Phylogenetic Analysis by Maximum Likelihood (PAML)

Version 3.13a, September 2002

Ziheng Yang

Suggested citation:

Yang, Z. 1997. PAML: a program package for phylogenetic analysis by maximum likelihood *CABIOS* **13**:555-556 (`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
Email: z.yang@ucl.ac.uk
Phone: +44 (20) 7679 4379
Fax: +44 (20) 7679 7096

Table of Contents

0 Recent changes and bug fixes	3
1 Introduction	3
2 Getting started	5
Windows 95/98NT/2000/XP and UNIX (including Mac OS X)	5
MACs	6
3 Files in the Package	6
Which files are needed?	9
4 Using Programs in the Package	9
Sequence data format	9
Tree file and representations of tree topology	12
`baseml`	14
`basemlg`	21
`codeml` (`codonml` and `aaml`)	21
Codon sequences (`seqtype` = 1)	22
Amino acid sequences (`seqtype` = 2)	25
evolver	27
`yn00`	29
`mcmctree`	30
5 Models and Methods	31
Nucleotide substitution models	31
Codon substitution models	34
Amino acid substitution models	34
Models for combined analyses of partitioned data	35
For nucleotides (baseml)	35
For codons (codeml with seqtype = 1)	36
For amino acids (codeml with seqtype = 2)	37
Global and local clocks, and dated sequences	37
Reconstruction of ancestral sequences	38
Analysing large data sets and iteration algorithms	39
Tree search algorithms	40
Simulation	41
6 Seldom used features	42
Inputting site pattern frequencies (baseml)	42
More codon models	43
Another pair of NSsites models for detecting positive selection (codeml)	44
7 Technical Notes	45
The rub file recording the progress of iteration	45
How to specify initial values	45
Fine-tuning the iteration algorithm	46
Adjustable variables in the source codes	46
PowerMAC memory allocation problem	47
MS Windows version window auto-close	47
8 Acknowledgments	47
9 References	47

0 Recent changes and bug fixes

See the file pamlHistory.txt for recent changes and bug fixes.

The paml documentation in paml v3.13 from August 2002 - 12 December 2002 has a mistake about the critical values for the newer test using a modified M8. The critical values for the test are 2.71 at the 5% significance level and 5.41 at the 1% level, rather than 1.95 and 3.32 as in the documentation. I made a mistake and thought I corrected it a few months ago, but apparently not. I am sorry for the mistake.

I checked and edited the readme.txt files in the examples folder. You are advised to check the examples and try to duplicate results we published before analyzing your data sets. Not all the examples are mentioned in this documentation.

1 Introduction

PAML is a package of programs for phylogenetic analyses of DNA or protein sequences using maximum likelihood (ML). The PAML web page (http://abacus.gene.ucl.ac.uk/software/paml.html) explains what the programs can and cannot do, how to download and compile the programs, and how to report bugs. Those will not be duplicated in this documentation. There is now also an FAQ page at http://abacus.gene.ucl.ac.uk/software/pamlFAQs.html.

The program `baseml` is for analyzing 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 a variable named `seqtype` in the control file `codeml.ctl`, that is, 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, the difference being that the unit of evolution in the substitution 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. A *discrete-gamma* model (Yang, 1994c) is used in `baseml`, `codonml` and `aaml` to accommodate rate variation among sites, by 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, 1995a) accounts for correlation of rates between adjacent sites.

The program `basemlg` implements the continuous gamma model of Yang (1993). It is slow and unfeasible for data of >6 or 7 species. The discrete-gamma model in `baseml` is recommended.

General assumptions of the models (programs) are

- 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 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`.

The sequences must be aligned. If there are alignment gaps, they will either be removed from all sequences before analysis, with appropriate adjustment to the sequence length (if `cleandata` = 1), or treated as ambiguity characters (if `cleandata` = 0).

Other small programs in the package include `evolver` for simulating sequence data sets, `pamp` for parsimony-based analysis (Yang and Kumar 1996), and `yn00` for estimating synonymous and nonsynonymous substitution rates in pairwise comparisons using the method of Yang and Nielsen (2000).

This document is now mainly an explanation of the control variables in the control files for individual programs. Topics that seem too complicated to explain there are dealt with in a section in the Chapter "Models and Methods".

2 Getting started

Windows 95/98NT/2000/XP and UNIX (including Mac OS X)

In the good old days, we type commands on a command line. Nowadays most people know dragging and double-clicking or scrolling only. PAML programs do not have a graphics or menu based interface, and so you have to know some basic techniques of the good old days. Here is the basics for getting you started.

(1) download and unpack the archive into a folder, say /paml/ (or ~/paml/ on UNIX). You should remember the name of the folder.

(2) Start a command box. On Windows, you do this by choosing "Start - Programs" and look for a command called "MS-DOS prompt" or "Command Prompt". It is usually in the group "Accessories". On UNIX, you will be in a shell window as soon as you log on by telnet. On MAC OS X, you open a terminal by Applications-Utilities-Terminal.

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

```
cd paml
cd ~/paml
cd \paml
```

(4) This step is necessary only if you are using a UNIX machine and if this is the first time. Then you need to compile the programs. You need to do this only once. You cd to the src folder and look at the readme.txt or Makefiel for compiling instructions. If you are lucky, all you have to do is to type

```
make
```

(5) Run a program. For example, to run codeml, you type the command codeml and hit the "Enter" key.

```
codeml
```

This makes codeml read the default control file codeml.ctl and do the analysis according to it. Now you can print out a copy of codeml.ctl, and open your favorite text editor to view the relevant sequence data and tree files.

Next you can prepare your own sequence data files and tree files. PAML data files are plain text files. If you use MS Word to prepare them, make sure you save them as "Text with line breaks" or "Text without line breaks" (Note that only one of those two may work). A few commonly used DOS and UNIX commands are given in the table.

DOS/Windows	UNIX	Function
cd, chdir	cd, chdir, pwd	Sets and displays current directory (folder)
copy	cp, cat	Copies files
del	rm	Deletes files
dir	ls	Lists files
exit	exit	Exits from the command processor
find	fgrep	Searches for a string in files
help	man	Gets help
md	mkdir	Makes a new directory
more	more, less	Displays file contents by screenfuls
path	set PATH	Sets search path for commands
print	lpr	Prints files
rd, rmdir	rmdir, rm -r	Removes directories
ren	mv	Renames a file
time, date	date	Displays or sets time and date
type	cat	Displays the contents of a file
xcopy	cp	Copies files and subdirectories
ftp	ftp	Starts an ftp session
telnet	telnet	Starts a telnet session
	Ctrl-Z, followed by bg	Puts a foreground job into the background
	fg	Brings a job to the foreground
	nice, renice	Be nice to others by running your jobs at a lower priority

MACs

You double-click on the executable such as codeml. The compiled executables should ask you for the name of the control file. The default name is codeml.ctl. My experience of MACs was several years old and was not a very good one.

3 Files in the Package

The following files are included in the package:

Source codes:

`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`: Markov chain Monte Carlo algorithm for Bayes estimation of phylogenies
`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
`eigen.c`: routines for calculating eigen values and vectors

Compiling commands

`Makefile`: make file
`paml.cc`: batch file for compiling PAML using the cc compiler
`paml.gcc`: batch file for compiling PAML using the GNU gcc compiler
`paml.acc`: batch file for compilation PAML using the SUN acc compiler

Control files:

`baseml.ctl`: control file for running baseml and basemlg;
`codeml.ctl`: control file for codeml (i.e., codonml and aaml)
`pamp.ctl`: control file for pamp
`yn00.ctl`: control file yn00
`mcmctree.ctl`: 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

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:

`paml.readme`: readme file
`paml.html`: paml web page, serving also as part of the manual (html file)
`pamlDOC.pdf`: this document

Example data sets:

Several example data sets are included. They were used in our papers to test new methods, and are included in the package for error-checking. Below is an incomplete list.

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

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

`abglobin.nuc`: the concatenated alpha and beta globin genes, example data for condonml

`exampleTipDate.phy` (phylip format), `exampleTipDate.trees`: data set of 17 dengo viral strains sequenced at different dates from Andrew Rambaut's TipDate program. This is used for testing the TipDate models of Rambaut (2000). Run `baseml` by specifying `clock` = 3. The results are included in the file `exampleTipDate.rst`.

`HIVenvSweden.paup` (paup* format), `HIVenvSweden.trees`, `HIVenvSweden.ctl`: 13 HIV *env* genes used by Yang *et al.* (2000) in developing models of variable selective pressures among sites (the `Nssites` models). (Use command: `codemlsites HIVenvSweden.ctl`)

`hummt25.nuc`: 25 human D-loop sequences used in Yang and Kumar (1995). Run `baseml` by specifying `fix_alpha` = 0, or run `pamp`.

`lysozymeSmall.nuc`, `lysozymeSmall.trees`, `lysozyme.ctl`: primate lysozyme genes of Messier and Stewart 1997, used by Yang (1998) in developing tests of positive selection along lineages. This is the "small data set" analyzed in that paper. See the control file `lysozyme.ctl` for details for specifying the different models. Run the analysis by `codeml lysozyme.ctl` `stewart.aa`, `stewart.trees`: lysozyme sequences of six mammals (Stewart *et al.* 1987), used by Yang *et al.* (1995) to test methods for reconstructing ancestral amino acid sequences.

`abglobin.aa`: the concatenated alpha- and beta-globins, translated from `abglobin.nuc`

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.

Note also that the programs produce result files. Some other files with names `rub`, `lnf`, `rst`, or `rates` may also be created. You should not use these names for your files.

4 Using Programs in the Package

Sequence data format

Have a look at the example data files in the package (*.nuc, *.aa, and *.paup). 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 (but see below). PAML now 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 all ignored. PAML does not deal with comment blocks in the sequence data block, so try to avoid them.

Below is an example of the PHYLIP format (Felsenstein, 1993). The first line contains the number of species and the sequence length (possibly followed by option characters). With `codonml` (`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, 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. (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.

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

Species names. Do not use special symbols like , : # ( ) in a species name as they may confuse the programs. The maximum number of characters in a species name (LSPNAME) is specifed at the beginning of the main programs `baseml.c` and `codeml.c`. The default value is 30. In PHYLIP, exactly 10 characters are used for a species name. To make this discrepancy less a problem, PAML considers two consecutive spaces as the end of a species name, so that the species name does not have to have exactly 30 (or 10) characters. To make this rule work, you should not have two consecutive spaces *within* a species name. For example the above data set can have the following format too.

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

Another thing you can do is to patch a few spaces after the species name in your PHYLIP data file, which will then be readable by both PHYLIP and PAML.

In a sequence, three special characters ".", "-", and "?" may be used: a dot means the same character as in the first sequence, a dash means an alignment gap, and a question mark means an undetermined site. Sites at which at least one sequence involves a "-" or "?" are excluded from all sequences before analysis, with the sequence length adjusted. For codon sequences, the whole codon is removed. Characters 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 used for amino acids. Other alphabetic characters cause errors. Non-alphabetic symbols such as ><!"£$%^0123456789 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.

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 classes. The following lines contain the site marks, one for each site in the sequence (or each codon in the case of `codonml`). The site mark specifies which class each site is from. If there are *g* classes, the marks should be 1, 2, ..., *g*, and if *g* > 9, the marks need to be separated by spaces. The total number of marks must be equal to the total number of sites in each sequence.

```
 5 895 G
G 4
3
123123123123123123123123123123123123123123123123123123123123
123123123123123123123123123123123123123123123123123123123123
123123123123123123123123123123123123123123123123123123123123
123123123123123123123123123123123123123123123123123123123123
123123123123123123123123123123123123123123123123123123123123
123123123123123123123123123123123123123123123123123123123123
123123123123123123123123123123123123123123123123123123123123
1231231231231231231231231231231231231
444444444444444444444444444444444444444444444444444444444444
444444444444444444444444444444444444444444444444444444444444
444444444444444444444444444444444444444444444444444444444444
444444444444444444
123123123123123123123123123123123123123123123123123123123123
123123123123123123123123123123123123123123123123123123123123
123123123123123123123123123123123123123123123123123123123123
12312312312312312312312312312312312312312312312312312312312
Human
AAGCTTCACCGGCGCAGTCATTCTCATAATCGCCCACGGACTTACATCCTCATTACTATT
CTGCCTAGCAAACTCAAACTACGAACGCACTCACAGTCGCATCATAATC........
Chimpanzee
.........
```

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

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

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

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

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

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

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

Tree file 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. Two methods for representing a tree topology are used in PAML.

Parenthesis notation: The first is the familiar parenthesis representation, that is used in virtually every 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, some programs may ask you whether you want to use those branch lengths as fixed and estimate other parameters in the substitution model only. You will then have three options: ignore the branch lengths, use them as initial values, and fix them and estimate other parameters.

Whether you should use rooted or unrooted trees depends on the model, and in particular 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 assumption, 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 of 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. 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 labels. Sometimes (such as the local clock models specified by `clock` = 2 in `baseml` and `codeml` and codon models with different ωs for branches), we need to label the branches. Branch labels are specified in the same way as branch lengths except that the symbol preceding the branch label is # rather than :. The branch labels are consecutive integers starting from 0, which is the default label 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 `lysozyme.trees`, with branch labels for fitting models of different d_N/d_S ratios (ω) for branches. The internal branch ancestral to human and gibbon has the ratio ω_1 while all other branches have the background ratio ω_0. This fits the model in table 1C for the small data set of lysozyme genes in Yang (1998).

I have found it convenient to create the tree file with labels and read the tree in using Rod page's TreeView to check that the tree is right. New versions of TreeView also allow you to add branch labels in the tree-edit window, but you have to remove the quotation marks ' in the tree manually afterwards. Also TreeView recognizes labels for internal branches (nodes) only and do not allow labels for tip branches. If you don't specify the labels in the tree file, the program will ask for input from the keyboard. For large trees, this may be very prone to mistakes. Another program that you can use to create branch or node labels is Andrew Rambaut's TreeEdit, available for the Mac, although I have not tested it.

Representation of tree topology by branches: 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 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) is specified by its 5 branches:

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

The nodes in the tree are labeled by consecutive natural numbers, with 1, 2, ..., *s* representing the *s* known sequences, in the same order as in the data. A number larger than *s* labels an interior 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. To use this format in the tree structure file, give the number of branches, and then the branches as specified by the start and end nodes.

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

It is also possible to allow some sequences in the data to be ancestors to others and such trees are easy to specify using this tree representation. For example, the tree in the following `5s.trees` file has 4 branches, with sequence 5 to be the common ancestor of sequences 1, 2, 3, and 4:

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

Warning. I have not tried to make all models implemented in baseml and codeml work with trees in which some sequences in the data are ancestral to others. You should exercise caution when using such trees.

baseml

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

```
 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

    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

        clock = 0   * 0:no clock, 1:clock; 2:local clock; 3:TipDate
        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)?
*       ndata = 1   * number of data sets
*       icode = 0   * (RateAncestor=1 for coding genes, "GC" in data)
*   readfpatt = 0   * read site pattern frequences instead of sequences
* 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.

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, and UNREST, respectively. Check Yang (1994 JME 39:105-111) for notation. Two more models are implemented recently. model = 8 are special cases of the REV model, while model = 9 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 model when model = 8 or 9. 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 here will have the rate 1. When model = 8, you specify TC or CT, but not both. When model = 9, TC and CT are different. See the following examples and Yang (1994a) for notation.

```
      model = 8   [2  (CT) (AG)]      /* TN93 */
      model = 8   [2  (TA AT TG CA CG) (AG)]         /* TN93 */
      model = 9   [1 (TC CT AG GA)]  /* K80 */
      model = 9   [0]  /* JC69 */
      model = 9   [11  (TA) (TG) (CT) (CA) (CG) (AT) (AC) (AG) (GT) (GC) (GA)]
/*UNREST*/
```

Mgene is used in combination with option G in the sequence data file, for combined analysis of data from multiple genes or 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.

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 1994c). 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 1995a). 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 1995a). The variable `nparK` takes precedence over `alpha` or `rho`.

`clock` specifies models concerning rate constancy among lineages. `clock` = 0 means no clock. `clock` = 1 means the global clock. `clock` = 2 implements local clock models (Yoder and Yang 2000), which assumes 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 marks in the tree file. If you choose noisy = 9, the program will ask for a reference (calibration) node and date and calculate dates for other nodes. `clock` = 3 implements Rambaut (2000)'s TipDate models. Evolution conforms to a global clock but sequences in the data are determined at different dates. The dates are specified at the end of the sequence names, based on Andrew's format. This model has one extra parameter (the mutation rate) than the global clock model (`clock` = 1).

`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 & 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. 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), I have added the calculation of 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 examined to infer the most likely amino acid encoded by the triplet. Posterior probabilities for stop codons are small and reset to zero to scale the posterior probabilities for amino acids. To use this option, you need add the control variable `icode` in the control file `baseml.ctl`. This is not listed in the above. The variable `icode` can take a value out of 0, 1, ..., 10, corresponding to the 11 genetic codes included in paml (See the control file `codeml.ctl` for the definition of different genetic codes). A nucleotide substitution model that is very close to a codon-substitution model can be specified as follows. You add the option characters GC at the end of the first line in the data file and choose model = 4 (HKY85) and `Mgene` = 4. The model then assumes different substitution rates, different base frequencies, and different transition/transversion rate ratio (kappa) for the three codon positions. Ancestral reconstruction from such a nucleotide substitution should be very similar to codon-based reconstruction. (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 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).

`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.

`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.

`readfpatt`: This forces the program to read site pattern frequencies instead of sequence data. See the section on "Rarely used features".

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

Output: The output should be self-explanatory. Descriptive statistics are always listed. The observed site patterns and their frequencies are listed, together with the proportions of constant patterns. Nucleotide frequencies for each species (and for each gene in case of multiple gene data) are counted and listed. lmax = ln(Lmax) is the upper limit of the log likelihood and may be compared with the likelihood for the best (or true) tree under the substitution model to test the model's goodness of fit to data (Goldman, 1993a; Yang et al. 1995). You can ignore it if you don't know what it means.

With getSE = 1, the S.E.s are calculated as the square roots of the large sample variances and listed exactly below the parameter estimates. Zeros on this line mean errors, either caused by divergence of the algorithm or zero branch lengths. The S.Es of the common parameters measure the reliability of the estimates. For example, $(\kappa - 1)/\mathrm{SE}(\kappa)$, 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 (e.g., Yang, 1994b, Goldman and Yang, 1994).

If the tree file has more than one tree, the programs baseml and codeml will calculate the bootstrap proportions using the RELL method (Kishino and Hasegawa 1989), as well as the method of Shimodaira and Hasegawa (1999) with a correction for multiple comparison. The bootstrap resampling accounts for possible data partitions (option G in the sequence data file). I did not bother to deal with ties, so if you include the same tree in the tree file more than once, you need to adjust the proportions for those trees yourself. The program rell, included in earlier versions, is now removed.

basemlg

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

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

codeml (codonml and aaml)

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

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

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

      seqtype = 2   * 1:codons; 2:AAs; 3:codons-->AAs
    CodonFreq = 2   * 0:1/61 each, 1:F1X4, 2:F3X4, 3:codon table
       aaDist = 0   * 0:equal, +:geometric; -:linear, 1-6:G1974,Miyata,c,p,v,a
   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, 7:AAClasses, 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; 2-10: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

        clock = 0   * 0:no clock, 1:clock; 2:local clock; 3:TipDate
        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)?
*       ndata = 10
* 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 (under 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.

runmode = -2 performs ML estimation of d_S and d_N in pairwise comparisons. 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 change. Otherwise you need to edit the files to cut the names short.

The variable **model** concerns assumptions about the d_N/d_S rate ratios (or the ω parameters) among branches (Yang 1998; Yang and Nielsen 1998). `model` = 0 means one d_N/d_S 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 may specify the ω ratios for branches using branch labels (see the section on tree file). I strongly advise you to use this method of specifying branch labels. Otherwise, the program will ask the user to input a branch mark for the d_N/d_S ratio assumed for each branch. This should be an integral number between 0 to k - 1 if k different d_N/d_S ratios (ω_0 - ω_{k-1}) are assumed for the branches of the tree. This process is prone to user mistakes if the tree is not very small. I run the program first to let it output the tree topology using the branch representation on the screen. I then prepare the branch labels in a file, say, `in`. Finally I run the program using redirection so that it will read the input from the prepared file

```
        codeml < in
```

Redirection is not permitted on a MAC, but you can prepare the branch labels in a file and then copy and paste them into the session running the program. Furthermore, under this model, the variable `fix_omega` fixes the last d_N/d_S ratio (ω_{k-1}) 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. It should, however, be noted that it is not proper to use the option model = 1 to estimate dN/dS ratios for all branches to find out which ratios are greater than one, and then to use model = 2 to test whether that difference is significant. This way the hypothesis is derived from the data and is tested using the same data. As a result, you tend to get significant results too often. The `lysozyme` example files (both the "large" and the "small" data sets) are included in the examples/ folder in the package for you to reproduce results published in 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.* 2000). `Nssites` = m corresponds to model Mm in Yang *et al.* (2000). 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 7 8
```

This forces the program to run M0, M1, M7, and M8 on the same data set. In a few previous versions of paml (up to version 3.12), I compiled a separate executable file `codemlsites` for the batch run, and that is now replaced by the above interface. Note that 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.* (2000), and you do not have any control over it.

As noted in Yang *et al.* (2000), some of the models are hard to use, in particular, M12 and M13. We recommended models 0 (one-ratio), 1 (neutral), 2 (selection), 3 (discrete), 7 (beta), and 8 (beta&ω). Some of the models like M2 and M8 are noted to be prone to the problem of multiple local optima. You are advised to run the program at least twice, once with a starting omega value <1 and a second time with a value > 1, and use the results corresponding to the highest likelihood.

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

Example files for NSsites models: The HIV *env* data set used in Yang *et al.* (2000: table 12) is included in the paml/examples/hivNSsites folder. The abalone sperm lysin data set was analyzed by Yang, Swanson and Vacquier (2000) 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 models of Yang and Nielsen (2002) are specified by

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

See that paper for details. Note that under both models A and B, the number of site classes is fixed (at four; see Yang & Nielsen 2002), and the variable `ncatG` is ignored by the program. (To run the discrete model with only 2 site classes, which is the null model to be compared with model B, you should specify `model` = 0, `NSsites` = 3, `ncatG` = 2. The discrete model with 3 site classes (`ncatG` = 3) is not nested within model B, and you cannot use it for comparison with model B using the χ^2 distribution.) The output should be self-explanatory. The (“large”) lysozyme data set analyzed in that paper is included in the examples folder in the package. Look at the readme file.

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: 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.

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. Note that the estimates of these quantities for the a- and b-globin genes shown in Table 2 of Goldman and Yang (1994), calculated using the MEGA package (Kumar *et al.*, 1993), are not accurate.

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 (`jones.dat`) (see Kishino *et al.*, 1990 for the construction), and Whelan and Goldman (wag.dat). The file `mtmam.dat` has a matrix for mitochondrial proteins estimated by maximum likelihood from a data set of 20 mammals. The mtREV24 model of the MOLPHY package 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.

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

If you do pairwise ML comparison (`runmode` = -2) and the data contain ambiguity characters or alignment gaps, the program will remove all sites which have such characters from all sequences before the pairwise comparison if `cleandata` = 1. This is known as "complete deletion". It will remove alignment gaps and ambiguity characters in each pairwise comparsion ("pairwise" deletion) if `cleandata` = 0. (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

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. 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, with some notes. Note that the first block of the file has the inputs for evolver, while the rest is 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. In the example, the sum of branch lengths is 1.12, and so using either 1.12 or –1 for the tree length has the same effect. 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
234567
4 200 2
-1
((1:.1, 2:.2):.12, 3:.3, 4:.4);
6
```

```
1 2 3 4 5
.5 4

0.25 0.25 0.25 0.25
 T    C    A    G (fixed order)

==================================================
The rest of this data file are notes, ignored by the program evolver.
evolver simulates nucleotide sequences under the REV+Gamma model
and its simpler forms.
The variables in this file are defined below:

==================================================
<format,0=paml,1=paup>
<random number seed>
<# seqs>  <# nucleotide sites>  <# replicates>
<tree length, use -1 if tree has absolute branch lengths>
<tree with relative branch lengthes>
<model: 0:JC69, 1:K80, 2:F81, 3:F84, 4:HKY85, 5:TN93, 6:REV>
<kappa or rate parameters in model>
<alpha>  <#categories for discrete gamma>
<base frequencies>
==================================================
```

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

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

This evolver program evolved from the old boring program `listtree` and still has the 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.

Simulation algorithm used in evolver. 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. Check any book on Monte Carlo simulation for procedures of sampling from a multinomial distribution, and see, e.g., Yang (1996c; 1997) for more details of simulations on phylogenies. Sequences at the ancestral nodes are generated during the simulation but not included in the output. If you want those ancestral sequences, you can search for the following line in the routine Simulate() in the file evolver.c, and change the value from 0 to 1.

    int verbose=0;

Recompile the evolver program. The program will then output the ancestral sequences in a file named `ancestral.seq`.

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). It should be possible to use those files directly with distance programs such as `NEIGHBOR` in Felesenstein's PHYLIP package.

mcmctree

The program mcmctree performs Bayesian estimation of phylogenies (Rannala and Yang, 1996; Yang and Rannala, 1997). The birth-death process with species sampling is used to specify the prior distribution of phylogenies ("labeled histories", which are rooted tree topologies with the interior nodes ordered according their associated speciaiton times), and the posterior probabilities of the labeled histories are compared to select the maximum posterior probability tree. The program implementing the method of Rannala and Yang (1996) is not distributed as the algorithm involves extensive computation. Instead, mcmctree implements a refined method (Yang and Rannala 1997), which uses Markov chain Monte Carlo to select candidate labeled histories, and Monte Carlo integration to integrate over the distribution of the ancestral speciation times.

The default control file name is mcmctree.ctl, and a sample copy is shown below.

```
      seqfile = mtprim9.nuc * sequence data file name
      outfile = mcmctree.out * main result file name
     treefile = 9s.trees * tree structure file name
       LHfile = Lhs * LH file name. read (MCMC=0) or overwritten (MCMC=1)
         MCMC = 0 *0: read LHs from LHfile, 1: use MCMC to generate Lhs
         beta = 0.15 * prob{change labeled history}, used only if MCMC=1
       delta0 = 1 * small number for MCMC, used only if MCMC=1
       delta1 = .2 * smaller number for comparing candidate LHs

        model = 2 * 0:JC69, 1:K80, 2:F81, 3:F84, 4:HKY85

        kappa = 2. * given kappa and omega in GY94.
        alpha = 0 * given alpha, 0:infinity
        ncatG = 8 * # of categories in the dG or AdG models of rates

     hierarch = 0 * 1:hierarchical; 0:empirical Bayes analysis
        birth = 6.7 * lineage birth rate
        death = 2.5 * lineage death rate
       sample = .06 * sampling proportion
       mutate = .24 * mutation rate (# of mutations from root to present)
```

The variables seqfile, outfile, treefile, model, kappa, alpha, and ncatG are defined as in baseml.ctl for program baseml. The program first collects a set of candidate labeled histories. If MCMC = 0, the candidate labelled histories are read from the file LHfile, while if MCMC = 1, they are generated from the Markov chain Monte Carlo, with the Monte Carlo integration over ancestral speciation times evaluated at a low accuracy level (controled by delta0). The program then calculates the (relative) posterior probabilities of these candidate labelled histories, with the Monte Carlo integration evaluated at a higher accuracy level (controlled by delta1). The Markov chain moves with probability beta to another labelled history of the same tree topology, or to a labelled history of a different tree topology through a nearest neighbor interchange. hierarch controls whether an empirical Bayes analysis or a hierarchical Bayes analysis is to be performed. birth, death, and sample are parameters of the prior distribution specified by the birth-death process with species sampling.

5 Models and Methods

This section provides some background information about the analysis that the programs in the `paml` package perform.

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.*, 1985), Tamura (1992), Tamura and Nei (1993), and REV (Yang, 1994a). The rate matrices of these models are given below

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

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

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

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

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

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

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

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

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

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

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 π_T, π_C, π_A, π_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.* (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 (1994b, 1994c) and Tateno *et al.* (1994). Thorne *et al.* (1992) described the transition probability matrix, and Yang (1994c) 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; Yang, 1994a; see also Tavare, 1986; Zharkikh, 1994). It is used in `baseml` only. It seems sufficiently general to enable accurate estimation of the substitution pattern from real data. See Gillespie (1986), Tavare (1986), Rodriguez *et al.* (1990), Yang (1994a), and Zharkikh (1994) for reviews of substitution models.

Unfortunately there are 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). 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 (2000) notation, $R = \alpha/(2\beta)$. PHYLIP and PAUP* use R and name it the "transition/transversion rate ratio", while referred to κ as the "transition/transversion rate parameter". For a general substitution model $Q = \{q_{ij}\}$, κ and R are related by the formula

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

Special examples are listed in the following table.

Model	Average transition/transversion rate ratio (R)
JC69	½
K80	$\kappa/2$
F81	$(\pi_T\pi_C + \pi_A\pi_G)/(\pi_Y\pi_R)$
F84	$[\pi_T\pi_C(1 + \kappa/\pi_Y) + \pi_A\pi_G(1 + \kappa/\pi_R)] / (\pi_Y\pi_R)$
HKY85	$(\pi_T\pi_C + \pi_A\pi_G)\kappa/(\pi_Y\pi_R)$
T92	To be filled in.
TN93	$(\pi_T\pi_C\kappa_1 + \pi_A\pi_G\kappa_2)/(\pi_Y\pi_R)$
REV (GTR)	$(\pi_T\pi_C a + \pi_A\pi_G)/(\pi_T\pi_A b + \pi_T\pi_G c + \pi_C\pi_A d + \pi_C\pi_G e)$

The case of no transition-transversion bias is represented by $\kappa = 1$ and R = ½ 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.

Codon substitution models

The model of Goldman and Yang (1994) specifies the probability of substitution between the sense codons, by using the matrix of amino acid distances of Grantham (1974). The model does not seem to fit real data well, however, and the user is advised to use the following simpler version, which is equivalent to use equal distances for any pair of amino acids. The substitution rate from codon *i* to codon *j* is given 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 model forms the basis for more sophisticated models implemented in codeml, such as those that allow the ω ratio to vary among branches in the phylogeny (Yang 1998; Yang and Nielsen 1998) implemented through the variable `model`, and those that allow the ω ratio to vary among sites (among codons or amino acids in the protein), implemented through the variable `Nssites`.

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 (in press) 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, b) 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.

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 equavilent 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 asumes different substitution rates but the same pattern of nucleotide substitution for different genes. `Mgene` = 2 means different frequency parameters for different genes but the same rate ratio parameters (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	`Mgene` = 0	everything equal
Option G	`Mgene` = 0	the same κ and π, but different *c*s (proportional branch lengths)
Option G	`Mgene` = 2	the same κ, but different πs and *c*s
Option G	`Mgene` = 3	the same π, but different κs and *c*s
Option G	`Mgene` = 4	different κ, πs, and *c*s
Option G	`Mgene` = 1	different κ, πs, and different (unproportional) branch lengths

The different *c*s 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 (`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, 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). Note that the data file used in that paper is slightly different from the lysin data analyzed in Yang, Swanson & Vacquier (2000), with one codon deleted.

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 *c*s (proportional branch lengths)
Option G	`Mgene` = 2	the same (κ,ω), but different πs and *c*s
Option G	`Mgene` = 3	the same π, but different (κ, ω) and *c*s
Option G	`Mgene` = 4	different (κ, ω), πs, and *c*s
Option G	`Mgene` = 1	separate analysis

For amino acids (codeml with seqtype = 2)

Se Yang (996 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 *c*s (proportional branch lengths)
Option G	`Mgene` = 2	different πs and *c*s
Option G	`Mgene` = 1	separate analysis

Global and local clocks, and dated sequences

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

The option `clock` = 3 implements Andrew Rambaut's TipDate models (Rambaut 2000; see also the `TipDate` program web page at <`http://evolve.zps.ox.ac.uk/software/TipDate/main.html`>). For viral sequences determined in different years, a global molecular clock can be fitted to the data with the dates of sequence determination used in the likelihood calculation. I have used Andrew's format, which specifies the dates at the end of the sequence names; see `exampleTipDate.phy`, which is the example file in Rambaut's package. This model has one extra parameter (the mutation rate) than the global clock model (`clock` = 1). Thanks to Andrew for help with the implementation.

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.* (1995) 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.* 1995). 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.* 1995). 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 (except for the simplest possible case; see Yang 2000). 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 submitted). 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 (1996a). The algorithm starts from either the star tree (`runmode` = 2) or a multifurcating tree read from the tree structure file (`runmode` = 1). The algorithm joins two taxa to achieve the greatest increase in log-likelihood over the star-like tree. This will reduce the number of OTUs by one. The process is repeated to reduce the number of OTUs by one at each stage, until no multifurcation exists in the tree. This algorithm works either with or without the clock assumption. The stepwise addition algorithm is implemented with the option `runmode` = 3. Options `runmode` = 4 or 5 are used for nearest neighbor interchanges , with the intial tree determined with stepwise addition under the parsimony criterion (`runmode` = 4) or read from the tree structure file (`runmode` = 5). The results are self-explanatory.

Besides the fact that ML calculations are slow, my implementations of these algorithms are crude. If the data set is small (say, with <20 or 30 species), the stepwise addition algorithm (`runmode` = 3) appears usable. Choose `clock` = 0, and `method` = 1 to use the algorithm that updates one branch at a time, and fix substitution parameters in the model (such as 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.

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, for example, Huelsenbeck (1995) and Yang (1996c), for more details.

Tips:

1. For analyzing multiple simulated data sets, it is advisable that you copy the tree topology from the Mcbase.dat or Mcaa.dat file into the tree file for

baseml or codeml. Then when you run baseml or codeml, the program will ask you what to do about the branch lengths in the tree topology and you choose "using them as initial values". This should speed up the iteration since the true parameter values should be good initial values.

2. A good test of the simulation as well as the analysis program is to use a small tree to simulate a large data set of very long sequences (say 1 million nucleotides or amino acids) and then use baseml or codeml to analyse the data to see whether you get estimates very close to the true values. As ML is consistent, it should return the correct values with infinitely long sequences.

3. 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.

6 Seldom used features

A main purpose of this section is to record what I have done. This section contains notes about features in the program that are not well-known or used. They also tend to be less well-maintained, so you should take extra care and try to do some tests before using them. If you find something strange, let me know and I'll try to fix the problem.

Inputting site pattern frequencies (baseml)

If you add a variable `readfpatt = 1` in the control file for `baseml`, the program will read the site pattern frequencies, instead of sequence data, from the `seqfile` you specify. The format of that data file will then be like this.

```
4   50
TTTT     25
CTTC     30
… <47 lines deleted>
GCTA     40
```

This file is for 4 sequences with 50 site patterns. The pattern TTTT is observed at 25 sites, and the pattern CTTC is observed 30 times, and so on.

There are a number of restrictions to this option. For example, the multiple-gene options are not available. You can't have ambiguity characters in the site patterns.

The site pattern frequencies can be decimal numbers or expected frequencies under another model. One use of this feature is to test the consistency of tree reconstruction methods (Yang 1995b). For example, you can calculate the expected site pattern frequencies under a complex model and then use this feature to read in those frequencies as input data and evaluate different tree topologies using a simpler model to see whether the correct tree is recovered. An alternative is to use evolver to simulate a data set with 10M sites under the complex model and then use baseml to analyze the large data set under the simple model.

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 $i_1 i_2 i_3$, and codon *j* is the triplet $j_1 j_2 j_3$. 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* = TTT to *j* = TCT is a nonsynonymous transversion, and the rate is $q_{\text{TTT}\to\text{TCT}} = \kappa\omega\pi_C^{(2)}$, where $\pi_C^{(2)}$ is the frequency of C at codon position 2. This model is obviously time-reversible, with the equilibrium codon frequencies 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 $q_{\text{TTT}\to\text{TCT}}$ = $\frac{\kappa\omega}{\pi_T^{(1)}\pi_C^{(2)}}\times\pi_T^{(1)}\pi_C^{(2)}\pi_T^{(3)}$ and $q_{\text{TCT}\to\text{TTT}}$ = $\frac{\kappa\omega}{\pi_T^{(1)}\pi_C^{(2)}}\times\pi_T^{(1)}\pi_T^{(2)}\pi_T^{(3)}$, which is the detailed balance condition for reversibility.

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

Another pair of NSsites models for detecting positive selection (codeml)

Rasmus Nielesn, Tim Massingham, and Nick Goldman are of the opinion that the comparison between M7 (beta) and M8 (beta&ω), which has $\omega > 0$, is not a very proper test of positive selection, and should be replaced by the following comparison:

Null model: M8 (beta&ω) with $\omega = 1$ fixed;
Alternative model: M8 (beta&ω) with the constraint $\omega \geq 1$.

Twice the log-likelihood difference between the two models should then be compared with a 50:50 mixture of a chi-square distribution with one degree of freedom and a point-mass at zero; that is, the critical value is 2.71 at the 5% significance level and 5.41 at the 1% level. Or better still, you can calculate the P value using the χ^2 distribution with df = 1 (just type "`chi2 1`") and then divide it by 2. So a statistic of $2\Delta\ell = 2.00$ for this test has the P value 0.157/2 = 0.079.

This formulation has the advantage that the asymptotic distribution of the test statistic is well-known and that it is more robust to violation of assumptions in the null model.

The null model M8 (beta&$\omega = 1$) is easy to implement. You choose `NSsites` = 8, fix_omega = 1, and `omega` = 1 in the control file. The alternative model M8 (beta&$\omega \geq 1$) is more complicated and there are two ways of implementing it. The first is to search for the routine SetxBound() in the source file codeml.c, and change the bounds for omega from `omegab[]={.0001,999}` to `omegab[]={1,999}`. You then recompile codeml and run model M8 with a starting `omega` that is >1; for example, you can have `NSsites` = 8, `fix_omega` = 0, and `omega` = 3.14 in the control file. A second way of implementing the alternative model M8 (beta&$\omega \geq 1$) is to run M8 a few times to locate all the peaks in the range (0, ω), and then determine the highest value in the interval (1, ω). So if there is a local peak in the interval (1, ω), you choose the greater of the log likelihood at the peak and at $\omega = 1$ (the value under the null model). If there is no local peak in the interval (1, ω), you use the log likelihood at $\omega = 1$, with the likelihood ratio statistic to be 0.

You can also run those two models as follows, if you have only one tree in the tree file and run one model at one time.

```
NSsites = 14   * 14:beta&w=1; 15:beta&w>=1
```

7 Technical Notes

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

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.

How to specify 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`. When this file exists, 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 base on those estimates such as reconstructing ancestral sequences, insert -1 before the initial values.

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 into the file of initial values, and if you like, change one or a few of the parameter values too.

Fine-tuning the iteration algorithm

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

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

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

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

Adjustable variables in the source codes

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

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

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

```
double tb[]={.0001,9}, rgeneb[]={0.1,99}, rateb[]={1e-4,999};
double alphab[]={0.005,99}, rhob[]={0.01,0.99}, omegab[]={.001,99};
```

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

PowerMAC memory allocation problem

When your data set is large, you may see a message like "oom ", which stands for "out of memory". If you think your data set should be manageable by the program/computer, you can change the memory that is allowed by the operating system for the program to use. If you select the file name and choose "File-Get information", you should see a pop up window. You can increase numbers in this window.

MS Windows version window auto-close

Run the Windows version from a DOS/Windows command box by typing the program names such as `baseml`. Do not run the programs by double clicking on the file names from Windows 95/98/2000/NT Explorer. Otherwise, the window will close automatically when the programs finish or abort and you won't have the chance to see any error messages.

8 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 Tianlin Wang for the eigen routine used in the package. I also 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.

9 References

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

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

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. In *Atlas of Protein Sequence and Structure*, Vol 5, Suppl. 3 (ed M. O. Dayhoff), National Biomedical Research Foundation, Washington D.C., pp. 345-352.

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

Felsenstein, J. 1985. Confidence limits on phylogenies: an approach using the bootstrap. *Evolution* **39**:783-791.

Felsenstein, J. 1993. *Phylogenetic Inference Package (PHYLIP), Version 3.5*. University of Washington, Seattle.

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

Goldman, N. 1993b. Simple diagnostic statistical tests of models for DNA substitution. *Journal of Molecular Evolution* **37**:650-661.

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.

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 estimation of the hominoid tree from mitochondrial DNA sequences. *Evolution* **43**:672-677.

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. Molecular phylogeny and evolution of primate mitochondrial DNA. *Molecular Biology and Evolution* **5**:626-644.

Huelsenbeck, J. P. 1995. The performance of phylogenetic methods in simulation. *Systematic Biology* 44:17-48.

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

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

Kishino, H., and M. Hasegawa. 1989. Evaluation of 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.

Kumar, S., K. Tamura, and M. Nei. 1993. *MEGA: Molecular Evolutionary Genetics Analysis.* The Pennsylvania State University, University Park, PA 16802.

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

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

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

Pupko, T., I. Pe, et al. 2000. A fast algorithm for joint reconstruction of ancestral amino acid sequences. *Molecular Biology and Evolution* **17**: 890-896.

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

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

Rodriguez, F., J. F. Oliver, A. Marin, and J. R. Medina. 1990. The general stochastic model of nucleotide substitutions. *Journal of Theoretical Biology* **142**:485-501.

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

Swanson, W. J., Z. Yang, M. F. Wolfner and C. F. Aquadro. 2001. Positive Darwinian selection in the evolution of mammalian female reproductive proteins. *Proceedings of the National Academy of Sciences of U.S.A.* **98:**2509-2514.

Swofford, D. L. 1993. *Phylogenetic Analysis Using Parsimony (PAUP), Version 3.2*. University of Illinois, Champaign.

Swofford, D. L., G. J. Olsen, P. J. Waddel, and D. M. Hillis. 1996. Phylogeny Inference. Pp. 411-501 in D. M. Hillis, C. Moritz, and B. K. Mable eds. Molecular Systematics, 2nd ed. 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.

Tavare, S. 1986. Some probabilistic and statistical problems on the analysis of DNA sequences. In *Lectures in Mathematics in the Life Sciences*, Vol. 17, pp. 57-86.

Thorne, J. L., H. Kishino, and J. Felsenstein. 1991. An evolutionary model for maximum likelihood alignment of DNA sequences. *Journal of Molecular Evolution* **33**:114-124. (Erratum: *Journal of Molecular Evolution* **34:**91 [1992].)

Thorne, J. L., H. Kishino, and J. Felsenstein. 1992. Inching toward reliability: An improved likelihood model of sequence evolution. *Journal of Molecular Evolution* **34**:3-16.

Wakeley, J. 1993. Substitution rate variation among sites in hypervariable region 1 of human mitochondrial DNA. *Journal of Molecular Evolution* **37**:613-623.

Wakeley, J. 1996. The excess of transitions among nucleotide substitutions: new methods of estimating transition bias underscore its significance. *Trends in Ecology and Evolution***11**:158-163.

Whelan, S. and N. Goldman, in press. A general empirical model of protein evolution derived from multiple protein families using a maximum likelihood approach. *Molecular Biology and Evolution*

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. Estimating the pattern of nucleotide substitution. *Journal of Molecular Evolution* **39**:105-111.

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

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

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

Yang, Z. 1995b. Evaluation of several methods for estimating phylogenetic trees when substitution rates differ over nucleotide sites. *Journal of Molecular Evolution* **40:**689-697.

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

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

Yang, Z. 1996*c*. Phylogenetic analysis using parsimony and likelihood methods. *Journal of Molecular Evolution* **42**:294-307.

Yang, Z. 1997. How often do wrong models produce better phylogenies? *Molecular Biology and Evolution* **14:**105-108.

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. 2000. Complexity of the simplest phylogenetic estimation problem. *Proceedings of the Royal Society B: Biological Sciences* **267**:109-116.

Yang, Z. 2000. 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, Chapter 12 (pp. 327-350) in *Handbook of statistical genetics*, eds. D. Balding, M. Bishop, and C. Cannings. Wiley, New York.

Yang, Z., and B. Bielawski. 2000. Statistical methods for detecting molecular adaptation. *TREE* **15**:496-503.

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

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

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

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

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 W. J. Swanson. 2002. Codon-substitution models to detect adaptive evolution that account for heterogeneous selective pressures among site classes. Mol. Biol. Evol. **19:**49-57.

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

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

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

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

Yang, Z., N. 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., N. Nielsen, N. Goldman, and A.-M. Pedersen. 2000. 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.

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