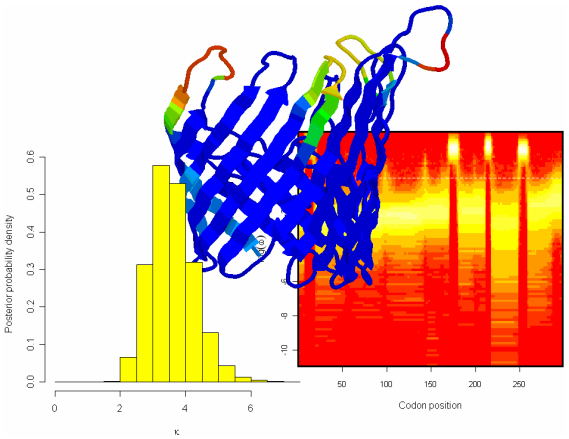
omegaMap version 0.5

DOCUMENTATION



Daniel J. Wilson

19th April 2006

Acknowledgements

Firstly, thanks to Gil McVean who is the co-author of the method implemented here in omegaMap[1]. I would like to thank my office-mates who have helped with programming in C++ and R[2]: Chris Spencer, Graham Coop and Adam Auton. omegaMap uses some functions from the PAML package[3], which is written by Ziheng Yang, who kindly provided the code. For offering useful ideas and statistical advice I would like to thank Stephen Leslie, Jonathan Marchini and Bob Griffiths. I am grateful to Rachel Urwin and Martin Maiden for their help and advice in applying omegaMap to meningococcal porin antigens, and Jeremy Derrick for sending me the molecular structures. Much of the simulation work required to test omegaMap was performed on a multinode AMD compute cluster that was bought with a grant awarded by the Wolfson Foundation to Peter Donnelly. Finally I would like to thank the Biotechnology and Biological Sciences Research Council, who funded this research.

1. The reference for omegaMap is D.J. Wilson and G. McVean (2006) *Genetics* doi: 10.1534/genetics.105.044917. Available from http://www.genetics.org.
2. R is available for free from http://www.r-project.org. R Development Core Team (2005). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria.
3. PAML is available for free from http://abacus.gene.ucl.ac.uk/software/paml.html. Z. Yang, (1997) PAML: a program package for phylogenetic analysis by maximum likelihood. *Comput. Appl. Biosci*. **13**: 555-556.

Table of Contents

1 Introduction

omegaMap is a program for detecting natural selection and recombination in DNA or RNA sequences. It is based on a model of population genetics and molecular evolution. The signature of natural selection is detected using the dN/dS ratio (which measures the relative excess of non-synonymous to synonymous polymorphism) and the signature of recombination is detected from the patterns of linkage disequilibrium. The model and the method of estimation are described in

Estimating diversifying selection and functional constraint
in the presence of recombination
Daniel J. Wilson and Gilean McVean
Genetics doi:10.1534/genetics.105.044917

This is an instruction manual for downloading, installing and running omegaMap. It is likely that you will want to read most of what is written if you want to use the program properly. Even so, reading a manual is nearly as tedious as writing one, so I've tried to set things out in a clear order and I've tried to emphasize important points.

The layout is designed to read as you go along, i.e. as you are installing the program, as you check it has installed correctly, and as you try out a first analysis. So hopefully you'll be able to read it as you go along, rather than sitting down and laboriously pouring over the entire manual.

There are several things you will need in addition to omegaMap. These are:

1. A text editing program such as Notepad or Emacs
2. A statistics package such as R or a spreadsheet such as Excel.
3. One or more fast computers that you can leave running for hours if not days.

And also an understanding of

1. The assumptions made by the model (see Key assumptions)
2. Bayesian inference and posterior distributions

2 Installing omegaMap

2.1 Download omegaMap

omegaMap is available for download from

www.danielwilson.me.uk

From there you have the choice of downloading ready-to-use executables for Windows and Mac, or downloading the source code which you can then compile manually on any platform with a compatible C++ compiler.

2.2 Ready-to-use executables

The easiest thing to do is to download the executable for your operating system (currently ready-to-use executables are available for several versions of Windows and Mac). The executables you need are:

1. omegaMap
2. decode
3. summarize

The programs `decode` and `summarize` are utilities for omegaMap, and are described in Encoded datafiles and Automatic interpretation using summarize respectively.

2.3 Compiling omegaMap manually

This might be preferable not only if you are using Linux or Unix, etc, but also because it allows the code to be optimized to your hardware which may improve performance by perhaps 10%.

1. Download the compressed source code from the website.
2. Unzip the compressed source code.
3. In the omegaMap directory, type

```
make
```

Didn't work? For this to work you'll need gcc installed and properly configured. If you have a different compiler you can edit `makefile` accordingly. If it doesn't work properly it's probably because

1. gcc or some of its libraries are not installed or configured correctly on your computer. You will need to speak to your computer support staff.
2. There are minor differences between your version of gcc and the version used to write omegaMap. In this case someone you know with knowledge of C++ might be able to fix the problem by making small changes to the source code or the makefiles.

If compiling omegaMap manually is too problematic you should consider downloading the ready-to-use executables.

3 Analysis

There are at least three steps to any analysis

1. Configure the model, which includes specifying the priors
2. Run omegaMap
3. Interpret the output

omegaMap offers additional functionality

1. Storing the output in an encoded format rather than a text file. This allows the whole information about the MCMC chain to be stored compactly. A text file can be extracted later using the program `decode`.
2. Simplifying the output of the MCMC chain using the program `summarize`, which automatically performs straightforward analyses of the data.
3. Finally, estimates of the dN/dS ratio can be superimposed on to the 3D structure of a molecule if you have the appropriate structural information and molecular rendering software.

For more details see the sections Encoded datafiles, Automatic interpretation using summarize and Molecular rendering.

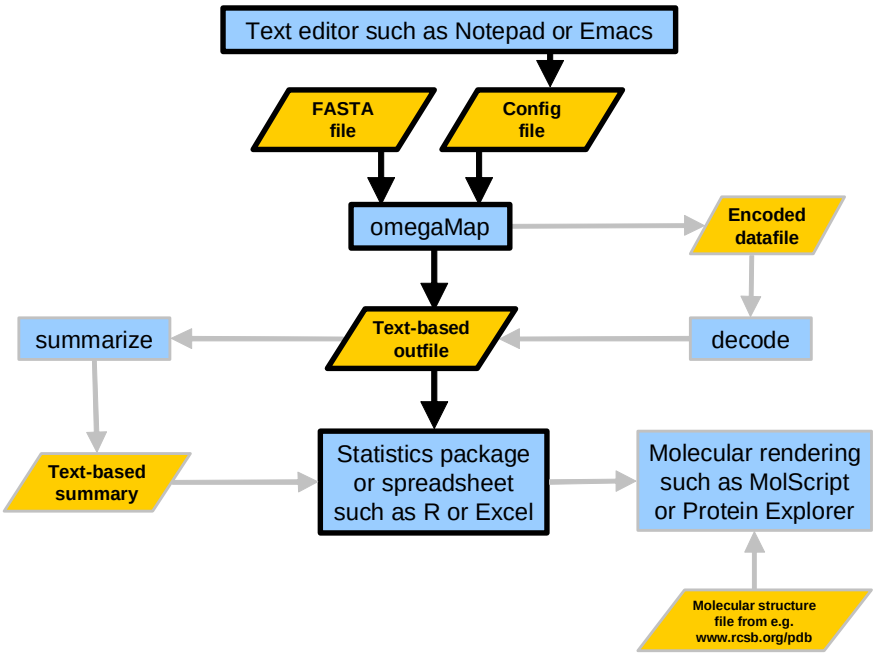


Figure 1 Flowchart of the analyses available using omegaMap.

As an example, we have 10 sequences each 1kb long in the file `genes.txt`. An analysis might run like this.

3.1 Configure the model

This is done by writing a configuration file, which you might call `genes.ini`.

3.1.1 General settings

First specify the filename for the FASTA file

```
FASTA = "/home/wilsondj/omegaMap/genes.txt"
```

Next specify the equilibrium frequencies of the 61 non-STOP codons. If this isn't specified the frequencies will be estimated empirically from `genes.txt`, which is a bad idea because some codons may not appear at all in your dataset. Better would be to refer to a whole genome sequence. The simplest approach is to assume all codons have equal frequency (i.e. frequency 1/61). This will probably suffice for a preliminary analysis, and can be revised later. So

```
pi=.016393,.016393,.016393,.016393,.016393,.016393,.016393,.016393,.0
16393,.016393,.016393,.016393,.016393,.016393,.016393,.016393,.016393
,.016393,.016393,.016393,.016393,.016393,.016393,.016393,.016393,.016
393,.016393,.016393,.016393,.016393,.016393,.016393,.016393,.016393,.
016393,.016393,.016393,.016393,.016393,.016393,.016393,.016393,.01639
3,.016393,.016393,.016393,.016393,.016393,.016393,.016393,.016393,.01
6393,.016393,.016393,.016393,.016393,.016393,.016393,.016393,.016393,
.016393
```

(The order of the codons in this list is assumed to be `TTT, TTC, TTA, TTG, TCT, ...`).

The number of orderings of the PAC likelihood must be chosen (see Wilson and McVean [2006] for details). In general, the more orderings the better, but the slower omegaMap will run. Ten orderings is suggested as a sensible number for a serious analysis. Only for a preliminary analysis, you might choose one ordering (this will make omegaMap run 10 times faster, but the results will not be reliable, meaning that replicate analyses undertaken with a different ordering will yield different results).

```
norders = 10
```

You can then specify the orderings using a list of numbers whose length equals the number of sequences ($n = 10$ in our case) multiplied by the number of orderings (*norders* = 10). The list consists of *norders* sequences of n numbers, each of which is one of the orderings. Each ordering should be a random permutation of the numbers $0, 1, \ldots, n-1$. An example is given below. The benefit of specifying the exact orderings is so that omegaMap can be run twice using the same orderings to make sure the results match. If they do, then the analysis has worked. To generate a list of random orderings use the program `order`, included with omegaMap. The syntax is

```
order sample-size number-of-orderings
```

```
orders = 0, 8, 5, 7, 6, 1, 3, 9, 4, 2, 4, 3, 1, 9, 6, 8, 7, 0, 2, 5,
8, 1, 5, 0, 7, 3, 2, 6, 9, 4, 5, 9, 0, 4, 1, 6, 7, 8, 2, 3, 9, 7, 8,
2, 5, 0, 4, 3, 6, 1, 1, 0, 5, 6, 7, 3, 8, 9, 2, 4, 5, 1, 3, 9, 0, 7,
6, 2, 8, 4, 8, 7, 1, 0, 4, 9, 3, 2, 5, 6, 7, 0, 6, 5, 4, 9, 8, 1, 3,
2, 3, 4, 5, 8, 0, 2, 9, 6, 7, 1
```

The number of iterations of the MCMC algorithm is specified using `niter`. For any dataset this number will be at least ~10 000 in magnitude, if not ~100 000 or even ~1 000 000. The choice of `niter` depends on the number of sequences, the length of

the sequences, the average length of an omega or rho block (specified using `oBlock` and `rBlock` below) and nuances of the data itself (in particular whether the model is a good fit to the data). Computer time might limit the number of iterations you can perform, but in general, `niter` must be large enough so that two runs of omegaMap with the same orderings give identical results (to within an acceptable margin of error). The best thing to do is to run a preliminary analysis to get an idea of (i) how long an iteration takes and (ii) how many iterations will be required. More info about assessing reliability of results is given later in Interpret the output.

```
niter = 250,000
```

The output file is a text file whose columns are the model parameters and whose rows correspond to particular iterations of the MCMC chain. Together, the rows constitute non-independent draws from the joint posterior distribution of the parameters. Specify the name of the output file using `outfile`.

```
outfile = "genes.out.txt"
```

One row of the output file is written every `thinning` iterations, so to output every hundredth iteration use `thinning = 100` or to output every iteration use `thinning = 1` (beware the amount of disk space required!)

```
thinning = 100
```

By default you cannot write to a file that already exists, unless you specify the following option

```
overwrite = true
```

More options can be found in Configuration file options.

3.1.2 Specifying the priors

There are two approaches to prior specification in Bayesian analysis: *objective* or *subjective*. The description here is meant as a guide only, and not an authoritative account. A *subjective* prior is one which represents the earnest prior beliefs of the researcher about the probable values of the parameters before performing the analysis. This can be achieved by careful consideration of what distribution describes your prior beliefs most closely. Many researchers prefer to try to specify an *objective* prior either because of the difficulty in carefully formulating a subjective prior or because they are uncomfortable with prejudicing the results of the analysis with their prior beliefs. There is disagreement over what exactly an objective prior is, some Bayesians think that there can be no truly objective prior, so instead objective priors are sometimes referred to as *reference*, *vague* or *non-informative* priors.

Currently, the distributions available in omegaMap are

1. Exponential
2. Exponential ratio*
3. Gamma
4. Improper inverse*
5. Improper uniform
6. Inverse*

7. Log normal*
8. Uniform

Details about the parameters of the distributions and how to set them are available in Configuration file options.

If the above distributions are to be used to specify subjective priors, then the choice will need justification when the results of the analysis are presented. All the parameters in omegaMap (μ, κ, ω, ρ and φ) are constrained to be positive. Therefore an objective prior will probably be one that is symmetric on the log scale. Distributions that are symmetric on the log scale are denoted with an asterisk above.

In the absence of any inspiration guiding you to a choice of prior, the following is a suggested reference prior:

- Improper inverse distributions on μ, κ, and φ.
- If using the constant or independent model for variation in ω and ρ along the sequence, use improper inverse distributions on these as well.
- Otherwise, use the inverse distribution and specify maximum and minimum values for ω and ρ.

```
muPrior = improper_inverse
kappaPrior = improper_inverse
indelPrior = improper_inverse
omegaPrior = inverse
omegaParam = 0.01, 100
rhoPrior = inverse
rhoParam = 0.01, 100
```

The inverse distribution corresponds to a uniform distribution on the log scale. The maximum and minimum values for uniform or inverse priors on ω and ρ need to be chosen carefully. They should be wide enough that they do not constrain the posterior (i.e. the posterior density outside the range would have been essentially zero even if a wider range had been used). However, making the range unnecessarily wide will inadvertently cause over-smoothing of the variation in ω or ρ along the sequence. If during the interpretation it appears that you set the range to be too narrow, you can repeat the analysis using a wider range.

Note that when using improper priors for any of the parameters (`improper_uniform` or `improper_inverse`), omegaMap can no longer automatically choose the initial values of those parameters in the MCMC chain. (For proper prior distributions, the initial values are by default drawn randomly from the prior.) The initial values must be set manually using the options `muStart`, etc (see Configuration file options). Initial values that have extremely low posterior probability can cause the MCMC to fail to converge. A sensible guess or a previous estimate should be used. As long as the number of iterations is sufficiently large, the initial values will not bias the results of omegaMap. It is good practice to use different initial values for different MCMC runs.

```
muStart = 0.1
kappaStart = 3.0
indelStart = 0.1
```

3.1.3 Modelling variation in ω and ρ

There are three models for variation in ω (and likewise ρ. The models do not need to be the same for both parameters. For the purpose of illustration, I'll refer simply to ω for the rest of this section.)

1. Constant
   All sites are assumed to share a common ω.
2. Variable
   A block-like model is used in which adjacent sites can share the same ω. When using the variable model, the average length of a block has to be specified. This is turn controls the strength of the block structure. Broadly speaking, the longer the average length of a block, the smoother the variation in ω will appear in the posterior.
3. Independent
   Each site is assumed to have its own ω independent of all other sites.

Computationally speaking, the constant model is by far the quickest. The independent model will take the longest for the MCMC to converge, because there are many more parameters. The variable model will take an intermediate amount of time, depending on the average block length (longer blocks lead to greater smoothing and faster convergence).

```
omega_model = variable
oBlock = 30
rho_model = variable
rBlock = 30
```

3.2 Run omegaMap

Having prepared the configuration file, `genes.ini` say, then from the command line type, in Windows

```
omegaMap genes.ini
```

or in Linux

```
./omegaMap genes.ini
```

By using additional flags after the configuration filename, it is possible to override options set in the configuration file. More details are contained in Configuration file options, but for example, to save the output to a different file use

```
omegaMap genes.ini –outfile different.txt              (in Windows)
./omegaMap genes.ini –outfile different.txt            (in Linux)
```

Overriding options in the configuration file might be useful, for example, when performing batches of analyses together, in which all but a few options stay the same.

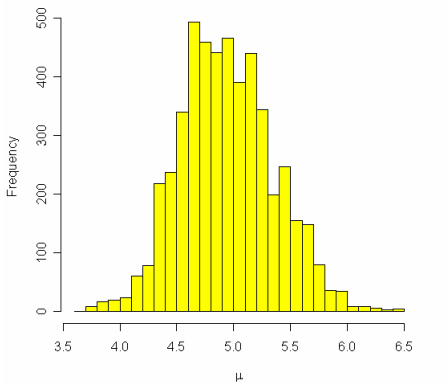
3.3 Interpret the output

omegaMap produces a text file (the name specified above by the `outfile` option), the columns of which correspond to the model parameters (i.e. μ, κ and φ, and ω and ρ for each codon plus the locations of the boundaries between blocks with common ω or ρ

parameters), as well as a few columns containing other information (e.g. the iteration number, the likelihood).

Statistical packages: Entries in the file are separated by tabs, so it can be read by many statistical packages or spreadsheets. However, for analysing proper datasets the file is likely to be very large. It may be too large for a program such as Excel. Something more powerful such as R or S Plus is recommended (R is available for free from www.r-project.org). If you are familiar with Excel you might want to use it to open the text output of exploratory analyses that have a small number of iterations, just to get a feel for the layout of the file. If you prefer to stick with Excel (or equivalent) for proper analyses, rather than go to the trouble of learning a new package (such as R), then the program `summarize` will produce a more manageable summary of the output. See section Automatic interpretation using summarize. If you would like to use R, then see the Sample R code section.

Each row of the text file corresponds to a particular iteration of the MCMC chain. Therefore, the entries of a particular column make up the estimate of the posterior distribution of that parameter. The posterior distribution for that parameter can be visualized by plotting a histogram of the entries of that column. For example, a histogram of the μ column gives an estimate of the posterior distribution of μ.

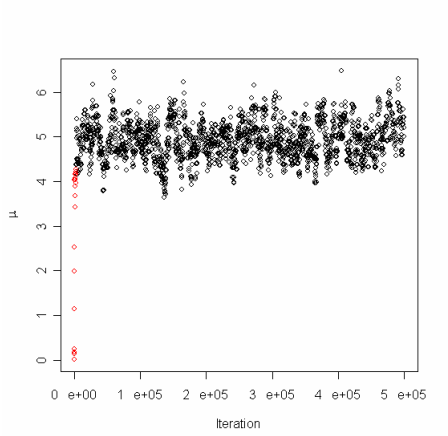


The mean and credible interval (the Bayesian equivalent of a confidence interval) can also be estimated from the entries of that column. In this case they would be 4.9 and (4.2, 5.8).

There are a couple of things to bear in mind

1. To improve the estimate of the posterior distribution, a burn-in is usually removed from the beginning of the chain. For example, the first 20 000 iterations of a 500 000 iteration chain might be discarded. The reason for removing the burn-in is because at the beginning the chain will be unduly influenced by the starting values, which are usually drawn from the prior.
2. Consecutive iterations in the MCMC chain are not independent of one another. This is a good reason for using a thinning interval of greater than 1. Depending on the strength of the autocorrelation (i.e. the strength of the non-independence between consecutive iterations), a large thinning interval (e.g. 100 or even 1 000) may be used without a great loss of information, thus saving on disk space.

The amount of burn-in to remove is usually chosen by plotting the *traces* of a handful of the parameters, and deciding how many iterations it takes before the chain is independent of the initial values. For example, the trace of μ below shows that the initial value of 0.14, which would have a very low posterior density, continues to affect the MCMC chain for the first 3 000 iterations (marked in red). So to improve the estimate of the posterior distribution of μ, a burn-in of 3 000 iterations should be removed.



The autocorrelation is also evident in this trace: adjacent points in the trace are likely to have similar values of μ, notwithstanding the fact that a thinning interval of 100 has been used. I.e. the autocorrelation is still apparent (even by visual inspection) at a lag (distance) of 100 iterations.

3.3.1 Automatic interpretation using `summarize`

The program `summarize` will read in the output file of omegaMap and automatically generate means and credible intervals for μ, κ and φ, and ω and ρ for each codon. The results can be opened in a spreadsheet, for example Excel, and plotted. By specifying multiple output files, the results of two or more independent runs can be merged.

The output from `summarize` is self-explanatory, and it should be straightforward to use the summary to produce figures in a familiar spreadsheet package. `summarize is run from the command line using the syntax`

```
summarize burnin filename1
```

or, when merging multiple runs (in this case three)

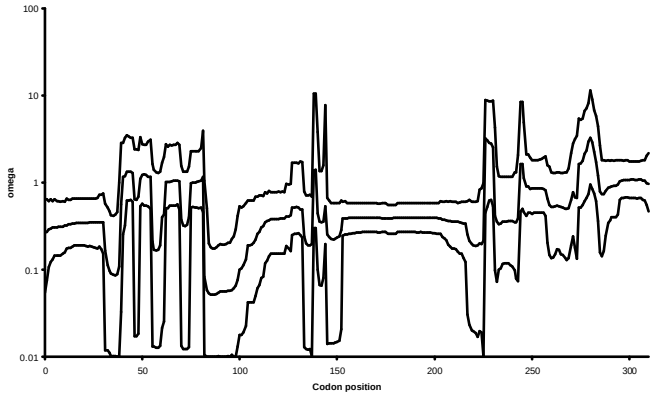
```
summarize burnin filename1 filename2 filename3
```

The program ought to take no more than a minute or so to run, maybe less.

The `summarize` program should not be treated as a black box, because it does not circumvent the problems of choosing the burn-in, or establishing whether the MCMC chain has converged. However, it can be used to diagnose convergence as follows

1. Take the output files of two or more runs of omegaMap that you hope have been allowed to run long enough to converge, e.g. 500 000 iterations. Suppose you have called them `genes.out1.txt` and `genes.out2.txt`.
2. Decide what a reasonable burn-in might be. If you really do believe that the chains have been allowed to run for long enough, then a burn-in of 10% of the total number of iterations might be appropriate.

3. Run `summarize` on each output file separately. E.g. (the > sends the output to a file)
```
./summarize 50000 genes.out1.txt > genes.summary1.txt
./summarize 50000 genes.out2.txt > genes.summary2.txt
```
4. Open `genes.summary1.txt` and `genes.summary2.txt` into a spreadsheet such as Excel. For every parameter, compare the mean and credible intervals for the two runs and determine whether they match to within an acceptable degree of error. For the ω and ρ parameters, you might want to plot the mean, upper and lower credible interval against the codon position to allow a visual comparison.
   For example, the figure below shows, for a single run, a plot of the mean and higher and lower 95% HPD bound for ω against codon position. The graph was produced in Excel. The y axis is on a logarithmic scale. By producing this plot separately for two independent runs, the two figures could be compared to see if the runs have converged.



5. If the chains have converged, the independent runs can be merged to obtain the final estimates of the mean and credible intervals for each parameter.

```
./summarize 50000 genes.out1.txt genes.out2.txt > genes.summary.txt
```

If the chains have failed to converge, they may simply need to be run for longer. However, a lack of convergence might also be caused by poor choice of priors, or severe model mis-specification, meaning that the data do not fit the model at all. In the latter cases, `summarize` may not be enough to diagnose the problems, and manual inspection of the output files might be necessary.

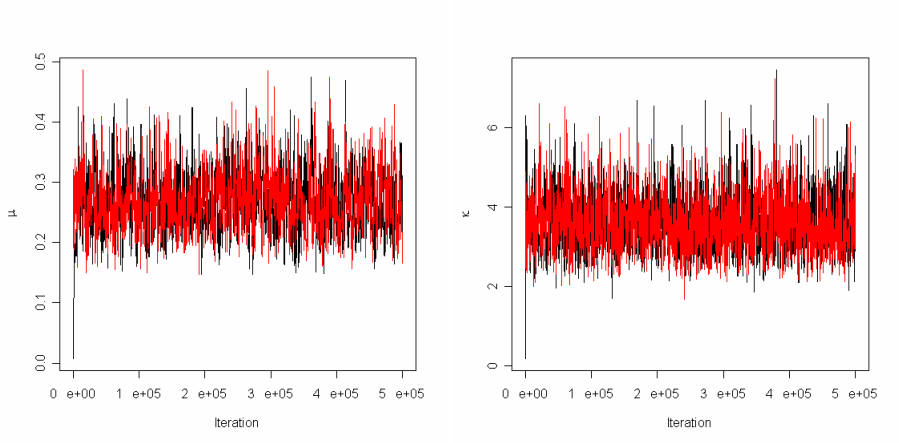
3.3.2 Manual interpretation of the output

The MCMC output files can be opened directly into a spreadsheet, allowing traces to be plotted, and summaries of the data not provided by `summarize`. Excel has an upper limit to the size of files it can open, so it might be necessary to use a more powerful program such as R (available free from www.r-project.org) or S Plus.

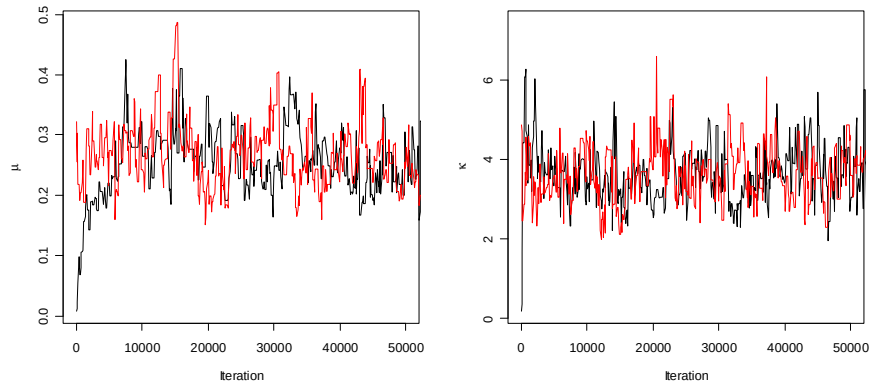
The following is an example analysis, using functions detailed in the section Sample R code, and performed in R. Code written in R should also work in S Plus. The session starts by copying and pasting the functions from the file `R-functions.txt`.

```
### Set the working directory (use forward slashes, even in Windows)
setwd("E:/Temp")

### Load the two independent MCMC runs.
```

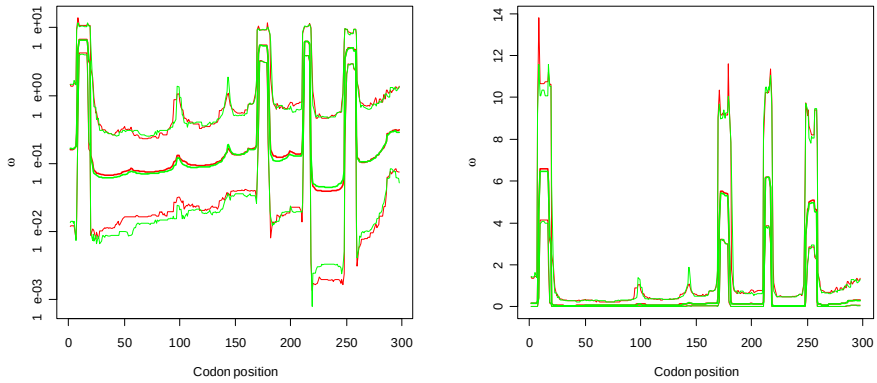
```
### The orderings were the same for both runs.
run1 <- open.omegaMap("o.carriage0.txt")
run2 <- open.omegaMap("o.carriage1.txt")

### Create a list of the runs
runs <- list(run1=run1,run2=run2)

### Look at the traces for a couple of parameters to assess
### what burn-in is required
trace.omegaMap(runs,"mu")
trace.omegaMap(runs,"kappa")
```



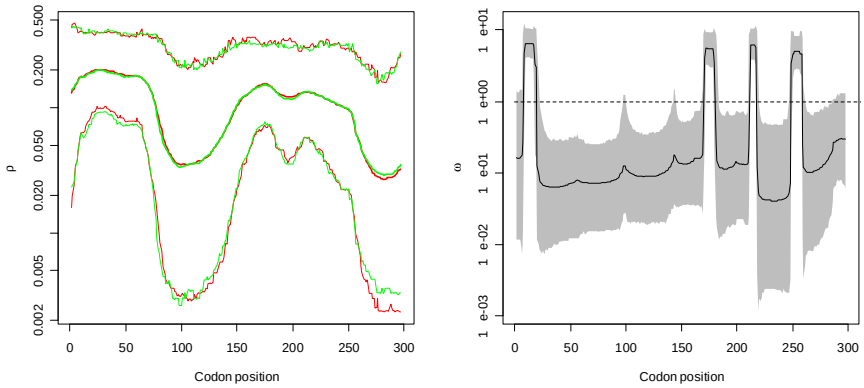

```
### The traces suggest a short burn-in is required.
### Use the xlim option to get a better view.
### Another useful option is ylim.
trace.omegaMap(runs,"mu",xlim=c(0,50000))
trace.omegaMap(runs,"kappa",xlim=c(0,50000))
```



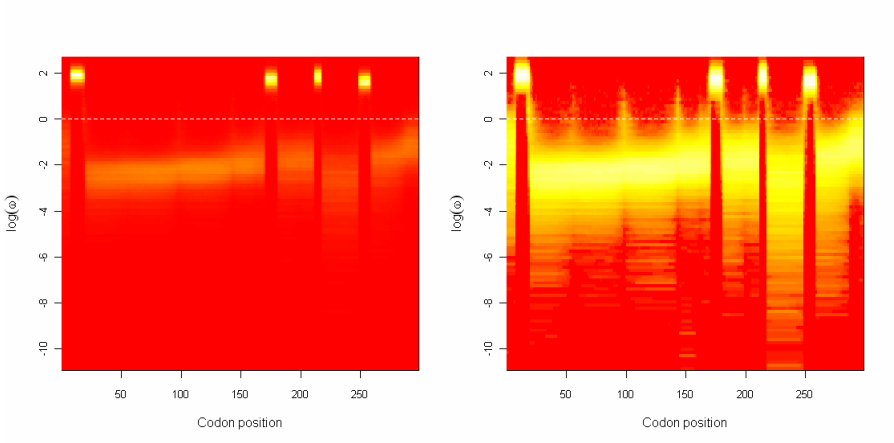

```
### Remove a short burn-in of 5000 iterations
run1b <- remove.burnin(run1,20000)
run2b <- remove.burnin(run2,20000)
runs <- list(run1=run1b,run2=run2b)

### Assess convergence of omega and rho
mycols <- c("red","green")            # Change the default colours
plot.omega.converged(runs,cols=mycols)
plot.omega.converged(runs,cols=mycols,log="") # Not on a log scale
```



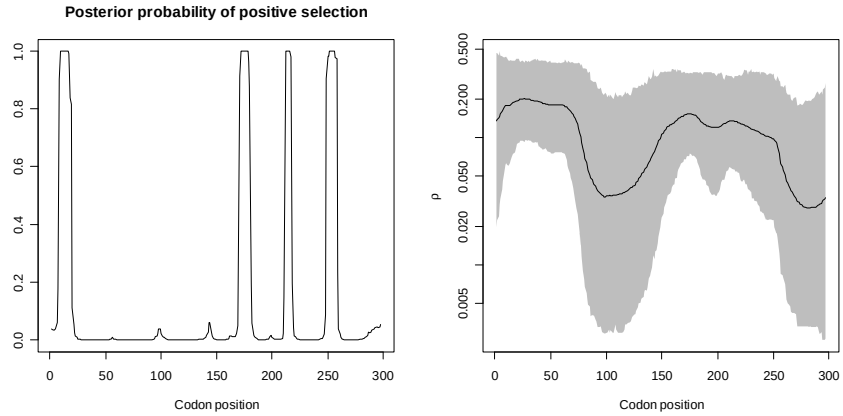

```
### The chains have converged for omega, so check rho
x11()                                # Create a new window
plot.rho.converged(runs,cols=mycols)
### The chains do appear to have converged, so continue the analysis.
### Plot the posterior distribution of omega across sites, for
### the two runs combined.
plot.omega(runs)
### Add a line at omega=1 to aid interpretation
lines(c(0,1000),c(1,1),lty="dashed")
```



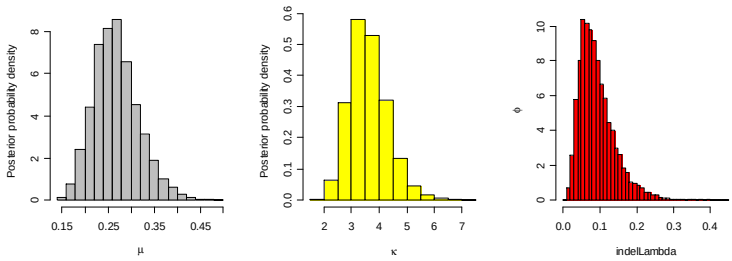

```
### The same information be alternatively represented by a fireplot.
fireplot.omega(runs)                 # This one can take a little while.
### Notice the difference in how to add a line at omega=1 here
lines(c(0,1000),log(c(1,1)),lty="dashed",col="white")
### The contrast between high, mid and low posterior densities
### is poor. So try changing the colfunc option.
fireplot.omega(runs,colfunc=function(x){log(x+.001)})
lines(c(0,1000),log(c(1,1)),lty="dashed",col="white")
```




```
### Plot the posterior probability of positive selection.
positively.selected.sites(runs)
### And the posterior on rho, for the runs combined.
plot.rho(runs)
```




```
### Can look at the histograms of the other parameters
names(run1)                          # Lists all parameters
par(mfcol=c(1,3))              # Three plots to a window
hist.omegaMap(runs,"mu")
### Set colour to yellow
hist.omegaMap(runs,"kappa",col="yellow")
### Set colour to red and number of bars to 200
hist.omegaMap(runs,"phi",col="red",breaks=200)
```




```
### Restore one plot to a window
par(mfcol=c(1,1))
### Obtain various summaries of the posteriors plotted above
### Effective sample sizes (ESS's) should be greater than 100.
### If not, the MCMC chains have not been run long enough.
```

```
point.estimate(runs,"mu")
hpd.omegaMap(runs,"mu")

effective.sample.size(runs,"mu")

point.estimate(runs,"kappa")
hpd.omegaMap(runs,"kappa")

effective.sample.size(runs,"kappa")

point.estimate(runs,"phi")
hpd.omegaMap(runs,"phi")

effective.sample.size(runs,"phi")
```

```
 [1] 0.2633618
Lower Bound Upper Bound
   0.182823    0.370466
 [1] 873.0125

 [1] 3.537345
Lower Bound Upper Bound
    2.48014     5.13964
 [1] 1692.624

[1] 0.08113412
Lower Bound Upper Bound
  0.0292725   0.2179990
 [1] 3787.016
```

4 Encoded datafiles

When the `outfile` option only is specified in the configuration file, then unless a thinning interval of 1 is used (which would result in a very large file), then it is impossible to reduce the thinning interval at a later stage (because the information was never saved). This is not usually a problem, but when diagnosing a problem with the MCMC chain it could be useful to view the chain at all iterations.

By specifying the `datafile` option, the whole MCMC chain is stored in an encoded format. From experience, the size of a `datafile` is roughly equal to the size of an `outfile` with a thinning interval of 100. The difference is that any thinning interval can subsequently be extracted from the `datafile` using the `decode` program.

So for example, instead of (or as well as) specifying `outfile` in the configuration file (call it genes.ini, say), `datafile` is specified.

```
datafile = "genes.data.txt"
```

Before any analysis can be performed on genes.data.txt, it must be decoded as follows. For example, with a thinning interval of 69.

```
decode genes.data.txt genes.decoded.txt 69
```

For independent runs, the `datafiles` are decoded separately.

The file `genes.decoded.txt` is then equivalent to an `outfile`, and can be manually interpreted as explained in Manual interpretation of the output, or summarised using the `summarize` program. For example,

```
summarize genes.decoded.txt > genes.summary.txt
```

or

```
summarize genes.decoded1.txt genes.decoded2.txt > genes.summary.txt
```

5 Sample R code

The file `R-functions.txt` contains R code for reading in the text output from omegaMap, and generating some simple plots. The functions are

```
open.omegaMap()
trace.omegaMap()
remove.burnin()
plot.omega.converged()
plot.rho.converged()
plot.omega()
plot.rho()
fireplot.omegaMap()
fireplot.rho()
positively.selected.sites()
hist.omegaMap()
point.estimate()
hpd.omegaMap()
effective.sample.size()
```

To use the code, simply copy and paste all the functions in to an R session. The file `R-example.txt` contains an example of how the functions might be used, and gives an indication of some other useful built-in functions in R. What follows is an explanation of the usage of the functions.

Some of these functions borrow code from the R packages boa and coda. The relevant code was lifted straight from these packages and copied into the omegaMap functions to make life as easy as possible for people wishing to use R to interpret output from omegaMap. So apologies to the following authors, part of whose R code I have stolen: Brian J. Smith, Kate Cowles, Nicky Best, Karen Vines and Martyn Plummer. The packages boa and coda are available in their entirety from www.r-project.org.

`open.omegaMap(filename, burnin=0, ...)`

`filename`	Name of the file to open, in quotes
`burnin`	Number of iterations of burn-in to remove from the beginning. Default is zero.
`...`	Further i/o options. Type `?read.table` for more information

Opens an MCMC outfile for omegaMap. Note that this function can take a long time to run, depending on the size of the file being opened.

```
Examples    run1 <- open.omegaMap("spinosa.outa.txt")
            run2 <- open.omegaMap("spinosa.outb.txt")
```

`trace.omegaMap(runs,param,cols=palette(),...)`

`runs`	Names of the objects containing the MCMC runs, in the form `list(run1)` for a single run, or `list(run1,run2)` for multiple runs.
`param`	Name of the parameter to plot in quotes. For a list of available parameters type `names(run)`.
`cols`	Colours to use to distinguish the traces of the different runs. Default is to use the built-in palette.
`...`	Further plotting options. Type `?plot` or `?par` for more information.

Plot the trace of a parameter against iteration number for one or more MCMC runs.

Examples
```
trace.omegaMap(list(run1,run2),"mu")
trace.omegaMap(list(run1,run2),"mu",cols=c("red","green")
)
trace.omegaMap(list(run1,run2),"kappa",ylim=c(0,20))
trace.omegaMap(list(run1,run2),"phi",type="p")
```

remove.burnin(run, burnin)

run — Name of an object containing a single MCMC run, *not* of the form list(run).

burnin — The number of iterations to remove from the start of the MCMC chain.

Removes a burn-in from the beginning of the MCMC chain, in preparation for further analysis.

Examples
```
run1 <- remove.burnin(run1,20000)
run2 <- remove.burnin(run2,20000)
```

plot.omega.converged(runs,log="y",cols=palette(),width=c(1,2,1))

runs — Names of the objects containing the MCMC runs, in the form list(run1) for a single run, or list(run1,run2) for multiple runs.

log — Whether the x and y axes are on a log scale. The default is "y", indicating that only the y axis is to be plotted on the log scale. The alternatives are "", "x" and "xy".

cols — Colours to use to distinguish the plots of the different runs. Default is to use the built-in palette.

with — Width of the lines used to plot each run, of the form width=c(i,j,k) where (i,j,k) correspond to (lower credible bound, mean, higher credible bound) respectively. The default is for the mean to be plotted thicker.

Plots the mean and credible intervals for omega along the gene, separately for each run. The runs can then be compared to see if they have converged to within an acceptable degree of tolerance.

Examples
```
plot.omega.converged(list(run1,run2))
plot.omega.converged(list(run1,run2),log="")
plot.omega.converged(list(run1,run2),width=c(1,4,1))
```

plot.rho.converged(runs,log="y",cols=palette(),width=c(1,2,1))

runs — Names of the objects containing the MCMC runs, in the form list(run1) for a single run, or list(run1,run2) for multiple runs.

log — Whether the x and y axes are on a log scale. The default is "y", indicating that only the y axis is to be plotted on the log scale. The alternatives are "", "x" and "xy".

cols — Colours to use to distinguish the plots of the different runs. Default is to use the built-in palette.

with — Width of the lines used to plot each run, of the form width=c(i,j,k) where (i,j,k) correspond to (lower credible bound, mean, higher credible bound) respectively. The default is for the mean to be plotted thicker.

Plots the mean and credible intervals for rho along the gene, separately for each run. The runs can then be compared to see if they have converged to within an acceptable degree of tolerance.

Examples `plot.rho.converged(list(run1,run2))`
`plot.rho.converged(list(run1,run2),log="")`
`plot.rho.converged(list(run1,run2),width=c(1,4,1))`

plot.omega(runs,log="y",...)

runs — Names of the objects containing the MCMC runs, in the form list(run1) for a single run, or list(run1,run2) for multiple runs.

log — Whether the x and y axes are on a log scale. The default is "y", indicating that only the y axis is to be plotted on the log scale. The alternatives are "", "x" and "xy".

... — Further plotting options. Type ?plot or ?par for more information.

Plots the mean and credible intervals for omega along the gene, combining the runs.

Examples `plot.omega(list(run1,run2))`
`plot.omega(list(run1,run2),log="")`
`lines(c(0,1000),c(1,1),lty="dashed") # draws line at 1`

plot.rho(runs,log="y",...)

runs — Names of the objects containing the MCMC runs, in the form list(run1) for a single run, or list(run1,run2) for multiple runs.

log — Whether the x and y axes are on a log scale. The default is "y", indicating that only the y axis is to be plotted on the log scale. The alternatives are "", "x" and "xy".

... — Further plotting options. Type ?plot or ?par for more information.

Plots the mean and credible intervals for rho along the gene, combining the runs.

Example `plot.rho(list(run1,run2))`

fireplot.omega(runs,log="y",yres=100,colfunc=function(x){x}, cols=heat.colors(100))

runs — Names of the objects containing the MCMC runs, in the form list(run1) for a single run, or list(run1,run2) for multiple runs.

log — Whether the y axis only is on a log scale. The default is "y", indicating that only the y axis is to be plotted on the log scale. The alternative is "".

yres — Number of equal-width bins to split log(omega) or omega into. Defaults to 100.

colfunc — A function used to vary the colour contrast of the histogram. See below for more details. Defaults to function(x){x}.

cols — The palette from which to draw the colours. Defaults to the heat colour palette, with 100 gradations. Type ?heat.colors for more information.

Produces a fireplot of log(omega) or omega. A fireplot visualizes the posterior on log(omega) or omega along the sequence using a colour gradient. By default higher posterior density is represented by more intense colour (closer to white), and lower posterior density is

represented by less intense colour (closer to red). The palette used can be changed using the option cols. For a smoother gradation of colours try cols=heat.colors(500). The contrast between high, mid and low density colours can be changed using the colfunc option. By default a linear function is used, but a useful alternative is colfunc=function(x){log(x+.001)}. Varying .001 can produce a wide range of colour contrasts.

Examples
```
fireplot.omega(list(run1,run2))
fireplot.omega(list(run1,run2),colfunc=function(x){log(x+
.001)})
fireplot.omega(list(run1,run2),cols=rainbow(100))
```

fireplot.rho(runs,log="y",yres=100,colfunc=function(x){x}, cols=heat.colors(100))

runs — Names of the objects containing the MCMC runs, in the form list(run1) for a single run, or list(run1,run2) for multiple runs.

log — Whether the y axis only is on a log scale. The default is "y", indicating that only the y axis is to be plotted on the log scale. The alternative is "".

yres — Number of equal-width bins to split log(rho) or rho into. Defaults to 100.

colfunc — A function used to vary the colour contrast of the histogram. See below for more details. Defaults to function(x){x}.

cols — The palette from which to draw the colours. Defaults to the heat colour palette, with 100 gradations. Type ?heat.colors for more information.

Produces a fireplot of log(rho) or rho. A fireplot visualizes the posterior on log(rho) or rho along the sequence using a colour gradient. By default higher posterior density is represented by more intense colour (closer to white), and lower posterior density is represented by less intense colour (closer to red). The palette used can be changed using the option cols. For a smoother gradation of colours try cols=heat.colors(500). The contrast between high, mid and low density colours can be changed using the colfunc option. By default a linear function is used, but a useful alternative is colfunc=function(x){log(x+.001)}. Varying .001 can produce a wide range of colour contrasts.

Examples
```
fireplot.rho(list(run1,run2))
fireplot.rho(list(run1,run2),colfunc=function(x){log(x+.0
01)})
fireplot.rho(list(run1,run2),cols=rainbow(100))
```

positively.selected.sites(runs,...)

runs — Names of the objects containing the MCMC runs, in the form list(run1) for a single run, or list(run1,run2) for multiple runs.

... — Further plotting options. Type ?plot or ?par for more information.

Plots the posterior probability of positive selection (i.e. omega>1) for each site along the gene. The runs are combined.

Examples
```
positively.selected.sites(list(run1,run2))
# For a thicker line:
positively.selected.sites(list(run1,run2),lwd=3)
```

hist.omegaMap(runs,param,col="grey",freq=F,...)

runs	Names of the objects containing the MCMC runs, in the form list(run1) for a single run, or list(run1,run2) for multiple runs.
param	Name of the parameter to plot in quotes. For a list of available parameters type names(run).
col	Colour of the histogram. Default is "grey".
freq	Whether the y axis should be probability density or count. The default is probability density.
...	Further plotting options. For more information type ?hist or ?par.

Plots the posterior probability density for param as a histogram. The runs are combined. The likely values of param will be "mu", "kappa" and "phi". Marginal posterior densities for omega and rho at individual sites can be obtained by using, for example, "omega0", "rho115". The names of all possible parameters can be viewed by typing, for example, names(run1).

Example
```
hist.omegaMap(list(run1,run2),"mu")
hist.omegaMap(list(run1,run2),"mu",col="yellow")
```

point.estimate(runs,param,log=TRUE)
hpd.omegaMap(runs,param,log=TRUE,alpha=0.05)
effective.sample.size(runs,param)

runs	Names of the objects containing the MCMC runs, in the form list(run1) for a single run, or list(run1,run2) for multiple runs.
param	Name of the parameter to plot in quotes. For a list of available parameters type names(run).
log	Specifies whether param is converted to a log scale before the point estimate or HPD interval is calculated. If so, the results are reported back on the original scale. Defaults to TRUE. Alternative is FALSE.
alpha	Specifies the width of the highest posterior density credible interval. The 100(1-alpha)% HPD interval is returned. By default, alpha=0.05.

These functions produce summaries of the posterior probability density for the specified parameter. In particular, point.estimate() returns the mean of the posterior distribution for param, hpd.omegaMap() returns the 100(1-alpha)% HPD credible interval and effective.sample.size() returns the effective number of independent data points for the MCMC chains combined.

Note that point.estimate() and hpd.omegaMap() by default operate on the logarithmic scale, then convert back to the original scale. The reason is that the parameters of omegaMap (mu, kappa, omega, rho and phi) are most naturally interpreted on a logarithmic scale. This feature can be disabled by specifying log=FALSE.

Examples
```
point.estimate(list(run1,run2),"mu")
point.estimate(list(run1,run2),"mu",log=FALSE)
hpd.omegaMap(list(run1,run2),"mu")
# Obtain a 99% HPD interval:
hpd.omegaMap(list(run1,run2),"mu",alpha=0.01)
effective.sample.size(list(run1,run2),"mu")
```

6 Molecular rendering

It is possible to use the output of omegaMap to colour three-dimensional representations of proteins encoded by structural genes. omegaMap doesn't provide any details about the molecular structure – for that a pdb file is required. Find out whether the structure for your protein exists from the database at www.rcsb.org/pdb/.

To render the image you will need additional software such as
Protein Explorer: http://molvis.sdsc.edu/protexpl/frntdoor.htm
MolScript : http://www.avatar.se/molscript/

For help using these programs you will need to consult their documentation or online tutorials. The RCSB protein data bank also has an online tutorial.

The example that follows is designed to give you a few pointers. Firstly, decide how to colour the protein. The two obvious choices are

1. Using the point estimate of ω.
2. Using the posterior probability of positive selection.

Either of these can be obtained using the `summarize` program (see Automatic interpretation using summarize).

Secondly, you need to align your FASTA sequence with the protein sequence in the pdb file. This can be done by hand; unfortunately it is a bit tedious. One reason you need to align them is because the pdb file will have a single amino acid sequence, whereas non-synonymous polymorphism will cause the sequences in your FASTA file to encode multiple amino acid sequences. Another reason is that indels that may exist in your sequences will have been removed from the pdb file. There is also a good chance that your FASTA sequences and the pdb amino acid sequence will begin and end at different codons.

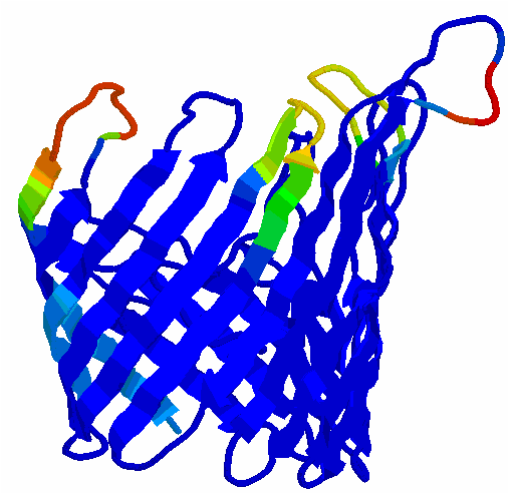
Thirdly, you need to edit the pdb file to enter the colouring. An example of the first dozen rows of a pdb file is shown below. In the pdb file, rows beginning ATOM have a 'temperature Factor' column, and this is where the omegaMap colour can be entered. According to the most recent file format specifications, available from
http://www.rcsb.org/pdb/file_formats/pdb/pdbguide2.2/guide2.2_frame.html
http://www.rcsb.org/pdb/static.do?p=file_formats/pdb/index.html
the temperature factor is situated between the 61st and 66th characters on rows beginning ATOM. In the pdb example below, the column starts with the number `0.05`. You need to manually go through this file changing the temperature factors to the colouring.

One difficulty with editing the pdb file is that columns are delimited by a fixed number of digits (this constrains the number of decimal places you can use), so they can be easily read into many spreadsheets, but e.g. Excel cannot save in this format. There are probably better ways to edit pdb files, but I used a text editor. The next difficulty is that you will want to colour every atom belonging to a particular amino acid the same colour. Fortunately the fourth (beginning ASP in the example) and fifth (beginning 1) columns in the pdb file record the amino acid residue that the atom belongs to.

```
REMARK Produced by MODELLER: 11-Apr-02 14:28:25                                 1
REMARK MODELLER OBJECTIVE FUNCTION:      2361.4534
ATOM      1  N   ASP     1      56.806  19.282  19.867  1.00  0.05       1SG   2
ATOM      2  CA  ASP     1      55.872  20.429  19.823  1.00  0.05       1SG   3
ATOM      3  CB  ASP     1      54.637  20.095  18.963  1.00  0.05       1SG   4
ATOM      4  CG  ASP     1      53.876  18.928  19.588  1.00  0.05       1SG   5
ATOM      5  OD1 ASP     1      54.373  18.345  20.588  1.00  0.05       1SG   6
ATOM      6  OD2 ASP     1      52.776  18.606  19.063  1.00  0.05       1SG   7
ATOM      7  C   ASP     1      55.418  20.806  21.191  1.00  0.05       1SG   8
ATOM      8  O   ASP     1      55.639  20.073  22.154  1.00  0.05       1SG   9
ATOM      9  N   VAL     2      54.784  21.986  21.310  1.00  0.06       1SG  10
ATOM     10  CA  VAL     2      54.311  22.412  22.590  1.00  0.06       1SG  11
```

Having edited the pdb file, and having taken care not to corrupt it in the process(!) all that remains is to render the molecule and make sure that it is coloured according to the temperature factor. Protein Explorer, which is available for free (see URL above), provides user friendly clickable molecular rendering which allows you to rotate the protein and use various representations of the protein such as balls and sticks and ribbons. By right-clicking in the protein window it is simple to change how the molecular is coloured, and the options include, in additional to the temperature factor, colouring it from the N to the C terminus, according to amino acid type, and others.

In Protein Explorer, once you have rotated the protein to your preferred view and coloured it appropriately, you simply copy the image and paste it into a graphics package. So what you see is what you get – which can be a bit grainy compared to others such as MolScript which can produce smoother, higher-resolution images. The following image was produced in Protein Explorer using the point estimate of omega to colour the protein, and the ribbon view. Because the FASTA file analysed with omegaMap contained sequences shorter than the amino acid sequence in the pdb file, I assigned a value of $\omega = 1$ for the unanalysed sites at the N and C termini (coloured light blue in the figure).



7 Key assumptions of the model

omegaMap uses an approximation to an explicit population genetics model, the key assumptions of which are outlined here.

1) The haplotypes constitute a random sample of the population
   a) As a result, it is important not to include each unique haplotype only once in the FASTA file, unless it was observed only once in the sample: if a particular haplotype was found *x* times there should be *x* copies of it in the FASTA file. Sequencing techniques that discard or distort information about the frequency of haplotypes are therefore problematic.
   b) To reduce the size of big datasets, haplotypes should be removed at random and with probability proportional to the frequency of the haplotype. Reducing the dataset size by representing each unique haplotype exactly once will violate the assumption of random sampling.
2) The population has a constant size
3) Mating (or horizontal gene transfer) in the population is random
   a) A consequence of this assumption is that for diploids, both chromosomes can be sampled without violating the first assumption.
   b) Deep structure in the population is problematic. In particular, sequences from different species that cannot conceivably mate at random should not be analysed together.
4) All individuals in the population have the same reproductive potential
5) The population size is large
   a) The sample size is supposed to be much smaller than the effective population size. Sample sizes that approach the effective population size will violate this assumption.

Violating the above assumptions will have unknown effects. Further research may show that results differ in their sensitivity to the violation of different assumptions. Extending the model could allow certain violations of the current assumptions to be treated properly, such as the analysis of haplotypes from populations with deep structure, or incorporating known demographic history.

8 Configuration file options

These options are not case-sensitive. Those marked *Required* must always be included in the initialization file or at the command line. Those marked *Optional* do not have to be, and those marked *Required** must be included in some situations. Any option specified in the initialization file can be overridden at the command line, by prefixing the option with a hyphen and using the following examples of syntax.

omegaMap genes.ini
omegaMap genes.ini –outfile different.txt
omegaMap genes.ini –muprior uniform –muparam 0 20

In addition to overriding options specified in the initialization file, the command line can be used to specify options that aren't specified at all in the initialization file.

Reading in the sequence data

`FASTA`
Required
Filename (and path) of the FASTA file containing the RNA/DNA sequence data

Fixed variables for the model

WARNING: while `pi` can be estimated from the data, this is not recommended. Should any codons not be represented in the dataset, that codon is excluded from the mutation model.

`pi`
Optional
List of the equilibrium frequencies of the 61 codon positions. Should sum to 1. Default is to estimate these from the data.

`piIndel`
Optional
Equilibrium frequency of indels relative to codons at sites segregating for indels. Must be between 0 and 1. Default is to estimate these from the data.

`norders`
Required
Number of orderings in the PAC likelihood.

`orders`
Optional
List of the orderings, of length L*norders (where L is the sequence length in codons). If omitted then chosen at random.

Output options

WARNING: one of `outfile` or `datafile` must be specified.

`niter`
Required
Number of iterations to run the MCMC.

`outfile`
Filename for text file to output the variables in a tab-separated

Optional — value file at the chosen `thinning` interval. Default is no output.

`thinning` *Required** — Thinning interval for the text file. Must be a positive integer. Required if `outfile` is specified.

`datafile` *Optional* — Filename for data file containing the entire encoded MCMC output that can be reanalyzed later. Default is no output.

`coutput` *Optional* — `true` or `false`
If true then a counter is sent to the standard output (usually the screen). Set to false to turn the counter off, desirable for example if the standard output is piped to a file. Defaults to `true`.

`overwrite` *Optional* — `true` or `false`
If `true` then `outfile` and/or `datafile` will be overwritten even if they already exist. Defaults to `false`.

Specifying the priors

For each of the parameters mu, kappa, omega, rho and indel, the prior distribution and the parameters for that distribution must be specified. For example, the prior for mu is specified by…

`muPrior` *Required* — `fixed`, `uniform`, `improper_uniform`, `exponential`, `gamma` or `exponential_ratio`.
Distribution to use for the prior on mu.

`muParam` *Required** — Parameter(s) for the prior distribution on mu. Required unless the distribution takes no parameters.

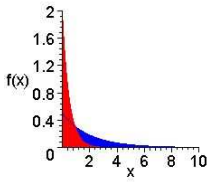
The available distributions and their parameterisations are detailed here.

`fixed` — Parameters: the value to fix the variable at. For mu and kappa this should be a single value. For omega it should be a single value if `omega_model` is `constant`, or a list of length L if `omega_model` is `independent`. For rho it should be a single value if `rho_model` is `constant`, or a list of length L–1 if `rho_model` is `independent`.
WARNING: specifying a `fixed` prior, for mu for instance, overrides the `muStart` option.

`exponential` — Parameter: the rate λ

$$f(x) = \lambda \exp(-\lambda x) \text{ for } x \geq 0, \lambda > 0$$

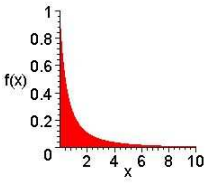
which has mean $1/\lambda$ and variance $1/\lambda^2$.



For example, $\lambda = 2$ (red), $\lambda = \frac{1}{2}$ (blue).

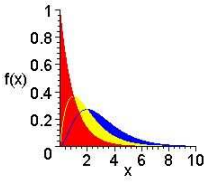
`exponential_ratio` — No parameters.

$$f(x) = \frac{1}{(1+x)^2} \text{ for } x \geq 0$$



which has its median at $x = 1$, but undefined mean and variance. This distribution is appropriate for modelling the ratio of two exponential variates with the same rate.

`gamma`



Parameters: the shape parameter *a*, the scale parameter *b*

$$f(x) = \frac{x^{a-1}\exp(-x/b)}{\Gamma(a)b^a} \text{ for } x \geq 0, a > 0, b > 0$$

which has mean ab and variance ab^2.

For example, $a = 1$ (red), $a = 2$ (yellow), $a = 3$ (blue). $b = 1$ in all.

`improper_inverse`

No parameters.

$$f(x) \propto 1/x \text{ for } x > 0$$

This is an improper distribution because there is no constant of proportionality that can satisfy the above.
WARNING: reversible-jump MCMC requires proper priors, so `improper_inverse` cannot be used for ω when `omega_model` is set to `variable` nor for ρ when `rho_model` is set to `variable`.

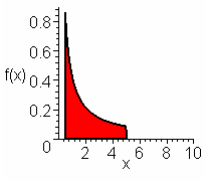
`improper_uniform`

No parameters.

$$f(x) \propto 1 \text{ for } x \geq 0$$

This is an improper distribution because there is no constant of proportionality that can satisfy the above.
WARNING: reversible-jump MCMC requires proper priors, so `improper_uniform` cannot be used for ω when `omega_model` is set to `variable` nor for ρ when `rho_model` is set to `variable`.

`inverse`



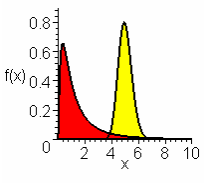
Parameters: the range *a*, *b*

$$f(x) = \frac{1}{x(\ln b - \ln a)} \text{ for } a \leq x \leq b$$

Note that *a* and *b* must be positive (greater than zero). The inverse distribution is a uniform distribution on the logarithmic scale.

For example, $a = 0.5, b = 5$.

`log_normal`



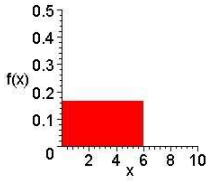
Parameters: the mean *m* and standard deviation σ of the transformed variate $y = \ln x$.

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}x}\exp\left(\frac{-(x-m)^2}{2\sigma^2}\right)$$

Use this distribution for parameters for which, on the logarithmic scale, you wish to fit a normal prior distribution. The distribution has mean $\exp(m + \sigma^2/2)$ and variance $\exp(2\sigma^2 + 2m) - \exp(\sigma^2 + 2m)$.

For example, $a = \ln 1 = 0.0, b = \ln 2.7 = 1.0$ (red),
$a = \ln 5 = -0.693, b = \ln 1.1 = 0.1$ (yellow).

`uniform`



Parameters: the range *a*, *b*

$$f(x) = \begin{cases} \dfrac{1}{b-a} & \text{if } a < x < b \\ 0 & \text{otherwise} \end{cases}$$

For example, $a = 0, b = 6$.

For both omega and rho, you must specify how the parameter varies along the gene sequence. If the variable model is used, the parameters for the blocks must be specified.
WARNING: the shorter `oBlock` and `rBlock`, the longer the MCMC will take to converge. The extreme case of this is for `independent` parameters for each site.

`omega_model`
Required

`constant` or `variable` or `independent`
Specifies whether a single omega is to be estimated for the entire sequence, or a variable omega.

`rho_model`
Required

`constant` or `variable` or `independent`
Specifies whether a single rho is to be estimated for the entire sequence, or a variable rho.

`oBlock`
*Required**

Mean length (in codons) of an omega block. Must be greater than 1. Required if `variable` omega_model is specified.

`rBlock`
*Required**

Mean length (in codons) of a rho block. Must be greater than 1. Required if `variable` rho_model is specified.

Controlling the MCMC chain

`seed`
Optional

A seed for the random number generator. If omitted then chosen using the system clock. Must be a negative integer.

These commands allow the initial state of the MCMC to be controlled. If omitted, the initial values for the parameters are drawn from their prior distributions. For improper priors, it is necessary to manually specify the initial values using these options.
WARNING: these options are overridden by specifying `fixed` in the prior.

`muStart`
*Required**

Sets the intial value of mu. Correspondly for kappa and indel.

`omegaStart`
*Required**

Sets the intial value of omega. If `omega_model` is `constant` then provide a single value. If `omega_model` is `independent` then provide a list of length L.

`rhoStart`
*Required**

Sets the intial value of rho. If `rho_model` is `constant` then provide a single value. If `rho_model` is `independent` then provide a list of length L– 1.

The following commands provide the relative weight of each MCMC move. All or none should be specified, and they should sum to one. If not they will be automatically re-normalised.
WARNING: convergence of the MCMC chain is sensitive to these commands.

`change_oBlock`
Default 0.1

Change value of omega for the sequence (`constant` model), a single block (`variable` model) or a single site (`independent` model).

`change_rBlock`
Default 0.1

Change value of rho for the sequence (`constant` model), a single block (`variable` model) or a single site (`independent` model).

`change_mu`
Default 0.0667

Change value of mu for the sequence.

`change_kappa`
Default 0.0667

Change value of kappa for the sequence.

`change_indel`
Default 0.0667

Change value of indel for the sequence.

`extend_oBlock`
Default 0.1

Extend an omega block 5′ or 3′

`splitMerge_oBlock`
Default 0.1

Split/merge omega blocks

`extend_rBlock`
Default 0.1

Extend a rho block 5′ or 3′

`splitMerge_rBlock`
Default 0.1

Split/merge rho blocks