Maximum Likelihood and Bayesian Practicals

Peter G. Foster
Natural History Museum, London

Rio de Janeiro, March 2008

For these practicals you will need the directory of data files, called `PeterPracticals`, which should be in your home directory. It has several subdirectories, one for each of the exercises. Do them in turn. You will usually need to go into, that is `cd`, to those directories. The data sets are small due to time constraints. Read and try to understand both the files that are given and the files that are produced during the analyses. The first few practicals are for review and orientation, and the rest deal with ML and Bayesian methods.

```
1_paup_pars_boot
2_repair_tree
3_add_node
4_choose_model
5_modeltest
6_paup_search
7_phyml
8_paup_search_sh_test
9_p4_tree-hetero_ml
10_mrbayes_tutorial
11_mrbayes
12_p4_tree-hetero_mcmc
13_twoPartitions
14_asdoss
```

Start up a terminal window. The current working directory in the shell should be your home directory. To start, say

```
cd PeterPracticals
```

and then issue the `ls` command at the terminal to see what is there. To get to the right place to do the first practical,

```
cd 1_paup_pars_boot
```

Tab completion.
Instead of typing out these long directory and file names in full, you can take advantage of the "tab completion" feature of the `bash` shell. Instead of typing the entire name (*eg.* `1_paup_pars_boot`) you can type just the beginning (*eg.* `1_`) and then hit the `<tab>` key – if the beginning part that you have typed is unambiguous, then the shell will finish typing the file or directory name for you (or program name, if what you typed is the first item in the command). This feature of course does not work inside interactive programs such as PAUP – it only works in the shell (although it will work in MrBayes if you have built it with the `readline` library).

1 Parsimony Bootstrap in PAUP

The practical in `1_paup_pars_boot` shows how to do a parsimony bootstrap analysis in `paup`. Look at the files in this directory. Read the `commands.nex` file, which is a series of commands to give to `paup` in the form of a NEXUS file. It has comments to explain the various steps.

Execute the file. You can run it by saying, to your command line:

```
paup commands.nex
```

The example data file contains only 12 taxa and will run quickly. An alignment of 100 taxa might take a few hours to run. At the end of the analysis, you will have a tree in a Nexus file, where internal nodes are decorated with bootstrap support values. The program will also make a log file as requested. Read it, and try to make sense out of it.

You can then import the resulting tree file into a tree-drawing program. If you want to import it into Treeview X, the file needs to be modified slightly – the zero just before the final semi-colon in the tree description needs to be removed. In this analysis, branch lengths are meaningless and so a "cladogram" format, where all of the leaf nodes line up on the right, is appropriate. Treeview X can place the bootstrap support values on the internal nodes. You can print it to a file for further manipulation. (A copy of an example output is in the file `bootTreeM.pdf` in the `Treeview` directory within `1_paup_pars_boot`.)

2 Editing a tree file

To do this practical, go, *ie* `cd`, to the directory `PeterPracticals/2_repair_tree`. Assuming that you are currently in the `1_paup_pars_boot` directory, you might want to do the following two steps. First, `cd` towards the root.

```
cd ..
```

and then possibly remind yourself of the contents with an `ls` command. Then `cd` to the correct directory by

```
cd 2_repair_tree
```

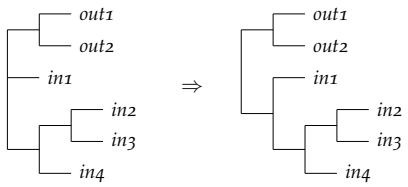
That directory contains a file called `t.nex`, a Nexus file containing one tree. It has a mistake in the Newick tree description. Find the mistake and correct it using a text

editor. You can read the file with paup, and if it is read successfully you can draw it with the `showtrees` command. You can also draw the tree to the terminal screen using p4 with the command `p4 -d t.nex` (note the spaces). When you have successfully repaired the file so that it can be read by paup, draw it using paup.

(The program p4 is a phylogenetics toolkit that also implements tree-heterogeneous models that we will use below. It is written in the Python language, and can use the Python interactive shell. You can get out of the interactive p4 with `control-d`.)

3 Adding a bifurcating root node

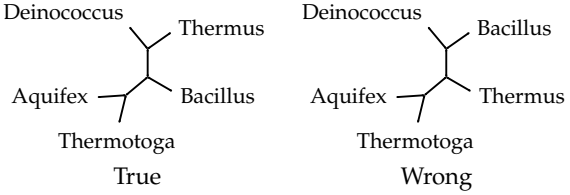
The files for this practical are in the directory `PeterPracticals/3_add_node`, so you need to go there, *ie* `cd` there. In that directory there are two tree files, both with the same tree—`t1.nex` without branch lengths, and `t2.nex` with branch lengths. In the first, add a node to make a bifurcating root node, as shown below. Make sure it is readable, and draw it with p4. (It is a little awkward to draw it with its bifurcating root with PAUP. How would you do that?)



Do the same with the tree in `t2.nex`, but take branch lengths into account. Divide the length of the branch that goes between the outgroup and the ingroup evenly between the two forks of the bifurcation. (Note that `p4 -d` distorts branch lengths, making shorter branch lengths longer in the picture, for clarity — this is a feature, not a bug.)

4 Choosing a model with the AIC

In this practical we will use PAUP to choose a model for an alignment of bacterial 16S genes. The data are found in a file `PeterPracticals/bacterial_16S.nex`.



The strategy that will be used is to evaluate the likelihood of the neighbor-joining tree with a series of different models. Then we will calculate the AIC for each model, which will tell us the best model.

We will be making much use of 2 commands in PAUP –

`lset`	to set the likelihood model
`lscore`	to get the optimized likelihood of a single tree

Recall that commands have options, and we can get the current settings of those options, and other possible settings of those options, with the built-in help in PAUP.

```
lset ?
lscore ?
```

For example,

```
paup> lset ?

Usage: LSet [options...] ;

Available options:

Keyword ---- Option type -------------- Current default setting
NST          1|2|6                      2
TRatio       <real-value>|Estimate|Previous   2
RMatrix      (<rAC><rAG><rAT><rCG><rCT>)|
             Estimate|Previous          (1 1 1 1 1)
RClass       (<cAC><cAG><cAT><cCG><cCT><cGT>) (a b c d e f)
Variant      HKY|F84                    HKY
BaseFreq     Empirical|Equal|Estimate|
             Previous|(<frqA><frqC><frqG>)    Empirical
Rates        Equal|Gamma|SiteSpec       Equal
Shape        <real-value>|Estimate|Previous   0.5
NCat         <integer-value>            4
RepRate      Mean|Median                Mean
PInvar       <real-value>|Estimate|Previous   0
SiteRates    Partition[:<charpartition-name>]|
             Rateset[:<rateset-name>]|Previous <none>
Wts          RepeatCnt|Ignore           RepeatCnt
InitBrLen    Rogers|LS|<real-value>     Rogers
LCollapse    No|Yes                     Yes
MaxPass      <integer-value>            20
Delta        <real-value>               1e-06
UseApprox    No|Yes                     Yes
ApproxLim    <real-value>               5
AdjustAppLim No|Yes                     Yes
LogIter      No|Yes                     No
ZeroLenTest  No|Full|Crude              No
Recon        Marginal|Joint             Marginal
AllProbs     No|Yes                     No
Clock        No|Yes                     No
UserBrLens   No|Yes                     No
MinMemReq    No|Yes                     No
StartVals    ParsApprox|Arbitrary       ParsApprox
ParamClock   Standard|Rambaut|BrLens|
             SplitTimes|MDRambaut|Thorne      Standard
MLDistforLS  No|Yes                     No
ShowQMatrix  No|Yes                     No

paup> lscore ?

Usage: LScores [tree-list] [/ options...] ;

Score-output options:

Keyword ---- Option type --- Current default setting --
SiteLikes    No|Yes           No
CategLikes   No|Yes           No
StoreAsWts   No|Yes           No
ParsApprox   No|Yes          *No
<...more...>
```

The overall strategy is to

- Make a NJ tree
- Evaluate the likelihood of that tree with the JC, F81, HKY, and GTR models
- Evaluate the likelihood of that tree with different among-site rate variation models with the chosen rate matrix, in this case the GTR model
- Use the likelihoods to calculate the AIC and choose the best model

Fill in this table as you go.

	$\ln L$	n	$-\ln L + n$	$-2\ln L + 2n$
JC	-4671.72506	0		
F81		3		
HKY85		4		
GTR		8		
GTR+I		9		
GTR+G		9		
GTR+IG		10		

1. Go to the `PeterPracticals` directory (this time, not to a subdirectory). Start the PAUP program and read in the data. Say

   ```
   paup bacterial_16S.nex;
   ```

   Then make a neighbor-joining tree. This will be the tree that we evaluate over and over with different models. When you first start up PAUP the default optimality criterion is parsimony, so we need to reset it to likelihood.

   ```
   nj
   set criterion=likelihood
   ```

2. Evaluate the likelihood under the Jukes-Cantor model

   ```
   lset nst=1 basefreq=equal
   lscore 1
   ```

   This sets the number of substitution types to 1, and the composition to 25% each nucleotide. This is the JC model. Among-site rate variation is left at the default values, which is none. Then the NJ tree is evaluated and branch lengths optimized under this model. The log likelihood is -4671.72506, and is our first entry in the table above.
3. Is the composition not equal?

   ```
   lset basefreq=estimate
   lscore 1
   ```

   Here `nst` "carries over" from the previous setting, and remains `nst=1`. This is the F81 model (Felsenstein, 1981), the same as the JC model except that the composition parameters are allowed to be free. There are 4 nucleotides, so there are 3 parameters and 3 degrees of freedom. Make a note of the log likelihood and put it in the AIC table.
4. Is there a transition/transversion ratio bias?

   ```
   lset nst=2 tratio=estimate
   lscore 1
   ```

   This is the HKY85 (Hasegawa, Kishino, Yano, 1985) model. Now `nst=2`, with one additional parameter. Note the log likelihood in the AIC table, for this and the other models tested below.
5. Now try the GTR model.

   ```
   lset nst=6 rmatrix=estimate
   lscore 1
   ```

   The GTR (`nst=6`) model has 5 more parameters than the F81 model, and 4 more than the HKY85 model. The `basefreq` remains at `estimate`. It is the most parameter-rich rate matrix that PAUP has to offer. This setting has no ASRV, but below we superimpose ASRV on the GTR model.
6. Among-site rate variation I. This is the GTR+I model.

   ```
   lset pinvar=estimate
   lscore 1
   ```

   Allowing a proportion of sites to be invariant (`pinvar=estimate`) adds one free parameter.
7. Among-site rate variation II. This is the GTR+G model. We turn pInvar off.

   ```
   lset pinvar=0.0 rates=gamma shape=estimate
   lscore 1
   ```

   Allowing Γ-distributed rates adds one free parameter over the GTR model.
8. Among-site rate variation: III. This is the GTR+GI model. We turn pInvar back on.

   ```
   lset pinvar=estimate
   lscore 1
   ```

9. Quit PAUP.

Having collected the likelihoods you can calculate the AIC values in the table. Choose the model with the lowest AIC. Is it the model that gives the highest likelihood?

Now we will do a similar series of models, but using a file with paup commands. Go to the `PeterPracticals/4_choose_model` directory and read the file `commands.nex`
This file has the following contents – try to make sense out of each command:

```
#nexus

begin paup;
    execute ../bacterial_16S.nex;
    nj;
    set crit=like;
    set warntsave=no;
    log start file=paupLog replace;
    lset basefreq=estimate;
    lset nst=2 tratio=estimate;
    lscore 1;
    lset nst=6 rmatrix=estimate;
    lscore 1;
    lset rates=gamma shape=estimate;
    lscore 1;
    lset rates=gamma shape=estimate
         pinvar=estimate;
    lscore 1;
    log stop;
    quit;
end;
```

Execute it by saying, to your command line,

```
paup commands.nex
```

What models were tested? Look at the log file that was produced.

Using a commands file containing a paup block of commands makes running a series of commands easier, when you know what you are doing. One example of this is the commands file used by ModelTest, in the next section.

5 ModelTest and SS ASRV

The program ModelTest can help remove some of the tedium of choosing a model. It uses PAUP to evaluate many different models using the PAUP block `modelblockPAUPb10`. Read it and try to make sense out of it. The results of the likelihood evaluations are fed to the program ModelTest, which assesses the likelihood values and suggests a model. So it is a two step process—first generating the likelihood scores using PAUP, and second assessing those scores with the ModelTest program proper.

We will use a simulated dataset with 7 taxa and 900 sites, in a file `simcdn900.nex`. It has a sets block at the end, which can be used by PAUP and other Nexus compliant programs (but not MrBayes) to partition the data.

1. Start PAUP and read in the data. Say

   ```
   paup simcdn900.nex
   ```

2. Execute the modelblock. Say

   ```
   execute modelblockPAUPb10
   ```

   `modelblockPAUPb10` makes a NJ tree with JC distances, and then evaluates that tree with several different models. This will take a couple of minutes. Don't quit yet, as we will need the tree again later.
   PAUP makes 2 files:

   a) `model.scores`, a concise output from PAUP
   b) `modelfit.log`, a verbose output from PAUP

3. Process the file `model.scores` with the program ModelTest, on the ModelTest server website. There is a link to it in the `sites.html` file. Note the chosen model and likelihood as suggested by the AIC.
4. ModelTest does not consider site-specific among-site rate variation (SS ASRV), so you need to do that "by hand". But first, redundantly, we repeat the evaluation with the HKY+G model, which was the best that ModelTest found.

   ```
   lset nst=2 rates=gamma shape=estimate
   lset basefreq=estimate tratio=estimate
   lscores 1;
   ```

   Do you get the same likelihood? It may differ slightly due to numerical round-off error.
5. Now we do the site-specific model. PAUP has read in the partition definition in the data file, and now we ask PAUP to apply it. Do the `lset` command all on one line.

   ```
   lset nst=2 rates=sitespec
        siterates=partition:by_codon
   lscores 1;
   ```

   Does the HKY+SS model have a higher likelihood than the HKY+G model? To assess whether it is better using the AIC, you may need to know that in data partitioned into n parts, and the parts are allowed their own overall rates (as is the case here), then there are $n - 1$ free parameters due to the site-specific among site rate variation. Don't forget that in this evaluation the gamma distributed ASRV is turned off. Is the HKY+SS model better using the AIC?

6 An heuristic search

See the file `commands.nex` in `6_paup_search`. It uses the bacterial 16S data, with the GTR+G model. The output from ModelTest was simply pasted in for the hsearch. (Usually we would need to use a "successive approximation" search, but this search is so easy that we only need to do it once. However, we do check that further searches are not needed.)

```
#nexus

begin paup;
  execute ../bacterial_16S.nex;
  log start file=paupLog replace;
  set crit=like;
  set autoclose=yes;
  Lset  Base=(0.2287 0.2621 0.3426)  Nst=6
        Rmat=(0.5147 1.3958 0.9436 1.4810 3.3455)
        Rates=gamma  Shape=0.3731  Pinvar=0;
  hsearch swap=tbr;
  lset basefreq=est rmatrix=est shape=est;
  lscore 1;
  savetrees file=bestTree.nex format=altnex
            brlens=yes replace;
  log stop;
  quit;
end;
```

Execute the file with `paup`.

In this set of commands, we search for the best tree with the `hsearch` command. We start with the model and parameters values suggested by ModelTest, and search with those parameters. After the search, we make the parameters free again (`xxx=estimate`) and optimize on the resulting tree. We do this because we want to reassure ourselves that we have made the (last) search based on fully optimized parameters. If so, the `hsearch` tree and the final tree should have the same likelihood, although they might differ by a small amount due to numerical round-off error. Is this so? Is the tree that is obtained biologically reasonable? If not, why not? How could you analyze these data that might give you the correct tree?

7 Tree search with phyml

Here we will use the bacterial 16S data again, and do a search with `phyml`. The data file, `bacterial_16S.nex` is in Nexus format, but `phyml` needs a modified phylip format. (The `phyml` format can handle taxon names up to 50 characters, but cannot handle spaces in names. It requires a space between the taxon name and the sequence, which the original phylip format does not.) Do the format conversion to the newest phylip format using the readseq web server, available as a link in the `PeterPracticals/sites.html` file, which you can use with a web browser. Save the sequences to a file in the directory `7_phyml`.

Go into that directory and start `phyml`. Read in the fasta data file and set up the GTR+G model (appropriate for these data), with estimated composition, and estimated gamma shape parameter. The program is fast enough that we can do a bootstrap analysis– do 200. Look at the output files.

8 Successive approximation search and SH test

In this practical, we look at 3 new strategies —

- Successive approximation search
- Topology-constrained search
- Shimodaira-Hasegawa test

This practical uses a different, simulated, dataset. Using it, we demonstrate *successive approximation* in the hsearch strategy. This strategy is usually needed for searches using PAUP, but is not needed in `phyml` as it is built-in to the `phyml` search strategy. It is important for speed that we do not search and optimize parameters at the same time. In the successive approximation strategy, we alternate between fixing parameter values and searching, and freeing parameter values and re-optimizing on the tree resulting from the search. This is continued long enough that we know that the last search used fully optimized parameters.

This exercise also demonstrates searching with a topological constraint. We search with and without a constraint for monophyly of a group of taxa, and obtain 2 different trees. We then ask if those trees are significantly different by the SH test.

I have already chosen a model to use; it is the TVMef+I model, chosen by the AIC in ModelTest. Read and execute `commands.nex`.

When we do the heuristic search for the ML tree, we notice that all the 'A' taxa (A1, A2, and so on) are not in the same split. It is not possible that the A-taxa are monophyletic in the ML tree. But let us say that we expected them to group together, and we expected the 'B' taxa to be all together in their own split. So we are a little surprised by our ML tree. We ask whether we can really reject monophyly of the A-taxa – perhaps it is only due to noisy, indecisive data? To find out, we find the best tree where the A-taxa are constrained to be in the same group, and ask, using the Shimodaira-Hasegawa test, whether that constrained tree is significantly worse than the ML tree.

So after finding the ML tree, we start over again, but this time with a topological constraint

```
constraints monoA = ((A1, A2, A3, A4, A5, A6, A7, A8));
```

When we do the hsearch with this constraint, we find a best tree, which is of course slightly less likely than the ML tree. Is this difference significant?

We assess significance using the SH test. We read in both trees (having saved them before) and ask for the `lscore` under the current model, and also we ask for an `shtest`:

```
gettrees file=ml_tree.nex;
gettrees file=constrained_tree.nex mode=7;
lsc 1-2/ shtest=rell;
```

Above, in the `gettrees` command, the default mode is 3, which replaces trees in memory with the trees from the file. So here the first `gettrees` command wipes out any trees in memory. Of course we do not want to do that when we do the second `gettrees` command, so we use mode 7, which adds the trees in the file to the trees in memory. At the end of the second `gettrees` command we have two trees in memory.

In the completed SH test, the P value that we get tells us that the constrained tree is not significantly worse than the ML tree, and so we do not have enough evidence, using this test and using these data, to reject monophyly of the A-taxa. However, note that the SH test depends on the number of trees compared and does not work well if there are only two trees; it should compare more plausible trees.

When the PAUP job is finished, look at the log file, and try to relate the contents of the log file with the commands in the file `commands.nex`. Were all of the steps of the successive iteration necessary?

9 ML with a heterogeneous model using p4

Here we re-analyse the bacterial 16S data with a heterogeneous model that allows a separate composition for the thermophiles, and a separate composition for the mesophiles. We only compare 2 trees, the true tree and the attract tree. Python commands determine which branches get which composition.

To analyse the data, say

```
p4 s.py
```

(For a description of the program, type in `p4` with no arguments. Quit with `ctrl-d`.)

The `s.py` Python script reads in the data, and the two trees. It then defines the model and places the two different composition vectors on the tree. Then, for each tree, it calculates the maximum likelihood of the tree under the model. How the results are written out can be specified by the user; here we draw the two trees, and write out their ML values. Is the ML value under the heterogeneous model better than the ML value under the GTR+G model? How many more parameters does this heterogeneous model have relative to the GTR+G model? Is is a better model by the AIC? Did use of a heterogeneous model allow recovery of the true tree as the ML tree?

10 MrBayes wiki tutorial

The tutorial is available in the MrBayes wiki manual, which is available as a link in the `sites.html` file. The tutorial section is also included as a standalone `Tutorial.html` file in `10_mrbayes_tutorial`. Do the tutorial using the file `primates.nex`, also in that directory. The executable for `MrBayes v3.1.2` is called `mb`. You can start the program and run it interactively by typing that command at the terminal. You can stop it with the `quit` command, or use `control-c`.

11 Bayesian analysis with MrBayes

You can run MrBayes interactively as in the tutorial, but here we will use a command file, in Nexus format, with a MrBayes block containing all the commands. At the moment, it contains the following.

```
begin mrbayes;
    execute ../bacterial_16S.nex;
    log start filename=mbout.log replace;
    set autoclose=yes;
    lset foo=bar;
    mcmc ngen=10000 printfreq=500 samplefreq=50
         nchains=4 savebrlens=yes filename=mbout;
    sump filename=mbout  burnin=101;
    sumt filename=mbout  burnin=101;
    log stop;
end;
```

It is correct and complete except that the line specifying the model is wrong. Recall that we want the GTR+G model for the bacterial data. Set it correctly, on the `lset` line. Hint: you can, using interactive MrBayes, issue the `help lset` command to show you how to do that. You can quit MrBayes with the `quit` command.

When you have corrected the model and saved the file, start the program by saying

```
mb commands.nex
```

Here is what the commands tell MrBayes to do —

- We ask that the chain run for 10000 generations, sampling every 50, for a total of 200+1 samples. We run 4 parallel chains in the MCMCMC. There are 2 separate concurrent runs, by default.
- We start with a random tree topology by default.
- At the end of the MCMC, after a burnin of 101 samples (not generations), the parameters are summarized using the sump command. Also, the trees in both the tree files are summarized, making a consensus tree `mbout.con` and a list of tree bipartitions in `mbout.parts`. Note that high confidence is given to the wrong tree.

The authors of MrBayes recommend that proposal acceptance rates be between 10 and 70%. Are they? I have noticed when I run these data that the chain swap acceptance is a little high sometimes. Is that the case? How would you lower the acceptance rate? (Hint: `help mcmc`)

How would you tell if the MCMC has converged?

12 Bayesian analysis with p4

To analyse the bacterial 16S data using a heterogeneous model, say

```
p4 s.py
```

Read the Python script to try to make some sense of it. Here is what happens — First the script tells p4 to read in the data. A random tree is made, and then the model is defined. To start, the two compositions are assigned to nodes randomly. A Bayesian MCMC run is set up, and some adjustments are made to the "tunings". A run of 10000 generations is done, and a consensus tree made from the sampled trees. Although the two compositions were assigned to branches randomly, the branches are allowed to choose either composition as part of the MCMC, allowing the two compositions to change what nodes they are assigned to in the tree while simultaneously adjusting their parameter values. What consensus tree is found? Is it biologically correct? How well supported is it?

13 More than one data partition with MrBayes

Here we analyse combined DNA and protein sequences. Look at the data file, `two_datatypes.nex`. MrBayes requires combining the 2 datasets into 1 alignment, and uses the Nexus incompatible

```
datatype=mixed(dna:1-200,protein:201-300)
```

1. Start MrBayes and read in the data file by

   ```
   execute two_datatypes.nex
   ```

2. Set up a character partition, and tell MrBayes to use it.

   ```
   charset one = 1-200
   charset two = 201-300
   partition p1 = 2:one,two
   set partition = p1
   ```

3. Let the two partitions have their own relative rates. The weighted mean rate will be 1.

   ```
   prset ratepr=variable
   ```

4. Set the model for the DNA, the first partition

   ```
   lset applyto=(1) nst=6 rates=equal
   ```

5. Set up the model for the protein partition, which is partition 2. This is done partly by setting the priors, but the ASRV is set via `lset`. We tell MrBayes to use the empirical composition of the data. (Do the `prset` all on one line)

   ```
   prset applyto=(2) aamodelpr=fixed(wag)
         statefreqpr=fixed(empirical)
   lset applyto=(2) rates=propinv
   ```

6. Check the model settings.

   ```
   showmodel
   ```

   In the previous step you told MrBayes to use empirical composition for the protein partition, and it did not complain. Did it make that setting?

7. Do the mcmc, collecting 201 samples. Do this command all on one line.

   ```
   mcmc ngen=10000 samplefreq=50 printfreq=1000
        nchains=1 filename=mbout savebrlens=yes
   ```

8. Digest the results, with a burnin of half the samples.

   ```
   sump filename=mbout burnin=101
   sumt filename=mbout burnin=101
   ```

How much faster is the overall rate of the DNA partition relative to the protein partition? Were the proposal acceptance rates good?

14 ASDOSS between two MCMC runs

In this practical you will calculate an ASDOSS point.

1. Start MrBayes interactively and read in the data, `star.nex`

   ```
   mb -i star.nex
   ```

2. Set the model

   ```
   lset nst=6
   ```

3. Do a very short MCMC. It is too short to converge well.

   ```
   mcmc ngen=1000 samplefreq=5
   ```

4. Quit. Note the files that were made. Two runs were done, and a tree file was made for each run.
5. Restart the program and read in the data file again.
6. Digest the tree file for the first run, with a burnin of half the samples.

   ```
   sumt nruns=1 filename=star.nex.run1 burnin=101
   ```

   Note the table of partitions.
7. Digest the tree file for the second run, again with a burnin of half the samples.

   ```
   sumt nruns=1 filename=star.nex.run2 burnin=101
   ```

   Again note the table of partitions.
8. Quit the program.
9. Make a table with 4 columns, where the first column is the partition, in dot-star notation, the second column is the support (probability) for that partition in the first run, and the third column is the support for that (same!) partition in the second run. There is no need to consider partitions where there is only 1 dot or 1 star. It may be that one of the two runs has a partition in the list but the other does not – in that case the run without the partition listed has a support of 0.0 for that partition.
10. Plot the support for run2 against the support for run1 for each partition. Do the points approximate a line on top of the $y = x$ line?
11. Calculate the standard deviation for each partition, putting the values in the fourth column in your table.
12. Calculate the average standard deviation of the split supports.

The average standard deviation of split supports (or split frequencies) can be used as a convergence diagnostic. In a pair of runs that are similar the ASDOSS will be low, and when that happens it is assumed or hoped that the runs have both converged to the posterior distribution.