable. Note that the discrete gamma model has one parameter (α), like the continuous gamma model, and the number of categories is not a parameter.

You can test whether the rates are variable among sites by comparing the log likelihood values between two models: the null model of one rate for all sites (`fix_alpha = 1, alpha = 0`) against the alternative gamma model (`fix_alpha = 0`). The df is a 50:50 mixture of point mass at 0 and χ_1^2 . If you use χ_1^2 and the one-rate model is rejected, the same conclusion will be reached if you use the mixture distribution.

Gu *et al.* (1995) extended the gamma model of Yang (1993) to include a class of "invariable" sites with rate 0. This model is somewhat pathological as there is typically a strong correlation between the proportion of invariable sites and the gamma shape parameter. Furthermore, the gamma model with a small α accommodate sites with rates virtually zero. Models involving a proportion of invariable sites are not implemented in PAML programs. They are implemented in PAUP and MrBayes. Note that the α parameter under HKY+G is not comparable with the α parameter under HKY+G+I.

Yang (1995) described a few models that allow the rates to be variable among sites but also correlated over neighbouring sites. The algorithms are then hidden Markov models. The so-called auto-discrete gamma model has two parameters: the gamma shape parameter α and a correlation parameter ρ , which is related to the correlation in rates between two neighbouring sites Yang (1995). This model is implemented in baseml using the specifications: `fix_alpha = 0` and `fix_rho = 0`. The variable `nparK` specifies a few nonparametric models that assume a few rate classes, with a transition probability matrix describing the transition from one rate at a site to another rate at the next site. Felsenstein and Churchill (1996) developed a similar hidden Markov model for variable and correlated rates over sites, as implemented in *phylip/dnaml*. This appears to be a special case of the nonparametric model mentioned above.

The rates at individual sites can also be calculated under those models, using a Bayesian (or more precisely, empirical Bayesian) approach (Yang and Wang 1995). If you choose `RateAncestor = 1`, baseml and codeml will print out such estimated rates into a file named `rst`. Under the continuous gamma model, the posterior mean of the rate at a site should minimize the mean squared error. However, the calculation under the discrete gamma model is more a crude approximation. As we use rate categories and take averages within the category, one can expect that the estimated rates will be too close to 1, as the extremely high and lower rates

disappear after the averaging. Using a large number of categories (say, $\text{ncatG} = 40$) may be helpful if you are interested in calculating such rates.

For a review of models of variable rates among sites, see Yang (1996b) and chapters 13 and 16 of Felsenstein (2004).

Models for Combined Analyses of Partitioned Data

For Nucleotides (baseml)

Several models are described by Yang (1996a) and implemented in programs `baseml` and `codeml` (`codonml` and `aaml`) for analyzing heterogeneous data sets (such as those of multiple genes or different codon positions). The implementation and description below refer to the case of multiple genes, but in the case of nucleotide-based models (`baseml`), the method can be used to analysed data of different codon positions. These models account for different aspects of heterogeneity among the different data sets and are useful for testing hypotheses concerning the similarities and differences in the evolutionary process of different data sets.

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 equivalent to a separate analysis. This 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 assumes 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 (κ 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).

Sequence file	Control file	Parameters across genes
No G	<code>Mgene = 0</code>	everything equal
Option G	<code>Mgene = 0</code>	the same κ and π but different cs (proportional branch lengths)
Option G	<code>Mgene = 2</code>	the same κ , but different π s and cs
Option G	<code>Mgene = 3</code>	the same π , but different κ s and cs
Option G	<code>Mgene = 4</code>	different κ , π s, and cs
Option G	<code>Mgene = 1</code>	different κ , π s, and different (unproportional) branch lengths

The different cs for different genes mean that branch lengths estimated for different genes are proportional. Parameters π represent the equilibrium nucleotide frequencies, which are estimated using the observed frequencies ($\text{nhomo} = 0$). The transition/transversion rate ratio κ 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.

For Codons (codeml with seqtype = 1)

Codon models for multiple genes are described in detail by Yang & Swanson (2002). The following is table 1 of that paper. 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).

Sequence file	Control file	Parameters across genes
No G	Mgene = 0	everything equal
Option G	Mgene = 0	the same (κ , ω) and π but different cs (proportional branch lengths)
Option G	Mgene = 2	the same (κ, ω), but different π s and cs
Option G	Mgene = 3	the same π but different (κ , ω) and cs
Option G	Mgene = 4	different (κ , ω), π s, and cs
Option G	Mgene = 1	separate analysis

For Amino Acids (codeml with seqtype = 2)

See Yang (1996 JME) for similar descriptions for nucleotide models

Sequence file	Control file	Parameters across genes
No G	Mgene = 0	everything equal
Option G	Mgene = 0	the same π but different cs (proportional branch lengths)
Option G	Mgene = 2	different π s and cs
Option G	Mgene = 1	separate analysis

Global and Local Clocks, and Sequences With Dates

PAML (baseml and codeml) implements three ML models regarding rate constancy among lineages. `clock = 0` means no clock and each branch has an independent rate. 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. = $n - 2$.

Between those two extremes are the local clock models, specified by `clock = 2` (Yoder and Yang 2000; Yang and Yoder 2003), which assume that branches in the phylogeny conform with the clock assumption and has the default rate ($r_0 = 1$) except for several pre-defined branches, which have different rates. Rates for branches are specified using branch labels 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.

Both the global-clock (clock = 1) and local-clock (clock = 2) models can accept a single or multiple fossil calibration points, in which case absolute substitution rates will be calculated. You use the symbol @ to specify fossil calibration information in the tree file. See the readme file in the examples/MouseLemurs/ folder for details. Both clock models can be applied to viral sequences with known sequencing dates (Rambaut 2000). You have to use the symbol @ in sequence names to specify the dates of sequence determination. See the readme file in the examples/TipDate/ folder.

The option clock = 3 is for combined analysis of multiple-partition data (multiple genes or multiple codon positions, for example), and allows the branch group rates to vary freely among data partitions. For example, the models allow some branches to be faster-evolving at codon position 1 while they are more slowly-evolving at codon position 2. See Yang and Yoder (2003) and the examples/MouseLemurs/ folder for details.

Reconstruction of Ancestral Sequences

Nucleotides or amino acids of extinct ancestors can be reconstructed using information of the present-day sequences. Parsimony reconstructs ancestral character states by the criterion that the number of changes along the tree at the site is minimized. Algorithms based on this criterion were developed by Fitch (1971) and Hartigan (1973), and are implemented in the program `pamp`. The likelihood approach uses branch lengths and the substitution pattern for ancestral reconstruction. It was developed by Yang *et al.* (1995a) and Koshi and Goldstein (1996) and is implemented in `baseml` for nucleotide sequences and in `aaml` (`codeml.c` with `seqtype = 2`) for amino acid sequences. Results are collected in the file `rst`.

Marginal reconstruction: This approach compares the probabilities of different character assignments to an interior node at a site and select the character that has the highest posterior probability (eq. 4 in Yang *et al.* (1995a)). The algorithm implemented in PAML works under both the model of a constant rate for all sites and the gamma model of rates at sites. If `verbose = 1`, the output will include the full probability distribution at each node at each site.

Joint reconstruction: This approach considers the assignment of a set of characters to all interior nodes at a site as a reconstruction and select the reconstruction that has the highest posterior probability (eq. 2 in Yang *et al.* 1995a). The implementation in PAML now is based on the algorithm of Pupko *et al.* (2000), which gives the best reconstruction at each site and its posterior probability. The algorithm works under the model of a constant rate for sites only and does not work for the gamma model. (It works under models for multiple genes or data partitions as well. My old algorithm looks at alternatives (sub-optimal reconstructions) although it is inefficient and may miss important reconstructions. I have taken that algorithm out, as well as the old option (`RateAncestor = 2`) of allowing the user to specify the reconstruction to be evaluated. If you need those options, let me know.

The marginal and joint approaches use slightly different criteria, and none is better than the other. They are expected to produce very similar 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. Differences may arise when the competing reconstructions have similar probabilities. Since the marginal reconstruction works with models of variable rates among sites, it is recommended for data analysis.

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 κ and the gamma shape parameter α) 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 κ and α). 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 initial 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 κ and α) so that only branch lengths are optimized. Parameters κ and α 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.

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).

Simulation

Computer simulation is a widely used approach to evaluating estimation procedures. In molecular phylogenetics, there are two major methods for simulating sequence data. The first approach samples data at different sites (nucleotide, amino acid, or codon sites) from the multinomial distribution. Under most models of sequence evolution, data at different sites are independently and identically distributed. This approach thus calculates the probability of observing each site pattern, and then sample from sites according to those site pattern probabilities. The number of categories in the multinomial distribution, that is, the number of distinct site patterns, is the number of character states raised to the power of the number of sequences. To simulate nucleotide sequences on a tree of 5 species, the multinomial will have $4^5 = 1024$ categories, and to simulate a pair of codon sequences under the universal code (with 61 sense codons), the multinomial will have $61^2 = 3721$ categories. This approach is faster for simulating data sets on small trees but impractical on large trees as the number of categories may be too large.

A second approach is to generates an ancestral sequence for the root of the tree, and then “evolve” the sequence along the tree according to the specified substitution model and using the specified branch lengths and substitution parameters. The `evolver` program implements this approach. The ancestral sequence is generated according to the equilibrium distribution of the characters, that is, by sampling characters repeatedly according to the equilibrium distribution. The program then evolves the sequence along branches of the tree, according to the transition probabilities calculated for each branch. For site-heterogeneous models, the substitution pattern may be different from site to site and the different sites may have different transition probabilities.

See Chapter 9 in Yang (2006) for a more detailed discussion.

Tips

1. For analyzing multiple simulated data sets, you can copy the tree with branch lengths from `MCbase.dat` or `MCaa.dat` into the tree file to be used by baseml or codeml. You can then use the variable `fix_blength` to let baseml or codeml use the branch lengths in the tree as initial values for the maximum likelihood iteration. This should speed up the iteration since the true parameter values should be good initial values.
2. Programs baseml and codeml output one line of results for each data set in a file named `rst1`. The output typically includes the log likelihood, the estimated substitution parameters but not branch lengths. If you can modify the source codes, you can go into `baseml.c` or `codeml.c` and search for `frst1`, and add or remove output. However, this may require familiarity with the program, especially about how the variables are arranged during the iteration.

5 Technical Notes

This section contains some technical notes for running PAML programs. Also see the FAQ page.

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 if 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.ct1` or `codeml.ct1`

```
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.

More Codon Models

Fcodon = 4 (F1x4MG) and 5 (F3x4MG) implement two codon models in the style of Muse and Gaut (1994). Those models are very similar to the F1x4 and F3x4 models described before except that the substitution rate from codon i to codon j is proportional to the frequency of the target nucleotide rather than the frequency of the target codon. Suppose codon i is the triplet $i_1i_2i_3$, and codon j is the triplet $j_1j_2j_3$. We have to specify the substitution rate for codons i and j that are different at one position (otherwise the rate is 0). Let that position be k ($k = 1, 2, 3$) and let the nucleotide frequency at codon position k be $\pi_{j_k}^{(k)}$. The rate matrix is then

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

For example, the change from $i = \text{TCA}$ to $j = \text{TCG}$ is a synonymous transition, and the rate is $q_{\text{TCA} \rightarrow \text{TCG}} = \kappa \pi_G^{(3)}$, where $\pi_G^{(3)}$ is the frequency of G at position 3. This model is obviously time-reversible, with the equilibrium codon frequency π_j proportional to $\pi_{j_1}^{(1)} \pi_{j_2}^{(2)} \pi_{j_3}^{(3)}$, as under the F3x4 models mentioned before. To see this note that $q_{\text{TCA} \rightarrow \text{TCG}} = \frac{\kappa}{\pi_T^{(1)} \pi_C^{(2)}} \times \pi_T^{(1)} \pi_C^{(2)} \pi_G^{(3)}$

and $q_{\text{TCG} \rightarrow \text{TCA}} = \frac{\kappa \omega}{\pi_T^{(1)} \pi_C^{(2)}} \times \pi_T^{(1)} \pi_C^{(2)} \pi_A^{(3)}$; that the rate q_{ij} can be written in the form $s_{ij} \pi_j$, where

$s_{ij} = s_{ji}$. The detailed balance condition for reversibility follows:

$$\left(\pi_{i_1}^{(1)} \pi_{i_2}^{(2)} \pi_{i_3}^{(3)} \right) \times q_{ij} = \left(\pi_{j_1}^{(1)} \pi_{j_2}^{(2)} \pi_{j_3}^{(3)} \right) \times q_{ji}.$$

Fcodon = 4 (F1x4MG) uses one set of base frequencies for all three codon positions, just like F1x4.

6 Appendixes

Appendix A. Using PAML with Other Phylogenetics Software

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 is 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 click board 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 ω ratio for each branch. In the output, codeml prints out a tree with the estimated ω 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.

Appendix B. Overcoming Windows Annoyances

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 – Accessories – 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.

Appendix C. Recent Changes, Since Version 3.15

Here is a list of recent changes since version 3.15, in two parts: incompatible changes and minor bug fixes. Changes in earlier versions are documented in the file doc/PAMLHistory.txt.

1.

Acknowledgments.

(This is written a few years ago, and have not been updated.) I thank Nick Goldman, Adrian Friday, and Sudhir Kumar for many useful comments on different versions of the program package. I thank a number of users for reporting bugs and/or suggesting changes, especially Liz Bailes, Thomas Buckley, Belinda Chang, Adrian Friday, Nicolas Galtier, Nick Goldman, John Heulsenbeck, Sudhir Kumar, Robert D. Reed, Fransisco Rodriguez-Trelles, John Heulsenbeck, John Mercer, and Xuhua Xia.

7 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.
- 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.
- 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.
- 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.
- Felsenstein, J., and G. A. Churchill. 1996. A hidden Markov model approach to variation among sites in rate of evolution. *Molecular Biology and Evolution* **13**:93-104.
- 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.
- Grantham, R. 1974. Amino acid difference formula to help explain protein evolution. *Science* **185**:862-864.
- Gu, X., Y. X. Fu, and W. H. Li. 1995. Maximum likelihood estimation of the heterogeneity of substitution rate among nucleotide sites. *Molecular Biology and Evolution* **12**:546-557.
- Hartigan, J. A. 1973. Minimum evolution fits to a given tree. *Biometrics* **29**:53-65.
- Hasegawa, M., and H. Kishino. 1989. Confidence limits on the maximum-likelihood estimate of the Hominoid tree from mitochondrial DNA sequences. *Evolution* **43**:672-677.
- 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.
- Huelsenbeck, J. P., and F. Ronquist. 2001. MrBayes: Bayesian inference of phylogenetic trees. *Bioinformatics* **17**:754-755.

- Jin, L., and M. Nei. 1990. Limitations of the evolutionary parsimony method of phylogenetic analysis [Erratum in *Mol. Biol. Evol.* 1990 7:201]. *Molecular Biology and Evolution* **7**:82-102.
- 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.
- Kimura, M. 1981. Estimation of evolutionary distances between homologous nucleotide sequences. *Proceedings of National Academy of Sciences USA* **78**:454-458.
- 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., T. Miyata, and M. Hasegawa. 1990. Maximum likelihood inference of protein phylogeny and the origin of chloroplasts. *Journal of Molecular Evolution* **31**:151-160.
- 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.
- Lio, P., and N. Goldman. 1998. Models of molecular evolution and phylogeny. *Genome Res.* **8**:1233-1244.
- 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.
- Muse, S. V., and B. S. Gaut. 1994. A likelihood approach for comparing synonymous and nonsynonymous nucleotide substitution rates, with application to the chloroplast genome. *Molecular Biology and Evolution* **11**:715-724.
- 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.
- 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-contemporaneous 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.
- Steel, M. A., L. Szekely, P. L. Erdos, and P. J. Waddell. 1993. A complete family of phogenetic invariants for any number of taxa under Kimura's 3ST model. *New Zealand J. Botany* **31**:289-296.
- 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.

- Swofford, D. L., G. J. Olsen, P. J. Waddell, and D. M. Hillis. 1996. Phylogeny Inference. Pp. 411-501 in D. M. Hillis, C. Moritz, and B. K. Mable, eds. *Molecular Systematics*. Sinauer Associates, Sunderland, Massachusetts.
- 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.
- Tateno, Y., N. Takezaki, and M. Nei. 1994. Relative efficiencies of the maximum-likelihood, neighbor-joining, and maximum-parsimony methods when substitution rate varies with site. *Molecular Biology and Evolution* **11**:261-277.
- Tavaré, S. 1986. Some probabilistic and statistical problems on the analysis of DNA sequences. *Lect. Math. Life Sci.* **17**:57-86.
- Thorne, J. L., and H. Kishino. 1992. Freeing phylogenies from artifacts of alignment. *Molecular Biology and Evolution* **9**:1148-1162.
- 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.
- Uzzell, T., and K. W. Corbin. 1971. Fitting discrete probability distribution to evolutionary events. *Science* **172**:1089-1096.
- Wakeley, J. 1994. Substitution-rate variation among sites and the estimation of transition bias. *Mol. Biol. Evol.* **11**:436-442.
- 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. 2001. Adaptive molecular evolution. Pp. 327-350 in D. Balding, M. Bishop, and C. Cannings, eds. *Handbook of statistical genetics*. Wiley, New York.
- 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*.

- 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., 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., 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

- aaDist** 25
- alignment 2
- alignment gap 11
- alpha** 19
- ambiguity characters 11
- BEB 26, 44
- BFGS 53
- BioEdit 2
- bootstrap 54
- branch label** 15, 42
- clade label** 15
- cleandata 11, 22
- clock 8, 15, 18, 22, 23, 51
- CLUSTAL 2, 60
- codon model
 - branch model 7, 42
 - branch-site model 26, 27
 - site model 7, 42
- codon position 11, 12, 22, 24, 42, 50, 58
- CodonFreq** 24
- convergence 53, 56
- evolver 30
- examples 8
- fix_alpha** 19
- fix_blength** 22
- fix_kappa** 19
- fix_rho** 19
- fossil 16, 18, 51
- GenDoc 2
- gene prediction 3
- genetic code 21
- getSE** 21
- Hadamard transform 48
- Hidden Markov model 49
- icode** 22, 27
- in.baseml 56
- in.codeml 56
- initial values 22, 54, 56
- likelihood ratio test 2, 21, 37
- LRT *See* likelihood ratio test
- lysozyme 7
- MacClade 59
- maximum likelihood estimate 37
- MCAA.dat 30
- MCbase.dat 30
- MCcodon.dat 30
- MEGA 60
- method** 22, 53
- Mgene 10, 17, 18, 21, 50
- missing data 11
- MLE *See* maximum likelihood estimate
- model** 18, 25, 28
- MOLPHY 28, 60
- Monte Carlo simulation 31
- mouse lemurs 8
- MrBayes 59
- ncatG** 19
- ndata** 18
- NEB 44
- nhomo** 20
- noisy** 18
- nonhomogeneous models 20
- nparK** 19
- NSsites 7, 25, 26
- OmegaAA.dat** 25
- optimization 37, 53
- Option G .. 10, 11, 12, 18, 19, 21, 23, 28, 50, 54, 57
- outfile** 17
- PAUP 59
- PHYLIP 59
- RateAncestor** 21, 27, 29, 45, 49
- reconstruction** 52
 - joint 52
 - marginal 52
- rho** 19
- runmode** 18, 25
- seqfile** 17
- simulation 55
- Small_Diff** 22, 57
- TipDate 8
- transition/transversion rate ratio 40
- tree
 - parenthesis notation 15
 - rooted 15
 - unrooted 15
- tree search 3, 18, 53
- TreeAlign 2
- treefile** 17
- TreeView 16, 60
- verbose** 18