

# ***BALi-Phy* User's Guide v3.0**

**Benjamin Redelings**

---

## **Table of Contents**

### [1. Introduction](#)

### [2. Installation](#)

#### [2.1. Hardware requirements](#)

#### [2.2. Install on MS Windows](#)

#### [2.3. Install on Mac OS X](#)

#### [2.4. Install on Linux](#)

#### [2.5. Add BALi-Phy to your PATH](#)

#### [2.6. Test the installed software](#)

#### [2.7. Install programs used for viewing the results](#)

### [3. Compiling \*BALi-Phy\*](#)

#### [3.1. Setup](#)

#### [3.2. Clone, Configure, Compile](#)

#### [3.3. Options: compiler and linker flags](#)

### [4. Running the program](#)

#### [4.1. Quick Start](#)

#### [4.2. Command line options](#)

#### [4.3. Multiple genes/partitions, models, and linking](#)

#### [4.4. Running on computing clusters](#)

#### [4.5. Option files \(Scripts\)](#)

#### [4.6. Examples](#)

### [5. Input](#)

#### [5.1. Sequence formats](#)

#### [5.2. Is my data set too large?](#)

### [6. Output](#)

#### [6.1. Output directory](#)

#### [6.2. Output files](#)

#### [6.3. Summarizing the output](#)

#### [6.4. Summarizing the output - scripted](#)

### [7. Models and Priors](#)

#### [7.1. Models and distributions are functions](#)

#### [7.2. Priors](#)

#### [7.3. Models and '+' notation](#)

#### [7.4. Default values and default priors](#)

#### [7.5. Argument and result types](#)

### [8. Substitution models](#)

#### [8.1. Default substitution models](#)

#### [8.2. Basic CTMC models](#)

#### [8.3. Substitution Frequency models](#)

#### [8.4. Substitution Mixture Models](#)

#### [8.5. The branch-site substitution model](#)

#### [8.6. Substitution model examples](#)

#### [8.7. Genetic Codes](#)

## [9. Insertion/deletion models](#)

## [10. Convergence and Mixing: Is it done yet?](#)

### [10.1. Definition of Convergence](#)

### [10.2. Definition of Mixing](#)

### [10.3. Diagnostics: Variation in split frequencies across runs \(ASDSF/MSDSF\)](#)

### [10.4. Diagnostics: Potential Scale Reduction Factors \(PSRF\)](#)

### [10.5. Diagnostics: Effective sample sizes \(ESS\)](#)

### [10.6. Diagnostics: Stabilization](#)

## [11. Alignment utilities](#)

### [11.1. alignment-info](#)

### [11.2. alignment-cat](#)

### [11.3. alignment-thin](#)

### [11.4. alignment-draw](#)

### [11.5. alignment-find](#)

### [11.6. alignment-indices](#)

### [11.7. alignment-chop-internal](#)

## [12. Tree utilities](#)

### [12.1. trees-consensus](#)

### [12.2. trees-bootstrap](#)

### [12.3. trees-to-SRQ](#)

## [13. Frequently Asked Questions \(FAQ\)](#)

### [13.1. Input files](#)

### [13.2. Running \*\*bali-phy\*\*.](#)

### [13.3. Run-time error messages](#)

### [13.4. Stopping \*\*bali-phy\*\*.](#)

### [13.5. Running \*\*bp-analyze\*\*.](#)

### [13.6. Interpreting the results.](#)

### [13.7. How do I...](#)

# 1. Introduction

*BAl i-Phy* is a Unix command line program that is developed primarily on Linux. *BAl i-Phy* also runs on Windows and Mac OS X, but it is not a GUI program and so you must run it in a terminal. Therefore, you might want to keep a [Unix tutorial](#) or [Unix cheat sheet](#) handy while you work.

In addition to the main **bali-phy** executable, *BAl i-Phy* comes with a collection of small command-line utilities such as **alignment-cat**, **trees-consensus**, etc. These utilities can be used to process alignments, assemble data sets, and summarize the results of MCMC.

# 2. Installation

## 2.1. Hardware requirements

We typically run *BAl i-Phy* on workstations with at least 8Gb of RAM and 2 cores. More cores will allow you to run more MCMC chains at once, and more RAM will allow you to run larger data sets. However, it is often easier and faster to run *BAl i-Phy* on a (Linux) computing cluster, if you have one available.

## 2.2. Install on MS Windows

First check that you have a 64-bit version of the Windows operation system installed. The executables for download will only run on a 64-bit installation of Windows.

### 2.2.1. Install a Unix command line: Cygwin (recommended)

Before you can use *BAlI-Phy* on Windows, you need to install a Unix command-line environment. We recommend installing [Cygwin](#). You may then access the Unix command line environment by running the Cygwin shell (not the normal windows command line). The Cygwin shell mounts the C: drive on `/cygdrive/c/`, so you can access the directory `C:/Users/` as `/cygdrive/c/Users/` from within the Cygwin shell, for example.

While running the Cygwin installer [setup-x86\\_64.exe](#), you will be given an opportunity to select additional packages.

- From *Science*, select *R*.
- From *Math*, select *gnuplot*.
- From *Interpreters*, select *perl*.
- From *Web*, select *wget*.
- From *Editors*, select *nano*.

You can re-run the installer to add packages that you did not add during the initial install.

### Note

BAlI-Phy refers to windows files using the normal `C:/` method because it is compiled as a native windows executable. The combination of native windows executables (which want `C:/`) and the Cygwin shell (which wants `/cygdrive/c/`) can be confusing. If you supply Cygwin filenames with `/cygdrive/` to native windows executables like BAlI-Phy, then it may complain that the files cannot be found.

## 2.2.2. Install a Unix command line: Msys2 (alternative)

You can optionally use [MSYS2](#) instead of Cygwin. Both MSYS2 and Cygwin can be installed at the same time. After installing MSYS2, You may access the Unix command line environment by running the MSYS2 shell (not the normal windows command line). The MSYS2 shell mounts the C: drive on `/c/`, so you can access the directory `C:/Users/` as `/c/Users/` from within the MSYS2 shell, for example.

After installing MSYS2 you will need to install a few packages before you proceed. Run the MSYS2 shell, and enter the command:

```
% pacman -S perl tar
```

## 2.2.3. Install BAlI-Phy executables from website

First, download and extract the executables:

```
% mkdir -p ~/Applications
% cd ~/Applications
% wget http://www.bali-phy.org/files/bali-phy-3.0-win64.tar.gz
% tar -zxf bali-phy-3.0-win64.tar.gz
```

Second, check that the **bali-phy** executable runs:

```
% ~/Applications/bali-phy-3.0/bin/bali-phy --version
```

You still need to add it to your PATH as described in [Section 2.5, "Add BAlI-Phy to your PATH"](#).

## 2.3. Install on Mac OS X

### 2.3.1. Install BAlI-Phy using homebrew (recommended)

First install the [XCode](#) (version 6 or higher) command line tools:

```
% xcode-select --install
```

Then install [homebrew](#) and use homebrew to compile and install **bali-phy**:

```
% brew tap breidelings/bioinformatics
% brew install bali-phy
```

Check that the executable runs:

```
% bali-phy --version
```

If you install with homebrew, you don't need to do anything extra to put bali-phy in your PATH.

### 2.3.2. Install BALi-Phy using executables from website (alternative)

First download and extract the executables:

```
% mkdir -p ~/Applications
% cd ~/Applications
% curl -O http://www.bali-phy.org/files/bali-phy-3.0-mac64.tar.gz
% tar -zxvf bali-phy-3.0-mac64.tar.gz
```

Check that the executable runs:

```
% ~/Applications/bali-phy-3.0/bin/bali-phy --version
```

You still need to add it to your PATH as described in [Section 2.5. "Add BALi-Phy to your PATH"](#).

### 2.3.3. Install programs used by bp-analyze using homebrew

You can install *gnuplot* via homebrew:

```
% brew install gnuplot
```

You can install *R* via homebrew:

```
% brew tap caskroom/cask
% brew cask install xquartz
% brew install r
```

However, note that this might conflict with R installed from other places, such as [MRAN](#).

### 2.3.4. Install some of the programs used for viewing the results using homebrew

You can install Figtree with homebrew:

```
% brew tap caskroom/cask
% brew cask install figtree
```

However, Seaview and Tracer don't have homebrew packages at the moment.

## 2.4. Install on Linux

### 2.4.1. Install BALi-Phy using apt-get (recommended if using *Debian: testing or unstable*)

```
% sudo apt-get install bali-phy
```

Check that the executable runs:

```
% bali-phy --version
```

If you install with **apt-get**, you don't need to do anything extra to put bali-phy in your PATH.

### 2.4.2. Install BALi-Phy using executables from website (alternative)

First install **wget**. If you have Debian or Ubuntu Linux, type:

```
% sudo apt-get install wget
```

Then download and extract the executables:

```
% mkdir -p ~/Applications
% cd ~/Applications
% wget http://www.bali-phy.org/files/bali-phy-3.0-linux64.tar.gz
% tar -zxf bali-phy-3.0-linux64.tar.gz
```

Second, check that the executable runs:

```
% ~/Applications/bali-phy-3.0/bin/bali-phy --version
```

You still need to add it to your PATH as described in [Section 2.5, “Add BALi-Phy to your PATH”](#).

### 2.4.3. Install programs used by bp-analyze

If you have Debian or Ubuntu Linux, you can install other recommended programs by typing:

```
% sudo apt-get install gnuplot
% sudo apt-get install r-base
```

### 2.4.4. Install programs used to view the results

```
% sudo apt-get install seaview
% sudo apt-get install figtree
```

However, there isn't a Debian or Ubuntu package for Tracer at the moment.

## 2.5. Add BALi-Phy to your PATH

### 2.5.1. Is bali-phy in your PATH already?

First check if the executable is in your PATH.

```
% bali-phy --version
```

If this shows version info, then **bal**i-**phy** is already in your PATH and you can skip this section. This should be true if you installed **bal**i-**phy** using a package manager such as homebrew or apt, or if you've already added it to your PATH.

If bali-phy is not in your path, then you should see:

```
% bali-phy --version
bali-phy: command not found.
```

If bali-phy is not in your PATH, then continue with this section.

### 2.5.2. Quick version

Add **bal**i-**phy** to your PATH, so that the shell knows where to find it. This command only affects the terminal in which it is typed, and will not affect new terminals:

```
% export PATH=~/Applications/bali-phy-3.0/bin:$PATH
```

To set the PATH automatically for new terminals, type:

```
% test -r ~/.bash_profile && echo 'export PATH=~/.Applications/bali-phy-3.0/bin:$PATH' >> ~/.bash_profile
% echo 'export PATH=~/.Applications/bali-phy-3.0/bin:$PATH' >> ~/.profile
```

This will affect new terminals only after you log out and log back in though.

Now check that the executable runs:

```
% bali-phy --version
```

If it does, then your PATH is set up correctly, and you can probably skip the rest of this section.

### 2.5.3. I have a path?

If you installed *BALi-Phy* to the directory `~/Applications`, then you can run `bali-phy` by typing `~/Applications/bali-phy-3.0/bin/bali-phy`. However, it would be much nicer to simply type **bali-phy** and let the computer find the executable for you. This can be achieved by putting the directory that contains the *BALi-Phy* executables into your "path". The "path" is a colon-separated list of directories that is searched to find program names that you type. It is stored in an environment variable called `PATH`.

Setting your `PATH` is also a pre-requisite for running the **bp-analyze** script to summarize your MCMC runs.

### 2.5.4. Examining your PATH

You can examine the current value of this environment variable by typing:

```
% echo $PATH
```

We will assume that you extracted the `bali-phy` archive in `~/Applications` and so you want to add `$HOME/Applications/bali-phy-3.0/bin` to your `PATH`. (If you installed to another directory, replace `$HOME/Applications/bali-phy-3.0/` with that directory.)

### 2.5.5. Adding BALi-Phy to your PATH

The commands for doing this depend on what "shell" you are using. Type **echo \$SHELL** to find out. If your shell is **sh** or **bash** then the command looks like this:

```
% PATH=$HOME/Applications/bali-phy-3.0/bin:$PATH
```

If your shell is **csh** or **tcsh**, then the command looks like this:

```
% setenv PATH $HOME/Applications/bali-phy-3.0/bin:$PATH
```

Note that these commands will only affect the window you are typing in, and will vanish when you reboot.

### 2.5.6. Making the change stick

To make this change survives when you logout or reboot, open your shell configuration file in a text editor, and add the command on a line by itself. This will ensure that it is run every time you log in.

To find the right configuration file, look in your `$HOME` directory for `.profile` (for the Bourne shell **sh**), `.bash_profile` (for **BASH**), or `.login` (for **tcsh**). You may have to create the file if it is not present. On Cygwin, you should put the change in the file `.bashrc`.

If you do not know which directory is your home directory, you can find its full name by typing:

```
% echo $HOME
```

## 2.6. Test the installed software

In order to determine that the software has been correctly installed, and the `PATH` has been correctly set, run the following commands:

```
% bali-phy ~/Applications/bali-phy-3.0/share/doc/bali-phy/examples/sequences/5S-rRNA/5d.fasta --iter=150
% bp-analyze 5d-1/
```

Furthermore, the directory `5d-1` should contain a file called `C1.log`. You should be able to load this file in Tracer, although the chain will not really have converged yet.

## 2.7. Install programs used for viewing the results

- [Tracer](#): MCMC parameter/diagnostic viewer.
- [FigTree](#): Phylogeny Viewer
- [SeaView](#): Alignment viewer.

## 3. Compiling *BALi-Phy*

Compiling *BALi-Phy* is intended to be a relatively painless process. However, most people will want to use the pre-compiled binaries as described in the standard installation instructions at [Section 2, “Installation”](#) instead of compiling BALi-Phy themselves. You might want to compile BALi-Phy yourself if you want to

- run BALi-Phy on a non-Intel CPU (such as ARM64 or Alpha).
- run BALi-Phy on Mac OS X versions older than 10.9
- enable the *draw-tree* program on Windows or Mac (this requires the Cairo graphics library).
- modify the source code and submit a patch with new functionality.
- change the optimization options used to compile BALi-Phy in the pre-compiled binaries.
- compile with debugging options to find the cause of a bug, and maybe fix it.

Otherwise, the pre-compiled binaries will be fine.

### 3.1. Setup

In order to compile BALi-Phy, you need

- a [C++14](#) compiler
- [meson](#) (version  $\geq 0.45$ )

We recommend the GNU C++ Compiler ([GCC](#)) version 5.0 (or higher) or the [Clang](#) compiler version 3.5.0 or higher. The [Cairo](#) graphics library is optional, but if it is missing, the **drawtree** tool that is used to draw consensus trees won't be built. See also [Section 2.7, “Install programs used for viewing the results”](#).

#### 3.1.1. Linux

On Debian and Ubuntu, you can type:

```
% sudo apt-get install g++ git libcairo2-dev
```

If your version of Debian or Ubuntu is recent enough to contain meson version 0.45 or higher, you can install meson with apt-get:

```
% sudo apt-get install meson
% dpkg -s meson | grep Version
Version: 0.45.1-2
```

Otherwise you can install meson through pip3:

```
% sudo apt-get install python3 python3-pip ninja
% python3 -m venv meson
% source meson/bin/activate
% pip3 install meson
```

#### 3.1.2. Mac

On Mac OS X, the simplest way to get a compiler is to install [XCode](#) (version 6 or newer) command line tools,

which come with [clang](#).

```
% xcode-select --install
```

To get the other tools, first install [homebrew](#), and then type:

```
% brew install git meson cairo pandoc
```

### 3.1.3. Windows (native)

The [MSYS2](#) project provides an MINGW64 compiler that can create native windows executables. MSYS2 itself is actually non-native (it is derived from cygwin), and therefore the MSYS2 shell refers to drives as `/c/` instead of `C:/`.

```
% pacman --needed --noconfirm -Sy pacman-mirrors
% pacman -Sy
% pacman -S mingw-w64-x86_64-ninja
% pacman -S mingw-w64-x86_64-toolchain
% pacman -S mingw-w64-python3-pip
% PATH=/c/msys64/mingw64/bin:$PATH # Put the mingw64 executables into your path
% pip3 install meson
```

Keep in mind that MSYS2 keeps its (non-native) executables in `C:/msys64/usr/bin`, while it keeps the (native) MINGW executables in `C:/msys64/mingw64/bin`. If you want to use the native MINGW executables, you need to make sure that `/c/msys64/mingw64/bin/` is in your PATH. If you forget to put the MINGW executables in the path, some of the installed MINGW programs (such as `pip3` above) will show up as missing when you try to run them.

## 3.2. Clone, Configure, Compile

First check out the code using git:

```
% git clone https://github.com/bredelings/BAlI-Phy.git
% cd BAlI-Phy
% git submodule update --init
```

Then run meson to configure the build process:

```
% meson build --prefix=$HOME/Applications/bali-phy-3.0/
```

In the MSYS2 environment, the command is called **meson.py** instead of **meson**:

```
% meson.py build --prefix=$HOME/Applications/bali-phy-3.0/
```

Finally, build and install the software:

```
% ninja -C build
% ninja -C build test
% ninja -C build install
```

The command **bali-phy** and its associated tools should then be located in `~/Applications/bali-phy-3.0/bin/`. To install to another directory *dir*, specify `--prefix=dir` to **meson**.

## 3.3. Options: compiler and linker flags

You can select the C++ compiler by setting the `CXX` variable. A useful example of this is to use **g++-5** on systems where **g++** invokes a compiler that is too old:

```
% CXX=g++-5 meson build --prefix=$HOME/Applications/bali-phy-3.0
```

You may also set compiler and linker options using the `CPPFLAGS`, `CXXFLAGS`, and `LDFLAGS` variables. For example, you can instruct the compiler to use all the features of your chip, instead of producing generic code



that will run anywhere:

```
% CXXFLAGS="-mtune=native -march=native" meson --prefix=$HOME/Applications/bali-phy-3.0
```

For example, you can set the CPPFLAGS and LDFLAGS variables to instruct the compiler where to look for libraries, such as cairo:

```
% CPPFLAGS="-I/usr/local/include" LDFLAGS="-L/usr/local/lib" meson build --prefix=$HOME/Applications/bali-phy-3.0
```

Another useful example of this is to produce an OS X executable on that can run on older versions of OS X:

```
% CXXFLAGS="-mmacosx-version-min=10.9" LDFLAGS="-mmacosx-version-min=10.9" meson build --  
prefix=$HOME/Applications/bali-phy-3.0
```

## 4. Running the program

Here are some examples and explanations of how to run **bali-phy**. You can get an overview of command line options by running **bali-phy --help**.

We recommend running multiple chains in parallel for each command, because

1. You can combine the samples, leading to faster run times.
2. You can compare the runs to determine if the chains have converged.

This can be done simply by starting several instances of the program, and does not require using MPI or special command-line options.

### 4.1. Quick Start

The simplest way to run **Bali-Phy** is to type all the arguments on the command line:

```
% bali-phy sequence-file
```

Here *sequence-file* is a FastA or PHYLIP file containing the sequences you wish to analyze. The filename should end in **.fasta** or **.phy** to indicate which format it is using.

In this simple example, **bali-phy** automatically detects whether *sequence-file* contains DNA, RNA, or Amino-Acids and uses default values for several command line options. Thus, if *sequence-file* contains DNA, then this is equivalent to the more verbose command line

```
% bali-phy sequence-file --alphabet DNA --smodel TN --imodel RS07
```

Here the substitution model is Tamura-Nei, the insertion/deletion model is RS07. If *sequence-file* contains amino acids, then the defaults will be:

```
% bali-phy sequence-file --alphabet Amino-Acids --smodel LG --imodel RS07
```

### 4.2. Command line options

You can specify a more complex substitution model as follows (See [Section 8.2, “Basic CTMC models”](#)):

```
% bali-phy sequence-file --smodel LG+Rates.Gamma+INV
```

You may specify an indel model of **none** to fix the alignment to its initial value, and ignore information in shared insertions or deletions.

```
% bali-phy sequence-file --imodel none
```

### 4.3. Multiple genes/partitions, models, and linking

#### 4.3.1. Multiple genes

You may analyze multiple genes by putting each one in its own data partition:

```
% bali-phy sequence-file1 sequence-file2
```

You should put the data from the first gene in *sequence-file1* and the second gene in *sequence-file2*. The sequence names in both files should be the same. In this scenario, both genes share the same tree, but their alignments vary independently. Furthermore, the branch lengths for each gene are scaled by an independent factor. By default, each partition will have its own default alphabet, substitution model, insertion/deletion model, and tree length.

#### 4.3.2. Multiple partitions with unlinked models

A substitution model or insertion-deletion model that is specified without qualification will apply to every partition. However, each partition will receive its own copy of each model with separate, or "unlinked", parameter values

```
% bali-phy sequence-file1 sequence-file2 --smodel TN --imodel RS07
```

You can select partition-specific values for 4 options: `--smodel`, `--imodel`, `--alphabet`, and `--scale`. For example, to specify different substitution models but the same alphabet:

```
% bali-phy sequence-file1 sequence-file2 --smodel 1:TN --smodel 2:GTR --alphabet DNA
```

You can fix the alignment and ignore insertion/deletion information in one partition, while allowing the alignment to vary and using insertion/deletion information in another partition:

```
% bali-phy sequence-file1 sequence-file2 --imodel 1:RS07 --imodel 2:none
```

#### 4.3.3. Multiple partitions with linked models

You can also specify that two partitions share a single copy of a single substitution model or indel model. This reduces the number of parameters and also pools information between the partitions:

```
% bali-phy sequence-file1 sequence-file2 --smodel 1,2:TN --imodel 1,2:RS07
```

By default each partition has a separate scale, but you can force groups of partitions to share a scale. If you leave the value of the scale blank, the default distribution on scales will be used:

```
% bali-phy sequence-file1 sequence-file2 --smodel 1:TN --smodel 2:GTR --scale 1,2:
```

Finally, you may specify `-Inone` or `--imodel none`, which affects all partitions:

```
% bali-phy sequence-file1 sequence-file2 --smodel 1:TN --smodel 2:GTR -t
```

#### 4.3.4. Linking models via the link command

It is also possible to link partitions by specifying `--link=partitions[:attributes]`. The link command is provided to allow a user to specify a model for each partition separately, and then afterwards choose which partitions to link.

```
% bali-phy sequence-file1 sequence-file2 --smodel 1:TN --smodel 2:TN --link=1,2 -t
% bali-phy sequence-file1 sequence-file2 --smodel TN --link=1,2 -t
```

If the linked partitions are specified as having the same model, BALi-Phy will give an error and refuse to run.

```
% bali-phy sequence-file1 sequence-file2 --smodel 1:TN --link=1,2 -t
bali-phy: Error! Partitions 1 and 2 cannot be linked because they have differing values 'TN' and ''
```

You can also specify which of the 3 attributes "smodel", "imodel", and "scale" are being linked:

```
% bali-phy sequence-file1 sequence-file2 --link=1,2:smodel,scale -t // Don't link the indel model
```

## 4.4. Running on computing clusters

Running **bali-phy** on a computing cluster is not necessary, but can speed up the analysis dramatically. This is because a cluster allows you to run several *independent* MCMC chains simultaneously and pool the resulting samples. You can run multiple chains simultaneously simply by starting several different instances of **bali-phy**. Each instance of bali-phy runs only one chain and does not require using MPI or special command-line options.

This approach to parallel computation is sometimes more efficient than MCMCMC-based parallelism involving heated chains. It is equivalent to running MCMCMC with no temperature difference between chains, with the exception that it allows results from *all* chains to be used, instead of just results from the single "cold" chain. Thus, if you run 10 independent chains in parallel, then you may gather samples 10 times faster than a single chain.

## 4.5. Option files (Scripts)

In addition to using the command line, you may also specify options in a file. Using an option file can be more convenient if you are going to run the same analysis many times, or if the number of options is large. Furthermore, the option file may contain comments and blank lines. Option files are a good to record what options you used in an analysis, and why.

An option file is specified with the command line option `--config file` or `-cfile`. If values for an option are given both on the command line and in an option file, then the command line value overrides the value in the option file.

### 4.5.1. Syntax

Option files use the same option names as the command line. However, the syntax is different: each option is given on its own line using the syntax "**option** = **value**" instead of the syntax "`--option value`". If the option has no value then it is given using the syntax "**option** = **option**".

### 4.5.2. Example

For example, consider the following option file:

```
# sequence data for 3 genes/partitions
align = ITS1.fasta
align = 5.8S.fasta
align = ITS2.fasta

# linked substitution model for 1st and 3rd partition
smodel = 1,3:TN+DP[n=3]

# substitution model for 2nd partition
smodel = 2:TN

# indel model for second partition
imodel = 2:none

# linked scale for 1st and 3rd partition
scale = 1,3:
```

The **align** option indicates sequence files, and has no name on the command line. Lines that begin with **#** are comments, and blank lines are ignored. This is thus equivalent to the rather long command line:

```
% bali-phy ITS1.fasta 5.8S.fasta ITS2.fasta --smodel=1,3:TN+DP[n=3] --smodel=2:TN --imodel=2:none --scale=1,3:
```

### 4.5.3. The configuration file

The file `~/bali-phy` is a special option file called the *configuration file*. If it exists, it is always loaded. Options given on the command line or an option file override values given in `~/bali-phy`.

## 4.6. Examples

Here are some examples which demonstrate how to run *BALi-Phy*. In order to run these examples, you must find the `examples/sequences/` directory which contains the example files. If you downloaded executables and extracted them in the `~/Applications` directory, then the `examples/sequences/` directory will be found at `~/Applications/bali-phy-3.0/share/doc/bali-phy/examples/sequences/`.

Also note that **bali-phy** does *not* run until it is "finished", but continues to gather samples until the user determines that enough samples have been gathered, and stops it. Thus, it is useful to continually examine the output files while the program is running.

### Example 1. No frills

Here we analyze the EF-Tu 5-taxon data set provided with the software.

```
% bali-phy ~/Applications/bali-phy-3.0/share/doc/bali-phy/examples/sequences/EF-Tu/5d.fasta
```

### Example 2. Multiple-Rate Substitution Model

We now modify the previous example by changing the substitution model to allow log-normal-distributed rate variation and invariant sites. The amount of rate variation and the fraction of invariant sites are estimated

```
% bali-phy ~/Applications/bali-phy-3.0/share/doc/bali-phy/examples/sequences/EF-Tu/5d.fasta --smodel  
LG+Rates.logNormal+INV
```

### Example 3. Fixed alignment

Here we use the 5S rRNA 25-taxon data set provided with the software. The `-Inone` option is used, fixing the alignment and making indels non-informative.

```
% bali-phy ~/Applications/bali-phy-3.0/share/doc/bali-phy/examples/sequences/5S-rRNA/25-muscle.fasta -Inone
```

## 5. Input

### 5.1. Sequence formats

*BALi-Phy* can read in sequences and alignments in both FastA and PHYLIP formats. Filenames for FastA files should end in `.fasta`, `.mpfa`, `.fna`, `.fas`, `.fsa`, or `.fa`. Filenames for PHYLIP files should end in `.phy`. If one of these extensions is not used, then *BALi-Phy* will attempt to guess which format is being used.

### 5.2. Is my data set too large?

Large data sets run more slowly than small data sets. We recommend a conservative starting point with few taxa and short sequence lengths. You can then increase the size of your data set until a balance between speed and size is reached. The tool *alignment-thin* described in [Section 11, "Alignment utilities"](#) can be used to construct a smaller data set.

The number of samples that you need depends on whether you are primarily interested in obtaining a point estimate or in obtaining detailed measures of confidence and uncertainty. For detailed measures of confidence and uncertainty you should obtain a minimum of 10,000 samples after the Markov chain converges. For an estimate, you don't need very many samples after convergence. (But you may need many samples to be sure that you've converged!)

See also [Section 4.4, "Running on computing clusters"](#).

#### 5.2.1. Too many taxa?

*BALi-Phy* is quite CPU intensive, and so we recommend using 150 or fewer taxa in order to limit the time required to accumulate enough MCMC samples.

When designing an MCMC analysis, I recommend performing an initial analysis with a much smaller sample size. This smaller analysis will run much faster, and allow discovering mistakes much more quickly. Then, once you are sure that you are running the program correctly and have chosen the best model, you can ramp up the sample size towards your desired number.

### 5.2.2. Sequences too long?

Aligning just a pair of sequences takes  $O(L^2)$  time and memory, where  $L$  represents the sequence length. Therefore sequences longer than (say) 1000 letters become increasingly impractical. However, you might try to see how long you can make your sequences before you run out of memory, or the program becomes too slow.

For multi-gene analyses, two separate data partitions (i.e. genes) of 500 letters will be twice as fast to align as one data partition of 1000 letters. So, it may be possible to analyze several genes as long as each gene individually is not too long.

Also, note that you can sometimes speed up the analysis of protein sequences by coding them as amino acids or codons, rather than nucleotides. This is because it decreases the sequence length.

## 6. Output

### 6.1. Output directory

*BALi-Phy* creates a new directory to store its output files each time it is run. By default, the directory name is the name of the sequence file, with a number added on the end to make it unique. *BALi-Phy* first checks if there is already a directory called *file-1/*, and then moves on to *file-2/*, etc. until it finds an unused directory name.

You can specify a different name to use instead of the sequence-file name by using the `--name` option.

### 6.2. Output files

*BALi-Phy* writes the following output files inside the directory that it creates:

<b>C1.out</b>	Iteration numbers, probabilities, success probabilities for transition kernels, etc..
<b>C1.Pp.fastas</b>	Sampled alignments for partition $p$ including ancestral sequences.
<b>C1.err</b>	Log file for hopefully irrelevant error messages.
<b>C1.MAP</b>	Successive estimates of the MAP alignment, tree and parameters.
<b>C1.log</b>	Numeric parameters: indel and substitution rates, etc.
<b>C1.trees</b>	Tree samples: one sample per line, in Newick format.

For the last two files, each line in these files corresponds to one iteration.

#### 6.2.1. Field names in C1.log

This section explains the meaning of the various field names in the file `C1.log`.

##### 6.2.1.1. Computed parameter names

<b>prior</b>	The log prior probability. This includes the probability of the alignment, since the alignment is not observed.
<b>prior_A<sub>n</sub></b>	The log of the probability $\Pr(A_n   \tau, T, A)$ of the alignment $A_n$ of the $n$ th partition, given the topology $\tau$ , the branch lengths $T$ , and insertion-deletion process parameters $A$ . This log probability is the probabilistic equivalent of a gap penalty on the alignment $A_n$ given the scoring parameters $A$ .

<b>likelihood</b>	The log of the likelihood. Conditional on the alignment, this is determined entirely by the substitution model, and ignores insertions and deletions. This is the probabilistic equivalent of the mismatch penalty.
<b>logp</b>	The log of the probability. The probability is the product of the prior and the likelihood.
<b> A </b>	The total number of alignment columns across all partitions.
<b>#indels<sub>n</sub></b>	The number of indel events in partition <i>n</i> , if we group adjacent indels that occur on the same branch.
<b>#indels</b>	The total number of indel events across all partitions, if we group adjacent indels that occur on the same branch.
<b> indels<sub>n</sub> </b>	The length of indel events in partition <i>n</i> , if we group adjacent indels that occur on the same branch.
<b> indels </b>	The total length of indel events across all partitions, if we group adjacent indels that occur on the same branch.
<b>#substs<sub>n</sub></b>	The unweighted parsimony score for substitutions in partition <i>n</i> .
<b>#substs</b>	The total unweighted parsimony score for substitutions across all partitions.
<b>Scale<sub>n</sub> *  T </b>	The branch lengths for partition group <i>n</i> .

#### 6.2.1.2. Model parameter names

The prefixes "S<sub>n</sub>/" and "I<sub>n</sub>/" will be dropped if not necessary to disambiguate parameters with the same name in different sub-models.

<b>Scale[<i>n</i>]</b>	The average number of substitutions per branch in scale group <i>n</i> . The <i>n</i> th scale group applies to the <i>n</i> th partition, unless multiple partitions are forced to have the same branch length scale using <b>--scale</b> or <b>--link</b> .
<b>S<sub>n</sub>/<i>name</i></b>	Parameter <i>name</i> in the <i>n</i> th substitution model.
<b>I<sub>n</sub>/<i>name</i></b>	Parameter <i>name</i> in the <i>n</i> th insertion/deletion model.

## 6.3. Summarizing the output

This section is primarily oriented to extracting estimates from output files. See [Section 10, "Convergence and Mixing: Is it done yet?"](#) for methods of determine effective sample sizes, and for checking mixing and convergence.

### 6.3.1. Finding the consensus tree (C1.trees)

To compute the majority consensus tree, do the following. (The program [FigTree](#) allows you to view the resulting tree file graphically.)

```
% trees-consensus C1.trees > c50.PP.tree
```

You can (and should) pool results from different MCMC runs by adding multiple tree sample files on the command line. The different MCMC runs should have the same input files and parameters.

```
% trees-consensus dir-1/C1.trees dir-2/C1.trees > c50.PP.tree
```

By default, the first 10% of tree samples are skipped as burn-in. You can specify the number of samples (e.g. 1000) to skip by adding the options **-s1000** or **--skip=1000**. You may also specify a percentage of all samples:

```
% trees-consensus -s20% C1.trees > c50.PP.tree
```

To discard some samples, keeping (say) every 10th sample, you may add the options **-x10** or **--subsample=10**. This can make the program a lot faster, at the possible expense of some loss in accuracy.

```
% trees-consensus -s20% -x10 C1.trees > c50.PP.tree
```

By default, splits are included in the consensus tree if they have a PP greater than 0.5. You can specify a more stringent level (e.g. 0.66) by adding the option `--consensus-PP=0.66` as follows:

```
% trees-consensus -s20% -x10 --consensus-PP=0.66 C1.trees > c66.PP.tree
```

You may also make the program write directly to the output file (e.g. `c66.PP.tree`) by using the more general form `--consensus-PP=0.66:c66.PP.tree`. Leaving off the `:c66.PP.tree` part (as we did above) or specifying `:-` sends the output to the standard output (e.g. the terminal, if not redirected).

```
% trees-consensus -s20% -x10 C1.trees --consensus-PP=0.66:c66.PP.tree
```

You can supply multiple levels and filenames separated by commas. This is faster than running the program multiple times with different consensus levels.

```
% trees-consensus -s20% -x10 C1.trees --consensus-PP=0.5:c50.PP.tree,0.66:c66.PP.tree
```

Finally, you may use the option `--consensus=` instead of the option `--consensus-PP=` if you do not wish the resulting tree to contain embedded posterior probabilities on branches, as well as branch lengths.

```
% trees-consensus -s20% -x10 C1.trees --consensus=0.5:c50.PP.tree,0.66:c66.PP.tree
```

Both the `--consensus=` and `--consensus-PP=` options may be given simultaneously.

See `trees-consensus --help` for a complete list of options.

### 6.3.2. Finding the M.A.P. topology (C1.trees)

To compute the *maximum a posteriori* tree topology do:

```
% trees-consensus --skip=burnin C1.trees --map-tree=MAP.tree
```

The MAP topology may be used instead of a consensus tree when a fully resolved (e.g. bifurcating) tree is required. However, when the topology has many tips, each topology may be sampled only once, leading to low quality estimates of the MAP topology.

The program [FigTree](#) allows you to view the consensus tree graphically.

### 6.3.3. Checking topology convergence (C1.trees)

```
% trees-bootstrap dir-1/C1.trees dir-2/C1.trees
```

This command computes the effective sample size for the posterior probability of each split. It also computes the Average Standard Deviation of Split Frequencies (ASDSF) between two or more independent runs.

See [Section 10, "Convergence and Mixing: Is it done yet?"](#) for more information.

### 6.3.4. Summarizing numerical parameters (C1.log)

This command gives a median and confidence interval, ESS, and a stabilization time:

```
% statreport C1.log > Report
```

This command compares multiple runs to give PSRF and joint ESS values as well:

```
% statreport dir-1/C1.log dir-2/C1.log > Report
```

The program [Tracer](#) allows you to view the same summaries graphically.

See [Section 10, "Convergence and Mixing: Is it done yet?"](#) for more information.

### 6.3.5. Computing an alignment using Posterior Decoding (C1.Pp.fastas)

```
% cut-range --skip=burn-in < C1.Pp.fastas | alignment-max > Pp-max.fasta
```

You can use the program [SeaView](#) to view the alignment graphically.

### 6.3.6. Find the alignment from the maximum a posterior (MAP) point (C1.MAP)

```
% alignment-find < C1.MAP > P1-MAP.fasta
```

### 6.3.7. Create an Au (Alignment Uncertainty) plot (C1.Pp.fastas)

To annotate a specific alignment *alignment.fasta*, choose a fully resolved tree estimate *tree*:

```
% cut-range --skip=burn-in < C1.Pp.fastas | alignment-gild alignment.fasta tree > alignment-AU.prob  
% alignment-draw alignment.fasta --AU alignment-AU.prob > alignment-AU.html
```

The majority consensus tree is usually not fully resolved, so we recommend using the MAP topology instead.

## 6.4. Summarizing the output - scripted

Instead of manually running each of the steps to analyze the output files, you may instead run the PERL script **bp-analyze** to execute these commands. The script will create an HTML page *Results/index.html* that summarizes the posterior distribution.

You may run **bp-analyze** inside the output directory, like this:

```
% bp-analyze --burnin=iterations
```

You may also run it with one or more output directories as arguments, like this:

```
% bp-analyze --burnin=iterations directory-1/ directory-2/
```

In this case, output from multiple runs will be used to assess convergence and mixing, as well as to increase the precision of the estimates.

All the commands that are executed by **bp-analyze** will be logged to *Results/bp-analyze.log*. You can also see these commands as they are executed by supplying the **--verbose** option:

```
% bp-analyze --burnin=iterations --verbose
```

#### 6.4.1. Meaning of generated files

The *Results/* directory will contain the following useful files:

<b>Report</b>	A summary of numerical parameters: credible intervals and mixing.
<b>consensus</b>	A summary of supported splits (clades).
<b>c-levels.plot</b>	The number of splits (clades) supported at each LOD level.
<b>c50.tree</b>	The majority consensus topology + branch lengths (Newick format)
<b>c50.PP.tree</b>	The majority consensus topology + branch lengths + Posterior Probabilities (Newick format)
<b>MAP.tree</b>	An estimate of the MAP topology + branch lengths (Newick format)

The following files will be generated to summarize alignment uncertainty, unless the analysis uses a fixed alignment.



<b>MAP.fasta</b>	An estimate of the MAP alignment.
<b>P<sub>p</sub>-max.fasta</b>	An estimate of the alignment for partition $p$ using maximum posterior decoding.
<b>MAP-AU.html</b>	An AU plot of the MAP alignment (AA/DNA color-scheme).
<b>P<sub>p</sub>-max-AU.html</b>	An AU plot of the maximum posterior decoding alignment for partition $p$ (AA/DNA color-scheme).

The following files describe convergence and mixing:

<b>partitions.bs</b>	Confidence intervals on the support for partitions, generated using a block bootstrap.
<b>partitions.SRQ</b>	A collection of SRQ plots for the supported partitions.
<b>c50.SRQ</b>	An SRQ plot for the majority consensus tree.

The SRQ plots can be viewed by typing "`plot 'file' with lines`" in *gnuplot*.

#### 6.4.2. Mixing/partitions.bs: partition mixing

This file reports the quality of estimates of support for each partition in terms of the posterior probability (PP) and log-10 odds (LOD). It also reports the auto-correlation time (ACT), the effective sample size (Ne), the number of samples that support (1) or do not support (0) the partition, and the number of regenerations. Only partitions with PP > 0.1 are shown by default.

## 7. Models and Priors

### 7.1. Models and distributions are functions

Models and probability distributions are treated as functions in Bali-Phy because all of them have parameters or arguments. Parameters have names in Bali-Phy. Parameter values are specified using square brackets as follows:

```
HKY[kappa=2]           // model
log[x=2]               // function
Normal[mean=0,sigma=1] // probability distribution
```

It is possible to specify parameter values by position instead of by name:

```
HKY[2]
log[2]
Normal[0,1]
```

It is even possible to mix positional and named arguments, as long as all the positional arguments come before all the named arguments:

```
Normal[0,sigma=1] // OK
Normal[mean=0,1]  // not OK
```

The order and type of parameters for a function can be found with the **help** command. For example,

```
% bali-phy help HKY
```

A value must be given for each parameter, unless the parameter has a default value (See [Section 7.4, "Default values and default priors"](#)).

### Note

If you are using the C-shell command line shell (**csh** or **tcsh**), then it will try to interpret the square brackets [...] as an array reference and give the rather confusing error message "bali-phy: No match."

```
% bali-phy file.fasta --smodel=HKY[kappa=2]
bali-phy: No match.
```

To avoid this, put quotes around the substitution model, like this:

```
% bali-phy file.fasta --smodel="HKY[kappa=2]"
```

## 7.2. Priors

### 7.2.1. Specifying priors

Priors on model parameters are specified by giving a random value. Random values can be obtained from distributions using the function `Sample`. For example, this places a log-normal prior on the parameter `kappa` of the `HKY` model:

```
HKY[kappa=Sample[logNormal[1,1]]]
```

You can write `~Dist` as a shorthand for `Sample[Dist]`:

```
HKY[kappa=~logNormal[1,1]]
```

The `=~` can be further shortened to just `~`:

```
HKY[kappa~logNormal[1,1]]
```

### 7.2.2. Random function arguments

It also is possible to use random values as inputs to other functions. For example:

```
Add[1,~Exponential[10]]
```

In such cases the parameter value should be specified with `=`, as in the following example:

```
RS07[meanIndelLength=Add[1,~Exponential[10]]]
```

### 7.2.3. Distributions are not random values

Random values and distributions have different types. For example, the following is of type `Distribution[Double]`:

```
Uniform[0,1]
```

In contrast, the following are both of type `Double`:

```
Sample[Uniform[0,1]]
~Uniform[0,1]
```

This is important when passing distributions as arguments to other distributions and functions. For example, the distribution `iid` is used to generate a specific number of samples from another distribution. Thus, it needs to receive a distribution as an argument:

```
~iid[4,Normal[0,1]]    // OK    : 4 samples from the Normal[0,1] distribution
~iid[4,~Normal[0,1]]   // not OK: 4 samples from ... a random number?
```

(See [Section 7.5, "Argument and result types"](#).)

## 7.3. Models and '+' notation

Bali-Phy allows using `+` as a short-cut to combine models. Models expressed this way are rewritten internally to a more precise form:

```

WAG+F           // rewritten to RCTMC[WAG,F]
M0+F3x4         // rewritten to RCTMC[M0,F3x4]
HKY+Rates.Gamma // rewritten to Rates.Gamma[submodel=HKY]
HKY+INV          // rewritten to INV[submodel=HKY]

```

These examples illustrate the two rules for rewriting `+`-expressions. In the first rule, an expression such as `M1[args1]+M2[args2]` is rewritten to `RCTMC[M1[args1],M2[args2]]` if `M1` has type `ExchangeModel[a]` and `M2` has type `FrequencyModel[a]`. This allows natural syntax for creating a reversible continuous-time Markov chain (RCTMC) from an exchange model (such as WAG or M0) and a frequency model (such as +F or +F3x4).

In the second rule, an expression such as `model+M[args]` is rewritten to `M[args,submodel=model]` if `M` has a `submodel` argument. When `+` occurs multiple times, the submodel is assumed to be as large as possible thus:

```
HKY+F+Rates.Gamma+INV
```

is rewritten to

```
INV[submodel=Rates.Gamma[submodel=RCTMC[HKY,F]]]
```

This allows a simple method for combining models, when one model is an argument to another model.

## 7.4. Default values and default priors

Some function arguments have default values. For example, the `Rates.Gamma` parameter `n` has a default value of 4. Thus the following are equivalent:

```

HKY+Rates.Gamma[n=4]+INV
HKY+Rates.Gamma+INV

```

When the default value is random, then the argument has a default prior. For example, the `kappa` parameter of `HKY` has a default value of `~logNormal[log[2],0.25]`, so the following are equivalent:

```

HKY[kappa~logNormal[log[2],0.25]]
HKY

```

The `help` command can be used to determine the default value for a parameter, if there is one.

## 7.5. Argument and result types

Every function has a *result type*, as well as an *argument type* for each argument. The argument type specifies what kind of arguments are acceptable, and the result type specifies what kind of result the function produces. Types include **Int** for integers, **Double** for double-precision floating point numbers, and **String** for text strings. Integer arguments are implicitly converted to **Double** when the argument type is **Double**.

Some types contain parameters. For example `List[Int]` indicates a list of integers and `List[Double]` indicates a list of real numbers. In order to indicate a list of unknown type, we use a *type variable* `a` and write `List[a]`. Type variables always begin with a lower-case letter. They are able to match any specific type, and their value is found by pattern-matching. For example, the function `Add[x,y]` takes two arguments of type `a` and has a result of type `a`. Thus:

```

Add[1,2]           // arguments are a=Int, so result is of type Int
Add[1.0,2.0]       // arguments are a=Double, so result is of type Double

```

`Pair[a,b]` is a parameterized type that can be specialized to (for example) `Pair[String,Double]` and `Pair[Int,Int]`.

Types for components of substitution models are often parameterized by type of the alphabet. For example, `HKY` has a result type of `ExchangeModel[a]`, where `a` could be `DNA` or `RNA`. The use of alphabet types in substitution models prevents combining substitution models with mismatched alphabets.

## 8. Substitution models

Substitution models can be specified using notation like:

See section [Section 7, “Models and Priors”](#) for the general syntax.

## 8.1. Default substitution models

If the substitution model is not specified, then the default model for the alphabet is used. For DNA or RNA, the default model is TN. For Triplets, the default is TN+x3. For Codons, the default model is M0. For Amino-Acids, the default model is LG.

## 8.2. Basic CTMC models

The basic substitution models in *BALI-Phy* are continuous-time Markov chains (CTMC). CTMC models can be characterized by transition rates  $Q_{ij}$  from letter  $i$  to letter  $j$ . After a given time  $t$  the probability for transition from state  $i$  to state  $j$  is given by

$$P(t)_{ij} = e^{Q_{ij} \times t}$$

using a matrix exponential. Because the CTMC models used in *BALI-Phy* are all reversible, the rate matrix for these reversible models can be decomposed into a symmetric matrix  $S$  and equilibrium frequencies  $\pi$  as follows:

$$Q_{ij} = S_{ij} \times \pi_j$$

The matrix  $S$  is called the exchangeability matrix, and represents how exchangeable letters  $i$  and  $j$  are, independent of their frequencies.

The basic CTMC models are EQU, HKY, TN, GTR, HKY+x3, TN+x3, GTR+x3, JTT, WAG, LG, and M0. Each of these models is a way of specifying the exchangeability matrix  $S_{ij}$ .

**Table 1. Substitution Models**

Model	Alphabet	Parameters
<b>EQU</b>	any	none
<b>HKY</b> Hasegawa, Kishino, Yano (1985)	DNA or RNA	kappa: the ts/tv ratio.
<b>TN</b> Tamura, Nei (1993)	DNA or RNA	kappaPur: the purine ts/tv ratio. kappaPyr: the pyrimidine ts/tv ratio.
<b>GTR</b> General Time-Reversible Tavare (1986)	DNA or RNA	$S_{i \neq j}$
<b>JTT</b> Jones, Taylor, Thornton (1992)	Amino-Acids	none.
<b>WAG</b> Whelan and Goldman (2001)	Amino-Acids	none.
<b>LG</b> Le and Gascuel (2008)	Amino-Acids	none.
<b>Empirical[<i>file</i>]</b>	Amino-Acids	none.
<b>HKY+x3</b> <b>TN+x3</b> <b>GTR+x3</b>	Triplets	<i>nuc-model</i> parameters.
<b>M0</b> <b>M0[,HKY]</b> (default) <b>M0[,GTR]</b> Nielsen and Yang (1998)	Codons	omega: the dN/dS ratio

### 8.3. Substitution Frequency models

The rate matrix  $Q_{ij}$  can be more generally expressed as

$$Q_{ij} = S_{ij} \times \frac{\pi_j^f}{\pi_i^{1-f}},$$

where  $f$  ranges from 0 to 1. Here the parameter  $f$  specifies the relative importance of unequal conservation ( $f = 0$ ) and unequal replacement ( $f = 1$ ) in maintaining the equilibrium frequencies  $\pi$ .

In fact, this can be generalized even further to

$$Q_{ij} = S_{ij} \times R(\pi)_{ij}$$

where

$$\pi_i \times R_{ij} = \pi_j \times R_{ji}.$$

These models can therefore be expressed as a combination of an "exchange model" (for  $S$ ) and a "frequency model" (for  $R$ ).

**Table 2. Frequency Models**

Model	Alphabet	Parameters
<b>F</b> Whelan and Goldman (2001)	any	$\pi[l]$ : frequency of letter $l$ .
<b>F61</b> Synonym of F for codon models	any	$\pi[l]$ : frequency of letter $l$ .
<b>gwf</b> Goldman and Whelan (2002)	any	$f$ : determines cause of high-frequency letters. $\pi[l]$ : frequency of letter $l$ .
<b>F1x4</b> Single nucleotide frequency model	Triplets	$\pi[l]$ : frequency of nucleotide $l$ .
<b>F3x4</b> Independent nucleotide frequency model	Triplets	$\pi 1[l]$ : frequency of letter $l$ in codon position 1. $\pi 2[l]$ : frequency of letter $l$ in codon position 2. $\pi 3[l]$ : frequency of letter $l$ in codon position 3.
<b>MG94</b> Muse and Gaut (1994) Single nucleotide frequency model	Triplets	$\pi[l]$ : frequency of nucleotide $l$ .
<b>MG94w9</b> Muse and Gaut (1994) Independent nucleotide frequency model	Triplets	$\pi[l]$ : frequency of letter $l$ in codon position 1. $\pi[l]$ : frequency of letter $l$ in codon position 2. $\pi[l]$ : frequency of letter $l$ in codon position 3.

### 8.4. Substitution Mixture Models

Complex substitution models in *BALI-Phy* are constructed as mixtures of reversible CTMC models (see [Section 8.2, "Basic CTMC models"](#)) that run at different rates (e.g.  $\Gamma_4 + \text{INV}$ ) or have different parameters (e.g. an M2a codon model).

**Table 3. CTMC Mixture Models**

Model	Alphabet	Parameters
<b>model + Rates.Gamma</b> <b>model + Rates.Gamma[4]</b> (default) Yang (1994)	<b>model</b> alphabet	<b>sigmaOverMu</b> noise to signal ratio for Gamma(a,b). Also equal to $1/\text{sqrt}(a)$ . <b>a</b> parameter for Gamma(a,b) <b>b</b> parameter for Gamma(a,b)
<b>model + Rates.logNormal</b> <b>model +</b>	<b>model</b> alphabet	<b>sigmaOverMu</b> noise to signal ratio for logNormal. <b>pInv</b> Fraction of invariant sites.

**Rates.logNormal[4]**  
(default)

<i>model</i> + INV	<i>model</i> alphabet	<b>pInv</b>	Fraction of invariant sites.
<b>M1a</b> <b>M1a[HKY,F61]</b> (default) <b>M1a[GTR,F3x4]</b> Wong, et. al. (2004)	Codons	<b>w1</b> <b>f1</b> <b>f2</b>	the conserved dN/dS ratio the fraction of conserved sites among non-positively selected sites the fraction of neutral sites among non-positively selected sites
<b>M2a</b> <b>M2a[HKY,F61]</b> (default) <b>M2a[GTR,F3x4]</b> Wong, et. al. (2004)	Codons	<b>w1</b> <b>f1</b>  <b>f2</b> <b>posW</b> <b>posP</b>	the conserved dN/dS ratio the fraction of conserved sites among non-positively selected sites the fraction of neutral sites among non-positively selected sites the positively selected dN/dS ratio the fraction of positively selected sites
<b>M2a_Test</b> <b>M2a_Test[HKY,F61]</b> (default) <b>M2a_Test[GTR,F3x4]</b> Wong, et. al. (2004)	Codons	<b>w1</b> <b>f1</b>  <b>f2</b>  <b>posW</b> <b>posP</b> <b>posSelection</b>	the conserved dN/dS ratio the fraction of conserved sites among non-positively selected sites the fraction of neutral sites among non-positively selected sites the positively selected dN/dS ratio the fraction of positively selected sites. model selector (1 if positive selection, 0 if not).
<b>M3</b> <b>M3[HKY,F61]</b> (default) <b>M3[GTR,MG94,n=4]</b> Yang, et. al. (2000)	Codons	<b>M3.omegas[i]</b> <b>M3.ps[i]</b>	the conserved dN/dS ratio for category i the fraction of sites in category i
<b>M3_Test</b> <b>M3_Test[HKY,F61]</b> (default) <b>M3_Test[GTR,MG94]</b> Yang, et. al. (2000)	Codons	<b>M3.omegas[i]</b> <b>M3.ps[i]</b> <b>posW</b> <b>posP</b> <b>posSelection</b>	the conserved dN/dS ratio for category i the fraction of sites in category i the positively selected dN/dS ratio the fraction of positively selected sites. model selector (1 if positive selection, 0 if not).
<b>M7</b> <b>M7[4, HKY, F61]</b> (default) <b>M7[6, GTR, F3x4]</b> Yang, et. al. (2000)	Codons	<b>mu</b> <b>a</b> <b>b</b>	mean of the Beta(a,b) distribution. parameter of Beta(a,b) distribution parameter of Beta(a,b) distribution
<b>M8a</b> <b>M8a[4, HKY, F61]</b> (default) <b>M8a[6, GTR, F3x4]</b> Yang, et. al. (2000)	Codons	<b>mu</b> <b>a</b> <b>b</b> <b>posW</b> <b>posP</b>	mean of the Beta(a,b) distribution. parameter of Beta(a,b) distribution parameter of Beta(a,b) distribution the positively selected dN/dS ratio the fraction of neutral sites.
<b>M8</b> <b>M8[4, HKY, F61]</b> (default) <b>M8[6, GTR, F3x4]</b> Yang, et. al. (2000)	Codons	<b>mu</b> <b>a</b> <b>b</b> <b>posW</b> <b>posP</b>	mean of the Beta(a,b) distribution. parameter of Beta(a,b) distribution parameter of Beta(a,b) distribution the positively selected dN/dS ratio the fraction of positively selected sites.
<b>M8a_Test</b> <b>M8a_Test[4, HKY, F61]</b> (default) <b>M8a_Test[6, GTR, F3x4]</b> Yang, et. al. (2000)	Codons	<b>mu</b> <b>a</b> <b>b</b> <b>posW</b> <b>posP</b> <b>posSelection</b>	mean of the Beta(a,b) distribution. parameter of Beta(a,b) distribution parameter of Beta(a,b) distribution the positively selected dN/dS ratio the fraction of positively selected sites. model selector (1 if positive selection, 0 if not).
<b>branch-site</b>	Codons	<b>fs[1]</b>	the fraction of conserved sites on background branches.

<b>branch-site[HKY,F61]</b> (default)	<b>fs[2]</b>	the fraction of neutral sites on background branches.
<b>branch-site[GTR,F3x4]</b> Zhang, et. al. (2005)	<b>omegas[1]</b>	the conserved dN/dS ratio.
	<b>posW</b>	the positively selected dN/dS ratio.
	<b>posP</b>	the fraction of sites (neutral or conserved) that switch to positive selection on foreground branches.
	<b>posSelection</b>	model selector (1 if positive selection, 0 if not).

## 8.5. The branch-site substitution model

In order to use the branch-site substitution model, the user needs to

- Specify an unrooted tree topology in Newick format.
- Label foreground branches on the tree using NHX attributes.
- Disable topology changes in order to fix the tree topology (branch lengths are estimated).

Here is an example tree file:

### Example 4. An initial tree file with branch lengths

```
((((A1:0.1, B1:0.1):0.1,(C1:0.1, D1:0.1):0.1):0.1,((E1:0.1, F1:0.1):0.1,(G1:0.1, H1:0.1):0.1):0.1,(((A2:0.1, B2:0.1):0.1,(C2:0.1, D2:0.1):0.1):0.1,((E2:0.1, F2:0.1):0.1,(G2:0.1, H2:0.1):0.1):0.1):[&&NHX:foreground=1]0.2);
```

Any branch lengths provided will be used as initial values in the MCMC analysis. However, it is not necessary to provide them:

### Example 5. An initial tree file without branch lengths

```
((((A1, B1),(C1, D1)),((E1, F1),(G1, H1))),(((A2, B2),(C2, D2)),((E2, F2),(G2, H2)))):[&&NHX:foreground=1];
```

The NHX attribute must be applied to the branch, not the node. Therefore it must occur after a colon. Multiple branches may be marked as foreground branches.

An example command line is as follows:

```
% bali-phy alignment.fasta --smodel branch-site[HKY,F3x4] --disable=topology --tree=tree.tree
```

The posterior probability of positive selection is the posterior mean of the posSelection parameter. This may be computed using the statreport program with the **--mean** option.

In case this probability is extremely close to 1 or 0, you may wish to add the option **--Rao-Blackwellize BranchSiteTest:posSelection**. This will report the log-probability of positive selection each iteration. The user may exponentiate the reported values and then average them (using R, for example) in order to compute a more accurate estimate of the posterior probability of positive selection.

## 8.6. Substitution model examples

Example: `--smodel WAG+F+Rates.logNormal+INV`

Example: `--smodel WAG+Rates.logNormal+INV` (same as above)

Example: `--smodel LG+gwF+Rates.logNormal+INV`

Example: `--smodel EQU --alphabet Triplets`

Example: `--smodel HKY`

Example: `--smodel M0`

Example: `--smodel M0+F1x4`

Example: --smodel M2a

Example: --smodel M2a[HKY] (same as above)

### 8.7. Genetic Codes

When using a codon-based substitution model like **M0**, you may select the genetic code by specifying **--alphabet Codons[,genetic-code]**. Available genetic codes are **standard**, **mt-vert**, **mt-invert**, **mt-yeast**, **mt-protozoan**.

If the genetic code is not specified, then the standard code is used:

```
% bali-phy sequence-file --smodel M0 --alphabet Codons
% bali-phy sequence-file --smodel M0 --alphabet Codons[RNA]
```

These examples specify the vertebrate mitochondrial code:

```
% bali-phy sequence-file --smodel M0 --alphabet Codons[DNA,mt-vert]
% bali-phy sequence-file --smodel M0 --alphabet Codons[,mt-vert]
```

## 9. Insertion/deletion models

The current models are RS05, RS07, and **none**. Models (except for **none**) can be specified using notation like:

```
RS07[logLambda~logLaplace[-4,0.707],meanIndelLength=2]
```

See section [Section 7, “Models and Priors”](#) for the general syntax. RS07 is the default insertion/deletion model.

Each of these models is a probability distribution on pairwise alignments. The probability distribution on multiple sequence alignments  $\Pr(A \mid T, \tau, A)$  is constructed by factoring the multiple sequence alignment into pairwise alignments along each branch of the tree, as described in Redelings and Suchard (2005).

**Table 4. Substitution Models**

Model	Parameters	Description
<b>RS05</b> Redelings and Suchard (2005)	$\delta$ : the gap-opening probability $\epsilon$ : the gap-extension probability	Gap lengths are geometrically distributed with extension probability $\epsilon$ . This indel model is independent of the branch length connecting the ancestor and descent sequences.
<b>RS07</b> Redelings and Suchard (2007)	$\lambda$ : the insertion and deletion rate $\epsilon$ : the gap-extension probability	Gap lengths are geometrically distributed with extension probability $\epsilon$ . The probability of an indel event depends on the branch length in this model.
<b>none</b>		Indicates the lack of a model.

Specifying an indel model of **none** for a given partition results in fixing the alignment for that partition to its initial value, and ignoring information in shared insertions or deletions.

## 10. Convergence and Mixing: Is it done yet?

When using Markov chain Monte Carlo (MCMC) programs like *MrBayes*, *BEAST* or *Bali-Phy*, it is hard to determine in advance how many iterations are required to give a good estimate. The number depends on the specific data set that is being examined. As a result, *Bali-Phy* relies on the user to analyze the output of a running chain periodically in order to determine when enough samples have been obtained. This section describes a number of techniques to diagnose when more samples must be taken.

Some of the better diagnostics for lack of convergence rely on running at least 4 independent copies of the Markov chain (preferably 10) from different random starting points to see if the sampled posterior distributions for each chain are the same. Unfortunately, when the distributions all seem to be this same, this doesn't *prove* that they have all converged to the equilibrium distribution. However, if the distributions are different then you



can reject either convergence or good mixing.

## 10.1. Definition of Convergence

Convergence refers to the tendency of a Markov chain to "forget" its starting value and become typical of its equilibrium distribution. Note that convergence is a property of the Markov chain itself, not of individual runs of the Markov chain. Ideally a number of individual runs should be examined in order to determine how many initial iterations to discard as "burnin".

## 10.2. Definition of Mixing

In MCMC, each sample is not fully independent of previous samples. In fact, even after a Markov chain has converged, it can get "stuck" in one part of the parameter space for a long time, before jumping to an equally important part. When this happens, each new sample contributes very little new information, and we need to obtain many more samples to get good precision on our parameter estimates. In such a case, we say that the chain isn't "mixing" well.

## 10.3. Diagnostics: Variation in split frequencies across runs (ASDSF/MSDSF)

### 10.3.1. ASDSF and MSDSF

To calculate the ASDSF and MSDSF run:

```
% trees-bootstrap dir-1/C1.trees dir-2/C1.trees ... dir-n/C1.trees > partitions.bs
```

For each split, the SDSF value is just the standard deviation across runs of the Posterior Probabilities for that split. By averaging the resulting SDSF values across splits, we may obtain the ASDSF value (Huelsenbeck and Ronquist 2001). This is commonly considered acceptable if it is  $< 0.01$ .

However, it is also useful to consider the maximum of the SDSF values (MSDSF). This represents the range of variation in PP across the runs for the split with the most variation.

### 10.3.2. Split-frequency comparison plot

To generate the split-frequency comparison plot, you must have R installed. Locate the script `compare-runs.R`. Then run:

```
% trees-bootstrap dir-1/C1.trees dir-2/C1.trees ... dir-n/C1.trees --LOD-table=LOD-table > partitions.bs
% R --slave --vanilla --args LOD-table compare-SF.pdf < compare-runs.R
```

Following [Beiko et al \(2006\)](#), this displays the variation in estimates of split frequencies across runs. Splits are arranged on the x-axis in increasing order of Posterior Probability (PP), which is obtained by averaging over runs. We then plot a vertical bar from the minimum PP to the maximum PP.

## 10.4. Diagnostics: Potential Scale Reduction Factors (PSRF)

Potential Scale Reduction Factors check that different runs have similar posterior distributions. Only numerical variables may have a PSRF. To calculate the PSRF for each numerical parameter, you may run:

```
% statreport dir-1/C1.log dir-2/C2.p ... dir-n/C1.log > Report
```

The PSRF is a ratio of the width of the pooled distribution to the average width of each distribution, and should ideally be 1. The PSRF is customarily considered to be small enough if it is less than 1.01.

We compare the PSRF based on the length of 80% credible intervals (Brooks and Gelman 1998) and report the result as PSRF-80%CI. For integer-valued parameters, we avoid excessively large PSRF values by subtracting 1 from the width of the pooled CI.

We also report a new PSRF that is more sensitive for integer distributions. For each individual distribution, we find the 80% credible interval. We divide the probability of that interval (which may be more than 80%) by the probability of the same interval under the pooled distribution. The average of this measure over all distributions gives us a PSRF that we report as PSRF-RCF.

This convergence diagnostic gives a criterion for detecting when a parameter value has stabilized at different values in several independent runs, indicating a lack of convergence. This situation might occur if different runs of the Markov chain were trapped in different modes and failed to adequately mix between modes.

## 10.5. Diagnostics: Effective sample sizes (ESS)

### 10.5.1. ESS for numerical values

To calculate the split ESS values, run:

```
% statreport dir-1/C1.log dir-2/C1.log ... dir-n/C1.log > Report
```

We calculate effective sample sizes based on integrated autocorrelation times. This method has the nice property that simply duplicating every sample does not increase the ESS.

The program [Tracer](#) also computes ESS values.

### 10.5.2. ESS for split frequencies

To calculate the split ESS values, run:

```
% trees-bootstrap dir-1/C1.trees dir-2/C1.trees ... dir-n/C1.trees > partitions.bs
```

To compute the ESS for a split, we consider the presence or absence of a split in each iteration as a series of binary values. We compute the integrated autocorrelation time for this binary sequence, which leads to an ESS. This approach is similar to dividing the iterations into blocks and computing the ESS on the PP estimates in the blocks. It is also similar to estimating the variance reduction under a block bootstrap.

## 10.6. Diagnostics: Stabilization

### 10.6.1. Stabilization of numerical values

To obtain estimates of the stabilization time for each numerical parameter, you may run:

```
% statreport C1.log > Report
```

Each series of values is counted as having stabilized after the series crosses its upper and then lower 95% confidence bounds twice (if the initial value is below the median) or crosses its lower and then upper confidence bounds twice (if the initial value is above the median). The confidence bounds are those based on its equilibrium distribution as calculated from the last third of the values in the sequence.

### 10.6.2. Stabilization of tree topologies and tree distances

In addition to examining convergence diagnostics for continuous parameters, it is important to examine convergence diagnostics for the topology as well ([Beiko et al 2006](#)). In theory, we recommend the web tool [Are We There Yet \(AWTY\)](#) (Wilgenbush et al, 2004). However, AWTY gives incorrect results if you upload plain NEWICK tree samples -- which is what BALi-Phy outputs. Therefore, if you wish to use AWTY, you must convert the tree samples files to NEXUS before you upload them to AWTY in order to get correct results.

It is also possible to assess stabilization of tree topologies using tools distributed with *bali-phy* by using commands like the following. Here, sub-sampling and burnin does not apply to the equilibrium tree files. Also, note that you need to manually construct the equilibrium samples, which we recommend to contain at least 500 trees; you might do this by sub-sampling using the *BALi-Phy* tool **sub-sample**.

1. To report the average distances within and between two tree samples:

```
% trees-distances --skip=burnin --subsample=factor compare dir-1/C1.trees dir-2/C1.trees
```

2. To compute the distance from each tree in C1.trees to all trees equilibrium.trees, as a time series:

```
% trees-distances --skip=burnin --subsample=factor convergence C1.trees equilibrium.trees
```

3. To assess when the above time series stabilizes:

```
% trees-distances --skip=burnin --subsample=factor converged Cl.trees equilibrium.trees
```

The stabilization criterion is the same one described above for numerical values.

Note that the running time is the product of the number of trees in the two files. Therefore, comparing two complete tree samples without sub-sampling will take too long.

## 11. Alignment utilities

Most of these tools will describe their options if given the "`--help`" argument on the command line.

### 11.1. alignment-info

Show basic information about the alignment:

```
% alignment-info file.fasta
% alignment-info file.fasta file.tree
```

### 11.2. alignment-cat

To select columns from an alignment:

```
% alignment-cat -c1-10,50-100,600- file.fasta > result.fasta
% alignment-cat -c5-250/3 file.fasta > first_codon_position.fasta
% alignment-cat -c6-250/3 file.fasta > second_codon_position.fasta
```

To concatenate two or more alignments:

```
% alignment-cat file1.fasta file2.fasta > all.fasta
```

### 11.3. alignment-thin

Remove columns without a minimum number of letters:

```
% alignment-thin --min-letters=5 file.fasta > file-thinned.fasta
```

Remove sequences:

```
% alignment-thin --remove=seq1,seq2 file.fasta > file2.fasta
```

Remove short sequences:

```
% alignment-thin --longer-than=250 file.fasta > file-long.fasta
```

Remove sequences while preserving sequence diversity:

```
% alignment-thin --down-to=30 file.fasta > file-30taxa.fasta
% alignment-thin --down-to=30 file.fasta --keep=seq1,seq2 > file-30taxa.fasta
```

Remove sequences that are missing conserved columns:

```
% alignment-thin --remove-crazy=10 file.fasta > file2.fasta
```

### 11.4. alignment-draw

Draw an alignment to HTML, optionally coloring residues by AU.

```
% alignment-draw file.fasta --show-ruler --color-scheme=DNA+contrast > file.html
% alignment-draw file.fasta --show-ruler --AU=file-AU.prob --color-scheme=DNA+contrast+fade+fade+fade+fade > file-
```

## 11.5. alignment-find

Find the last (or first) FastA alignment in a file.

```
% alignment-find --first < file.fastas > first.fasta
% alignment-find < file.fastas > last.fasta
```

## 11.6. alignment-indices

Turn columns from a template alignment into alignment constraints:

```
% alignment-indices template.fasta > constraints.txt
% alignment-indices -c100-110,200,300- template.fasta > constraints.txt
```

Each line in this file corresponds to one alignment column.

## 11.7. alignment-chop-internal

Remove internal-node ancestral sequences from an alignment. (This probably only works for alignments output by bali-phy.)

```
% alignment-chop-internal file.fasta > file-chopped.fasta
```

# 12. Tree utilities

## 12.1. trees-consensus

This program analyzes the tree sample contained in *file*. It reports the MAP topology, the supported taxa partitions (including partial partitions), and the majority consensus topology.

## 12.2. trees-bootstrap

Usage: trees-bootstrap *file1* [*file2* ... ] --predicates *predicate-file* [OPTIONS]

This program analyzes the tree samples contained in *file1*, *file2*, etc. It gives the support of each tree sample for each predicate in *predicate-file*, and reports a confidence interval based on the block bootstrap.

Each predicate is the intersection of a set of partitions, and is specified as a list of partitions or (multifurcating) trees, one per line. Predicates are separated by blank lines.

## 12.3. trees-to-SRQ

Usage: trees-to-SRQ *predicate-file* [OPTIONS] *trees-file*

This program analyzes the tree samples contained in *trees-file*. It uses them to produce an SRQ plot for each predicate in *predicate-file*. Plots are produced in *gnuplot* format, with one point per line and with plots separated by a blank line.

If `--mode sum` is specified, then a "sum" plot is produced instead of an SRQ plot. In this plot, the slope of the curve corresponds to the posterior probability of the event. If the `--invert` option is used then the slope of the curve correspond to the probability of the inverse event. This is recommended if the probability of the event is near 1.0, because the sum plot does not distinguish variation in probabilities near 1.0 well.

# 13. Frequently Asked Questions (FAQ)

## 13.1. Input files

13.1.1. [Does BALi-Phy accept the wildcard characters "N" or "X"? How does it treat them?](#)

13.1.2. [Does BALi-Phy accept "?" characters?](#)

### 13.1.3. [Does BALi-Phy accept the characters "R" and "Y", etc.?](#)

#### 13.1.1. Does BALi-Phy accept the wildcard characters "N" or "X"? How does it treat them?

Yes, BALi-Phy accepts the wildcard characters "N" (for DNA) and "X" (for proteins). These characters indicate that some letter is present (as opposed to a gap), but that you don't know *which* letter it is.

#### 13.1.2. Does BALi-Phy accept "?" characters?

No. "?" characters are often used to indicate *either* letter presence (e.g. "N", "X") *or* absence (e.g. "-"). BALi-phy will insist that you replace each "?" with either "N"/"X" or "-" to indicate which one you mean.

(Most programs ignore indels and consider only substitutions, and in that case "N" and "-" have the same effect on the likelihood or parsimony score. However, since BALi-Phy takes indels into account, these two alternatives are quite different.)

#### 13.1.3. Does BALi-Phy accept the characters "R" and "Y", etc.?

Yes. BALi-Phy accepts the characters Y, R, W, S, K, M, B, D, H, and V for DNA, RNA, and Codon alphabets. BALi-Phy also accepts the characters B, Z, and J for amino acids. These characters indicate partial knowledge about a letter. For example, R indicates that a nucleotide is present, and is a puRine (A or G). J indicates that an amino acid is present and is either I or L. (Note that sequences sometimes contain such ambiguity codes because the DNA that was sequenced contains *both* values. This might occur when sequencing a heterozygote or when sequencing pooled DNA from several individuals. However, the model in BALi-Phy (and other phylogeny inference programs) is that only one letter is correct, but we do not know which one it is. This is probably not problematic when dealing with pooled sequences, but should be considered.)

## 13.2. Running bali-phy.

### 13.2.1. [Can I fix the alignment and ignore indel information, like MrBayes, BEAST, PhyloBayes and other MCMC programs?](#)

### 13.2.2. [Can I fix the tree topology, while allowing the alignment to vary?](#)

### 13.2.3. [Can I fix the tree topology and relative branch lengths, while allowing the alignment to vary?](#)

### 13.2.4. [Can I fix the tree topology and absolute branch lengths in all data partitions, while allowing the alignment to vary?](#)

#### 13.2.1. Can I fix the alignment and ignore indel information, like MrBayes, BEAST, PhyloBayes and other MCMC programs?

Yes. Add `-Inone` or `--imodel=none` on the command line.

#### 13.2.2. Can I fix the tree topology, while allowing the alignment to vary?

Yes. Add `--disable=topology` `--tree=treefile` on the command line.

#### 13.2.3. Can I fix the tree topology and *relative* branch lengths, while allowing the alignment to vary?

Yes. Add `--disable=tree` `--tree=treefile` on the command line.

#### 13.2.4. Can I fix the tree topology and *absolute* branch lengths in *all data partitions*, while allowing the alignment to vary?

Yes. Add `--disable=tree` `--tree=treefile` `--scale=1` on the command line.

## 13.3. Run-time error messages

### 13.3.1. [I tried to use --smodel LG+Rates.Gamma\[6\] and I got an error message "bali-phy: No match." What gives?](#)

#### 13.3.1. I tried to use `--smodel LG+Rates.Gamma[6]` and I got an error message "bali-phy: No match." What gives?

You are probably using the C-shell as your command line shell. It is trying to interpret **LG+Rates.Gamma[6]** as an array before running the command, and it is not succeeding. Therefore, it doesn't even run **bali-phy**.

To avoid this, put quotes around the substitution model, like this: `--smodel "LG+Rates.Gamma[6]"`. This will keep the C-shell from interfering with your command.

## 13.4. Stopping bali-phy.

- 13.4.1. [Why is bali-phy still running? How long will it take?](#)
- 13.4.2. [How do I stop a bali-phy run on my personal computer?](#)
- 13.4.3. [How do I stop a bali-phy run on a computing cluster?](#)
- 13.4.4. [So, how can I know when to stop it?](#)
- 13.4.5. [How can I tell when the chain has converged?](#)
- 13.4.6. [How can I check how many iterations the chain has finished?](#)

### 13.4.1. Why is **bali-phy** still running? How long will it take?

It runs until you stop it. Stop it when its done.

### 13.4.2. How do I stop a **bali-phy** run on my personal computer?

Simply kill the process -- there is no special command to stop **bali-phy**. If you are running it on your personal workstation, then you can use the command **kill**. To do that, you need to find the PID (process ID) of the running program. You can find this by examining the beginning of the file `C1.out`. For example:

```
% less 5d-1/C1.out
command: bali-phy 5d.fasta
start time: Wed Aug  9 13:44:39 2017

VERSION: 3.0-beta2 [type-constraints commit b792c30b+] (Aug 04 2017 13:40:40)
BUILD: Aug  7 2017 21:01:44
ARCH: x86_64-pc-linux-gnu
COMPILER: GCC 7.1.0
FLAGS: -isystem $(top_srcdir)/boost/include -isystem $(top_srcdir)/include -ffast-math -DNDEBUG -
DNDEBUG_DP -funroll-loops -Wall -Wextra -Wno-sign-compare -Woverloaded-virtual -Wstrict-aliasing -pipe
-O3 -pedantic -I/usr/include/cairo -I/usr/include/glib-2.0 -I/usr/lib/x86_64-linux-gnu/glib-2.0/include
-I/usr/include/pixman-1 -I/usr/include/freetype2 -I/usr/include/libpng16 -g -Wmisleading-indentation -
Wlogical-op -fno-var-tracking-assignments
directory: /home/bredelings/Work
subdirectory: 5d-257
hostname: telomere
PID: 18838
...
```

Here the PID is 18838. Therefore you can type:

```
% kill 18838
```

On some operating systems you can also type:

```
% killall bali-phy
```

However, be aware that this will terminate *all* of your **bali-phy** runs on that computer.

### 13.4.3. How do I stop a **bali-phy** run on a computing cluster?

Simply terminate the submitted job. The specific command to terminate a job will depend on the queue manager that is installed on your cluster. Examine the documentation for your cluster, or ask your cluster support staff how to delete running jobs on your cluster.

As an example, if the SGE software is used to submit jobs, then the command **qstat** should list your jobs and their job ID numbers (which is different than the process ID number). You can then use the command **qdel** to delete jobs by ID number. The SGE documentation describes how to use these commands.

### 13.4.4. So, how can I know when to stop it?

You can stop when it has both converged and also run for long enough to give you >1000 effectively independent samples.

#### 13.4.5. How can I tell when the chain has converged?

See section [Section 10, "Convergence and Mixing: Is it done yet?"](#).

#### 13.4.6. How can I check how many iterations the chain has finished?

Run **wc -l C1.log** inside the output directory, and subtract 2.

### 13.5. Running bp-analyze.

13.5.1. [Why does bp-analyze say "Program 'draw-tree' not found. Tree pictures will not be generated"?](#)

13.5.2. [Why does bp-analyze say "Program 'gnuplot' not found. Trace plots will not be generated"?](#)

13.5.3. [Why does bp-analyze say "Program 'R' not found. Some mixing graphs will not be generated"?](#)

13.5.4. [Why is bp-analyze stopping early, or failing to generate some files?](#)

#### 13.5.1. Why does **bp-analyze** say "Program 'draw-tree' not found. Tree pictures will not be generated"?

The program **draw-tree** was not distributed on this platform (Windows, Mac). This is not a fatal error message, it just means that a pretty picture of the tree will not be generated automatically. You can still view the tree with *FigTree*, for example.

#### 13.5.2. Why does **bp-analyze** say "Program 'gnuplot' not found. Trace plots will not be generated"?

This is because you have not installed *gnuplot*. This is not a fatal error message, it just means that pictures of traces, partition support, and SRQ plots will not be generated automatically. You can still view MCMC traces with *Tracer*.

#### 13.5.3. Why does **bp-analyze** say "Program 'R' not found. Some mixing graphs will not be generated"?

This is because you have not installed *R*. This is not a fatal error message, it just means that a plot showing differences in clade probabilities between runs will not be generated.

#### 13.5.4. Why is **bp-analyze** stopping early, or failing to generate some files?

Look in the file `Results/bp-analyze.log`. This should contain the actual commands that were run, along with error message from these commands. These error message should give you a hint as to what the problem might be.

### 13.6. Interpreting the results.

13.6.1. [How do I compute the clade support?](#)

13.6.2. [How do I compute the split/bi-partition support?](#)

#### 13.6.1. How do I compute the clade support?

Actually, BALi-Phy uses unrooted trees, so it only estimates bi-partition support. A bi-partition is a division of taxa into two groups, but it does not specify which group contains the root.

#### 13.6.2. How do I compute the split/bi-partition support?

After you analyze the output ([Section 6.4, "Summarizing the output - scripted"](#)), the partition support is indicated in `Results/consensus` and in `Results/c50.PP.tree`.

### 13.7. How do I...

13.7.1. [How do I concatenate alignments?](#)

13.7.2. [How do I select columns from an alignment?](#)

13.7.3. [How do I create an alignment-constraint file from an alignment?](#)

### 13.7.1. How do I concatenate alignments?

```
% alignment-cat filename1.fasta filename2.fasta > result.fasta
```

The alignments must have the same sequence names, but the names need not be in the same order.

### 13.7.2. How do I select columns from an alignment?

```
% alignment-cat -c1-10,50-100,600- filename.fasta > result.fasta
```

The resulting alignment will contain the selected columns in the order you specified.

### 13.7.3. How do I create an alignment-constraint file from an alignment?

To constrain the alignment to match some alignment file *filename.fasta* in columns 100, 200-250, and 300, run:

```
% alignment-indices -c100,200-250,300 filename.fasta > filename.constraint
```