Assessing Phylogenetic Reliability Using RevBayes and P^3

Model adequacy testing using posterior prediction

Lyndon M. Coghill, Sebastian Höhna and Jeremy M. Brown

1 Overview

This tutorial presents the general principles of assessing the reliability of a phylogenetic inference through the use of posterior predictive simulation (PPS). PPS works by assessing the fit of an evolutionary model to a given dataset, and analyzing several test statistics using a traditional goodness-of-fit framework to help explain where a model may be the most inadequate.

1.1 Requirements

We assume that you have read and hopefully completed the following tutorials:

- RB_Getting_Started
- RB_Data_Tutorial
- RB_MCMC_Tutorial

This means that we will assume that you know how to execute and load data into `RevBayes`, are familiar with some basic commands, and know how to perform an analysis of a single-gene dataset (assuming an unconstrained/unrooted tree).

2 Data and files

We provide the necessary data and scripts that we will use in this tutorial. Of course, you may want to use your own dataset instead. In the **`data`** folder, you will find the following files

- **data/primates_and_galeopterus_cytb.nex**: A simple dataset consisting of 8 taxa with 500 characters simulated under the GTR model.

3 Specific Style Notes For This Tutorial

Throughout this tutorial you will see code snippets in various boxes like this:

```
CODE SNIPPET IN FILE
```

if the snippet is in a grey box like the one above, that is code you should inspect or tweak in a **.Rev** file. If the code is in blue like this:

```
CODE SNIPPET TO TYPE DIRECTLY INTO RevBayes
```

you should type this code directly into the `RevBayes` prompt in your terminal window.

4 Introduction

Assessing the fit of an evolutionary model to the data is critical as using a model with poor fit can lead to spurious conclusions. However, a critical evaluation of absolute model fit is rare in evolutionary studies. Posterior prediction is a Bayesian approach to assess the fit of a model to a given dataset (Bollback 2002; Brown 2014b; Gelman et al. 2014), that relies on the use of the posterior and the posterior predictive distributions. The posterior distribution is the standard output from Bayeisan phylogenetic inference. The posterior predictive distribution represents a range of possible outcomes given the assumptions of the model. The most common method to generate these possible outcomes, is to sample parameters from the posterior distribution, and use them to simulate new replicate datasets (Fig. 1). If these simulated datasets differ from the empirical dataset in a meaningful way, the model is failing to capture some salient feature of the evolutionary process.

The framework to construct posterior predictive distributions, and compare them to the posterior distribution is conveniently built in to `RevBayes`. In this tutorial we will walk you through using this functionality to perform a complete posterior predictive simulation on an example dataset.

4.1 Substitution Models

The models we use here are equivalent to the models described in the previous exercise on substitution models (continuous time Markov models). To specify the model please consult the previous exercise. Specifically, you will need to specify the following substitution models:

- Jukes-Cantor (JC) substitution model (Jukes and Cantor 1969)
- General-Time-Reversible (GTR) substitution model (Tavaré 1986)
- Gamma (+G) model for among-site rate variation (Yang 1994)
- Invariable-sites (+I) model (Hasegawa et al. 1985)

5 Assessing Model Fit with Posterior Prediction

The entire process of posterior prediction can be executed by using the **full_analysis.Rev** script in the **scripts** folder. If you were to type the following command into `RevBayes`:

```
source("scripts/full_analysis_JC.Rev")
```

the entire posterior prediction process would run on the example dataset. However, in this tutorial, we will walk through each step of this process on an example dataset.

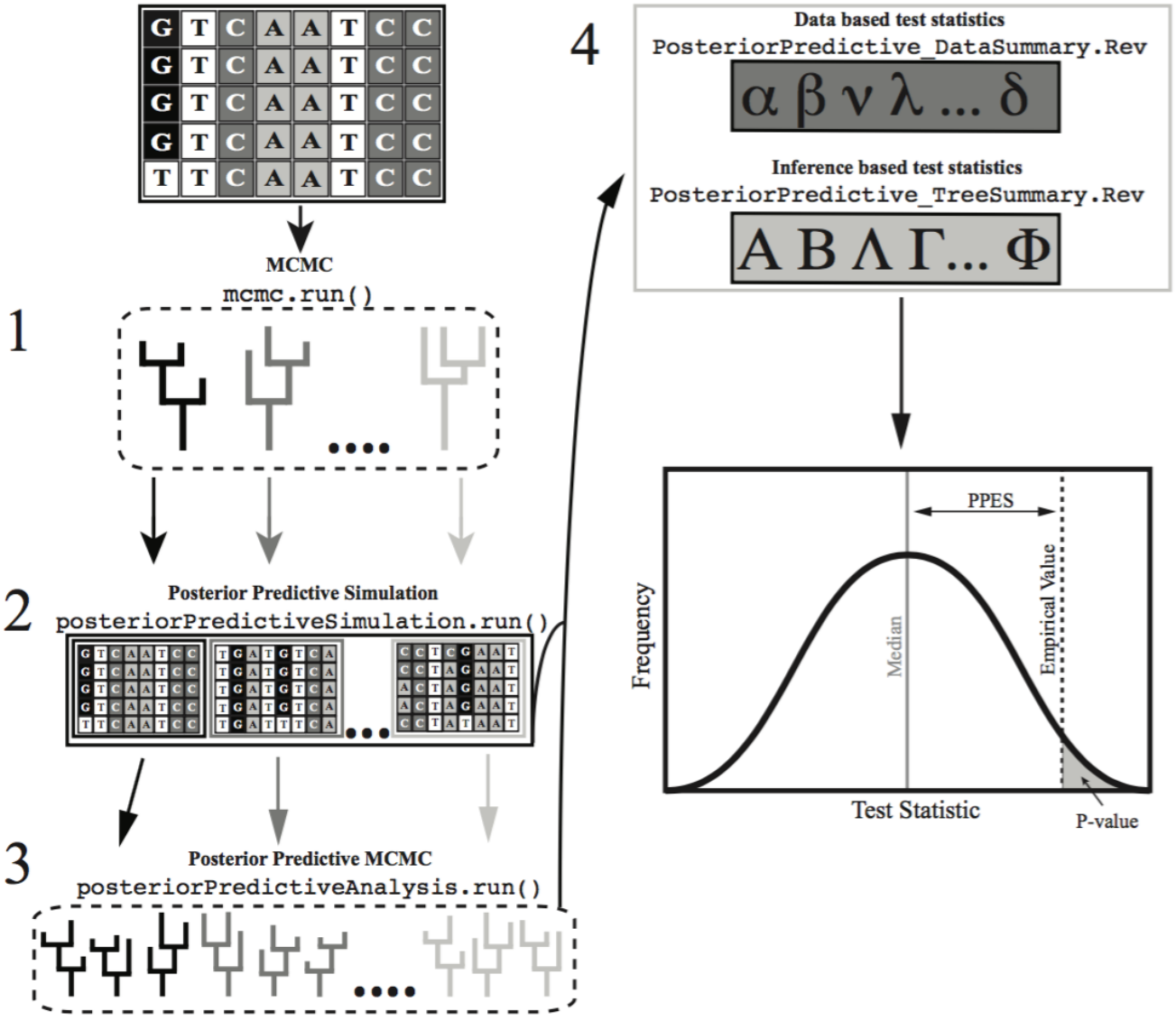

Figure 1: A cartoon schematic showing the process of assessing model fit with posterior predictive simulation built into P^3 and `RevBayes`.

5.1 Empirical MCMC Analysis

To begin, we first need to generate a posterior distribution from which to sample for simulation. This is the normal, and often only, step conducted in phylogenetic studies. Here we will specify our dataset, evolutionary model, and run a traditional MCMC analysis.

In the the **full_analysis_JC.Rev** script, this step is conducted with the following lines of RevScript:

```
## EMPIRICAL MCMC
inFile = "data/primates_and_galeopterus_cytb.nex"
analysis_name = "pps_example"
```

```
data <- readDiscreteCharacterData(inFile)
model_name = "JC"
model_file_name = "scripts/"+model_name+"_Model.Rev"
source("scripts/MCMC_Simulation.Rev")
```

Let's break down each of these lines and the scripts they call in order to better understand what is happening.

```
inFile = "data/primates_and_galeopterus_cytb.nex"
```

This line is just specifying an input file name. This is your data, in NEXUS format.

```
analysis_name = "pps_example"
```

This line is just specifying a general name to apply to your analysis. This will be used for future output files, so make sure it's something clear and easy to understand.

```
data <- readDiscreteCharacterData(inFile)
```

This line reads in the data file into a data object in `RevBayes`.

```
model_name = "JC"
model_file_name = "scripts/"+model_name+"_Model.Rev"
```

Here we are specifying an evolutionary substitution model to use with the MCMC analysis. In this specific case, we are using the Jukes-Cantor (JC) model for this example. If we open the **JC_Model.Rev** script, most of this script should look familiar from the

RB_MCMC_Tutorial. There are a couple of specific lines we can look at that might be a little different though, as we are using a unrooted tree for this analysis.

```
Q <-fnJC(4)
```

Here we are specifying that we should use the Jukes-Cantor model, and have it applied uniformly to all sites in the dataset. While this obviously is not likely to be a good fitting model for most datasets, we are using it for simplicity of illustrating the process.

```
topology ~ dnUniformTopology(names)
```

This sets a uniform prior on the tree topology.

```
br_lens[i] ~ dnExponential(10.0)
```

This sets an exponential distribution as the branch length prior.

```
phylogeny := treeAssembly(topology, br_lens)
```

This builds the tree by combining the topology with branch length support values.

After specifying a model, we will conduct the MCMC analysis with the script **MCMC_Simulation.Rev**. Much of that script should look familiar from the **RB_CTMC_Tutorial**, but let's take a look at the script and break down and revisit some of the major pieces as a refresher. Much of this will be a familiar to you from the **RB_CTMC_Tutorial**.

Here we specify the model we want to use. This is the name we specified earlier, in this case it's the variable holding the value **JC_Model.Rev**. This is just telling the MCMC script to use the Jukes-Cantor model we just reviewed.

```
source( model_file_name )
```

Now we need to specify some simple moves and monitors.

```
mni = 0
monitors[++mni] = mnModel(filename="output" + n + "/" + analysis_name + "_posterior.
    log",printgen=2500, separator = TAB)
```

```
monitors[++mni] = mnFile(filename="output" + n + "/" + analysis_name + "_posterior.
    trees",printgen=2500, separator = TAB, phylogeny)
```

```
monitors[++mni] = mnScreen(printgen=2500, TL)
```

```
monitors[++mni] = mnStochasticVariable(filename="output" + n + "/" + analysis_name + "
    _posterior.var",printgen=2500)
```

Here the monitors are just being advanced by the value of mni which is being incremented each generation.

Next, we will call the MCMC function, passing it the model, monitors and moves we specified above to set to build our mymcmc object.

```
mymcmc = mcmc(mymodel, monitors, moves, nruns=2)
```

Specify a burnin for this run.

```
mymcmc.burnin(generations=2500,tuningInterval=100)
```

Finally, we will execute the run for a specified number of generations. Here we are only using a small number of generations for the tutorial, however with empirical data you will most likely need a much larger number of generations to get a well mixed sample. The number of generations and printed generations is important to consider here for a variety of reasons, in particular for posterior predictive simulation. When we simulate datasets in the next step, we can only simulate 1 dataset per sample in our posterior. So, while the number of posterior samples will almost always be larger than the number of datasets we will want to simulate, it's something to keep in mind.

```
mymcmc.run(generations=50000)
```

Let's actually execute this step of the process in `RevBayes` by typing the full command at the `RevBayes` prompt:

```
inFile = "data/primates_and_galeopterus_cytb.nex"
analysis_name = "pps_example"
data <- readDiscreteCharacterData(inFile)
model_name = "JC"
model_file_name = "scripts/"+model_name+"_Model.Rev"
source("scripts/MCMC_Simulation.Rev")
```

This process should take approximately 15-20 minutes depending on the speed of your computer.

After the process completes, the results can be found in the **output** folder. You should see a number of familiar looking files, all with the name we provided under the **analysis_name** variable, **pps_example** in this case. Since we set the number of runs (nruns=2) in our MCMC, there will be two files of each type

(.log .trees .var) with an _N where N is the run number. You will also see 3 files without any number in their name. These are the combined files of the output. These will be the files we use for the rest of the process. If you open up one of the combined .var file, you should see that there are 200 samples. This was produced by our number of generations (250,000) divided by our number of printed generations, (2500) what we specified earlier. This is important to note, as we will need to thin these samples appropriately in the next step to get the proper number of simulated datasets.

Once you have examined the data files and are confident they are correct, go ahead clear() `RevBayes`.

```
clear()
```

5.2 Posterior Predictive Data Simulation

The next step of posterior predictive simulation is to simulate new datasets by drawing samples and parameters from the posterior distribution generated from the empirical MCMC anlaysis.

In the **full_analysis_JC.Rev** script, that is conducted using the following lines of RevScript:

```
inFile = "data/primates_and_galeopterus_cytb.nex"
analysis_name = "pps_example"
data <- readDiscreteCharacterData(inFile)
model_name = "JC"
model_file_name = "scripts/"+model_name+"_Model.Rev"
source("scripts/PosteriorPredictive_Simulation.Rev")
```

Most of these lines are the same as outlined above. Let's break down the new **source** call in detail though.

```
source("scripts/PosteriorPredictive_Simulation.Rev")
```

This is the script that simulates the datasets. Let's take a look at each of the calls in the **PosteriorPredictive_Simulation.Rev** script so that we can understand what is occurring.

```
source( model_file_name )
trace = readStochasticVariableTrace("output" + "/" + analysis_name + "_posterior.var",
    delimiter=TAB)
```

These lines set the model and read in the trace file of the Posterior Distribution of Variables.

```
pps = posteriorPredictiveSimulation(mymodel, directory="output" + "/" + analysis_name
   + "_post_sims", trace)
```

This line calls the **posteriorPredictiveSimulation()** function, which accepts any valid model of sequence evolution, and output directory, and a trace. For each line in the trace, it will simulate a new dataset under the specified model.

```
pps.run(thinning=2)
```

Now we run the posterior predictive simulation, generating a new dataset for each line in the trace file that was read in. This is the part where we need to decide how many simulated datasets we want to generate. If we just use the pps.run() command, one dataset will be generated for each sample in our posterior distribution. In this case, since we are reading in the combined posterior trace file with 200 samples, it would generate 200 simulated datasets. If you want to generate fewer, say 100 datasets, you need to use the thinning argument as above. In this case, we are thinning the output by 2, that is we are dividing our number of samples by 2. So that in our example case, we will end up simulating 100 new datasets.

Let's go ahead and run the simulation step. At your `RevBayes` prompt, execute the following commands:

```
inFile = "data/primates_and_galeopterus_cytb.nex"
analysis_name = "pps_example"
data <- readDiscreteCharacterData(inFile)
model_name = "JC"
model_file_name = "scripts/"+model_name+"_Model.Rev"
source("scripts/PosteriorPredictive_Simulation.Rev")
```

This process should finish in just a minute or two. If you look in the **output** folder, there should be another folder called **pps_example_post_sims**. This folder is where the simulated datasets are saved. If you open it up, you should see 100 folders named posterior_predictive_sim_*N*. Where *N* is the number of the simulated dataset. In each of these folders, you should find a seq.nex file. If you open one of those files, you'll see it's just a basic NEXUS file. These will be the datasets we analyze in the next step.

Once you have examined the data files and are confident they are correct, go ahead **clear()** `RevBayes`.

```
clear()
```

5.3 Posterior Predictive MCMC

The next step is to analyze the new simulated datasets that were generated in the above step. In the **full_analysis_JC.Rev** script that is accomplished using the following calls:

```
## POSTERIOR PREDICTIVE MCMC
inFile = "data/primates_and_galeopterus_cytb.nex"
analysis_name = "pps_example"
data <- readDiscreteCharacterData(inFile)
```

```
model_name = "JC"
model_file_name = "scripts/"+model_name+"_Model.Rev"
source("scripts/PosteriorPredictive_MCMC.Rev")
clear()
```

Again, most of these commands were covered earlier. Let's take a closer look at the new (source) call though.

```
source("scripts/PosteriorPredictive_MCMC.Rev")
```

Hopefully, this command is looking very familiar by now. This line is calling the **PosteriorPredictive_MCMC.Rev** script in the scripts folder. This script will perform an MCMC analysis on each of the simulated datasets we generated in the previous step. This step is the most time-consuming part of the analysis, so proceed with caution. While this script is mostly a standard MCMC analysis, there are some small changes. Let's briefly look at changes in this script for a better understanding.

```
mymcmc = mcmc(mymodel, monitors, moves, nruns=1)
```

This line sets up the mcmc object. This should look familiar, we are just calling the **mcmc()** function and passing it our model, monitors, moves, and number of runs to build a mymcmc object.

```
directory = "output" + "/" + analysis_name + "_post_sims"
```

This is a new command that specifies the output directory where the simulated datasets we generated in the previous section are found.

```
my_pps_mcmc = posteriorPredictiveAnalysis(mymcmc, directory)
```

This is also a new command. Last time we just called the .run() method of the mcmc object. This time we need to first call the **posteriorPredictiveAnalysis()** function. This function accepts our mcmc object, and an input directory that we specified in the previous command.

```
my_pps_mcmc.run(generations=50000)
```

Finally, we call the run method on the my_pps_mcmc object we created above. We specify the desired number of generations, and the .run() method will execute the complete MCMC analysis on each dataset in the directory we specified earlier. As you might expect, this process can take quite a long time if you

are conducting an entire MCMC analysis on a large number of simulated datasets. This is one of the steps in which taking advantage of the MPI functionality of `RevBayes` is critical in a real analysis. There is also ongoing work looking at faster approximations to speed up this entire process, so keep an eye out for those changes. For this tutorial, let's set our number of generations to something that will run in a reasonable amount of time, say 50,000.

Let's go ahead and run this step of the process. Execute the following commands at your `RevBayes` prompt:

```
inFile = "data/primates_and_galeopterus_cytb.nex"
analysis_name = "pps_example"
data <- readDiscreteCharacterData(inFile)
model_name = "JC"
model_file_name = "scripts/"+model_name+"_Model.Rev"
source("scripts/PosteriorPredictive_MCMC.Rev")
```

This part of the process should take around 45 minutes depending on the speed of your computer.

If you look the output folder, you should see 100 analysis folders names posterior_predictive_sim_N where N is a number corresponding to the seq.nex dataset that was analyzed. A quick look inside those folders should show a **pps_example_posterior.log** file, and a **pps_example_posterior.trees** file. This time, if you look at these files, you should see 100 samples in each file. While this is probably not what you'd want to do in a real analysis, for the sake of this tutorial it will work fine.

Once you have examined the data files and are confident they are correct, go ahead **`clear()`** `RevBayes`.

```
clear()
```

5.4 Calculating the Inference-Based Test Statistics

This is the final step, where we will calculate the test statistics from the posterior distributions generated in the previous round of MCMC analyses. We will calculate both the inference-based test statistics from the posterior distribution of **trees** and the test statistics from the data-based posterior distributions generated during the analysis process. The part in the **full_analysis_JC.Rev** script that generates the test statistics is the following line:

First, we will calculate the inference-based test statistics, which is done with the following lines:

```
## CALCULATE INFERENCE SUMMARY STATISTICS
inFile = "data/primates_and_galeopterus_cytb.nex"
analysis_name = "pps_example"
model_name = "JC"
model_file_name = "scripts/"+model_name+"_Model.Rev"
num_post_sims = listFiles(path="output_"+model_name+"/" + analysis_name + "_post_sims
    ").size()
data <- readDiscreteCharacterData(inFile)
```

```
source("scripts/PosteriorPredictive_TreeSummary.Rev")
clear()
```

The main script here is the **PosteriorPredictive_TreeSummary.Rev** script. We will look at the major concepts of this script to better understand how it works. For a more complete discussion of the statistics involved, please review Brown (2014a); Doyle et al. (2015). In general, this script and these statistics work by calculating the statistics of interest across each posterior distribution from the simulated datasets, and comparing those values to the values from the empirical posterior distribution.

The current version of this script generates several summary statistics:

- 25th Quantile
- 50th Quantile
- 75th Quantile
- 99th Quantile
- 999th Quantile
- Mean Tree Length
- Tree Length Variance
- Mean Robinson-Foulds Distance
- Entropy

The quantiles and Mean RF are calculated using the unweighted Robinson-Foulds distances calculated in a pairwise fashion between all trees in a posterior distribution (Fig. 2).

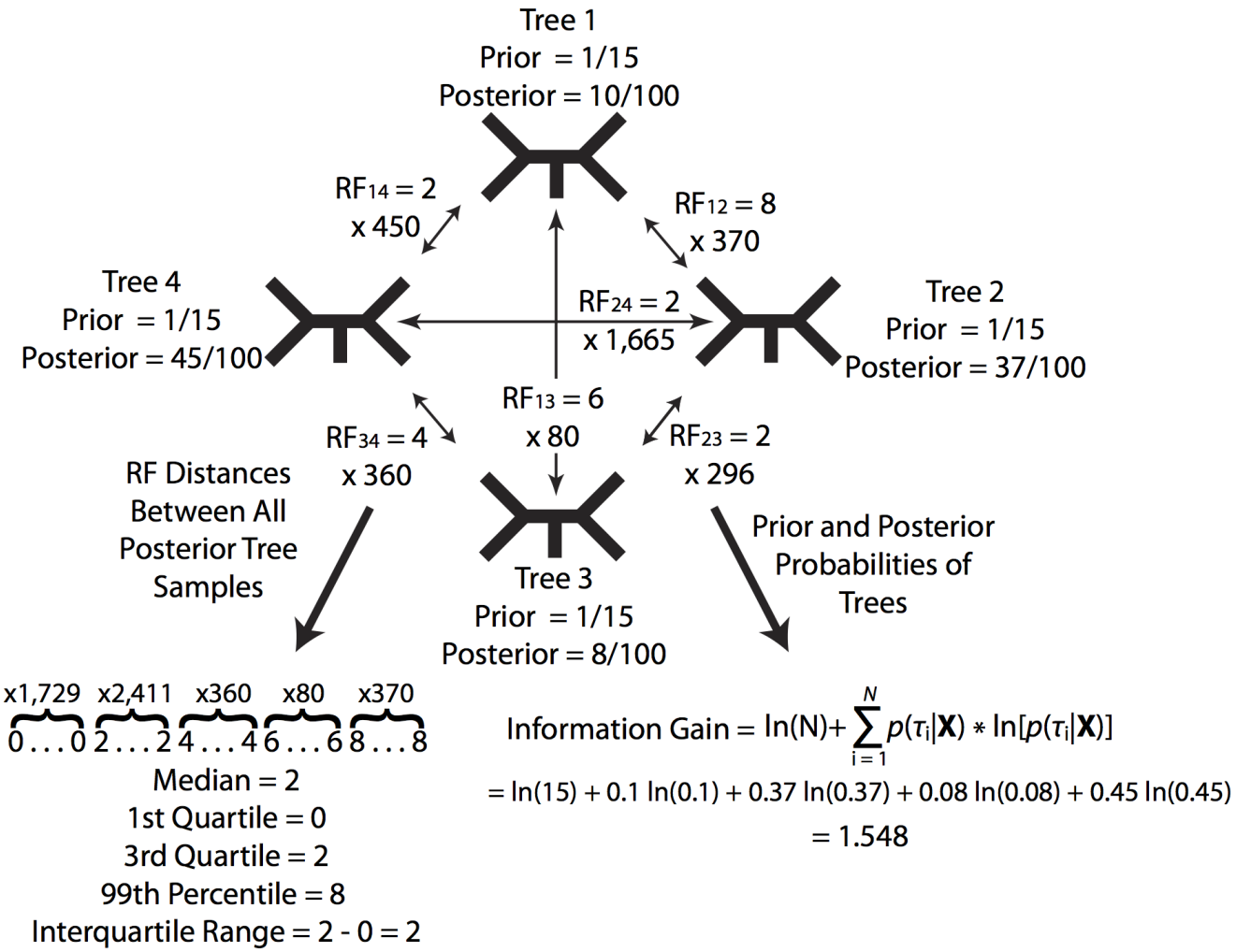

Figure 2: A diagram showing an example of calculating the inference-based test statistics for a single posterior predictive simulation (Brown 2014a).

There are functions built-in to to calculate these values for you. Here are some examples from the **PosteriorPredictive_TreeSummary.Rev** script:

```
rf_dists <- sim_tree_trace.computePairwiseRFDistances(credibleTreeSetSize=1.0,verbose=
    FALSE)
mean_rf <- mean(rf_dists)
quantile\_N = quantile(rf_dists, N)
```

For posterior predictive simulation, statistical entropy is analogous to the uncertainty associated with sampling from the prior or posterior. As the data provide more information and the posterior probabilities of various topologies become increasingly uneven, entropy decreases causing the difference between the entropy of the prior and the posterior to increase. Assuming a uniform prior on topologies, the change in entropy can be calculated as:

$$T_e(X, M_c) = ln[B(N)] + \sum_{i=1}^{B(N)} p(\tau_i|X) ln[p(\tau_i|X)]$$

(0)

The statistical entropy is calculated with the following function call:

```
sim_tree_trace = readTreeTrace(inFileName,treetype="non-clock")
entropy <- sim_tree_trace.computeEntropy(credibleTreeSetSize=1.0,verbose=FALSE)
```

These same statistics are calculated for both the posterior distributions from the simulated datasets and the posterior distribution from the empirical dataset.

Once we have the test statistics calculated for the simulated and empirical posterior distributions, we can compare the simulated to the empirical to get a goodness-of-fit. One simple way to do this is to calculate a posterior predictive *P*value for each of the test statistics of interest. This is done in the **full_analysis_JC.Rev** with the following lines :

```
## CALCULATE INFERENCE P-VALUES
analysis_name = "pps_example"
model_name = "JC"
emp_pps_file = "results_" + model_name + "/empirical_inference_" + analysis_name + ".
    csv"
sim_pps_file = "results_" + model_name + "/simulated_inference_" + analysis_name + ".
    csv"
outfileName = "results_" + model_name + "/inference_pvalues_effectsizes_" +
    analysis_name + ".csv"
statID = v("", "mean_rf", "quantile25", "quantile50", "quantile75", "quantile99", "
    quantile999", "mean_tl", "var_tl", "entropy")
source("scripts/PosteriorPredictive_PValues.Rev")
clear()
```

The 3 posterior predictive *P*values currently calculated by are:

- Lower 1-tailed
- Upper 1-tailed
- 2-tailed

The posterior predictive *P*value for a lower one-tailed test is the proportion of samples in the distribution where the value is less than or equal to the observed value, calculated as:

$$p_l = p(T(X_{rep}) \leqslant T(X)|(X)$$

The posterior predictive Pvalue for a lower one-tailed test is the proportion of samples in the distribution where the value is greater than or equal to the observed value, calculated as:

$$p_u = p(T(X_{rep}) \geqslant T(X)|(X)$$

and the two-tailed posterior predictive Pvalue is simply twice the minimum of the corresponding one-tailed tests calculated as:

$$p_2 = 2min(p_l, p_u)$$

Let's take a look at the **PosteriorPredictive_PValues.Rev** script to get a better idea of what is happening.

Using the equations outlined above, this script reads in the simulated and empirical data we just calculated, and calls a few simple functions:

```
## Calculate and return a vector of lower, equal, and upper pvalues for a given test
    statistic
p_values <- posteriorPredictiveProbability(numbers, empValue)

## 1-tailed
lower_p_value <- p_values[1]
equal_p_value <- p_values[2]
upper_p_value <- p_values[3]

## mid-point
midpoint_p_value = lower_p_value + 0.5*equal_p_value

## 2-tailed
two_tail_p_value = 2 * (min(v(lower_p_value, upper_p_value)))
```

these snippets calculate the various p-values of interest to serve as our goodness-of-fit test.

Another way that you can calculate the magnitude of the discrepancy between the empirical and PP datasets is by calculating the effect size of each test statistic. Effect sizes are useful in quantifying the magnitude of the difference between the empirical value and the distribution of posterior predictive values. The test statistic effect size can be calculated by taking the absolute value of the difference between the median posterior predictive value and the empirical value divided by the standard deviation of the posterior

predictive distribution (Doyle et al. 2015). Effect sizes are calculated automatically for the inference based test statistics in the P^3 analysis. The effect sizes for each test statistics are stored in the same output file as the *P*values.

```
effect_size = abs((m - empValue) / stdev(numbers))
```

calculates the effect size of a given test statistic.

Let's go ahead and calculate the inference-based test statistics now. At your `RevBayes` prompt, type the following commands:

```
analysis_name = "pps_example"
model_name = "JC"
emp_pps_file = "results_" + model_name + "/empirical_inference_" + analysis_name + ".csv
    "
sim_pps_file = "results_" + model_name + "/simulated_inference_" + analysis_name + ".csv
    "
outfileName = "results_" + model_name + "/inference_pvalues_effectsizes_" +
    analysis_name + ".csv"
statID = v("", "mean_rf", "quantile25", "quantile50", "quantile75", "quantile99", "
    quantile999", "mean_tl", "var_tl", "entropy")
source("scripts/PosteriorPredictive_PValues.Rev")
clear()
```

Depending on your computer and how densely you sampled the posterior this process can take from a few minutes to a few hours. If you followed along our tutorial suggestions, this step should take less than 30 minutes to complete.

Once execution of the script is complete, you should see a new directory, names **results_JC**. In this folder there should be 3 files. Each of these is a simple tab-delimited (TSV) file containing the test statistic calculation output.

- empirical_pps_example.tsv
- pps_example.tsv
- pvalues_pps_example.tsv

If you have these 3 files, and there are results in them, you can go ahead and clear() `RevBayes`.

```
clear()
```

You can get an estimate of how the model performed by examining the *P*values in the **inference_pvalues_effectsizes** and **data_pvalues_effectsizes_pps_example.csv** files. In this example, a quick view shows us that most of the statistics show a value less than 0.05. This leads to a rejection of the model, suggesting that the model employed is not a good fit for the data. This is the result we expected given we chose a very simplistic model (JC) that would likely be a poor fit for our data. However, it's a good example of the sensitivity of this method, showing how relatively short runtimes and a low number of generations will still detect poor fit.

5.5 Additional Exercises

Included in the **scripts** folder is a second model script called **GTR_Model.Rev**. As a personal exercise and a good test case, take some time now, and run the same analysis, substituting the **GTR_Model.Rev** model script for the **JC_Model.Rev** script we used in the earlier example. This is a good chance to run the full anlaysis using the single-run script by calling **full_analysis_GTR.Rev** script. You should get different results, this is an excellent chance to explore the results and think about what they suggest about the fit of the specified model to the dataset.

Once you have made the necessary changes, you should be able to execute the full PPS analysis by typing:

```
source("scripts/full_analysis.Rev")
```

5.5.1 Additional Exercises

Some Questions to Keep in Mind:

- Do you find the goodness-of-fit results to suggest that the GTR or JC model is a better fit for our data?
- Which test statistics seem to show the strongest effect from the use of a poorly fitting model?
- Other than *P*values, what other ways might you explore the test statistic distributions to identify poor fit?

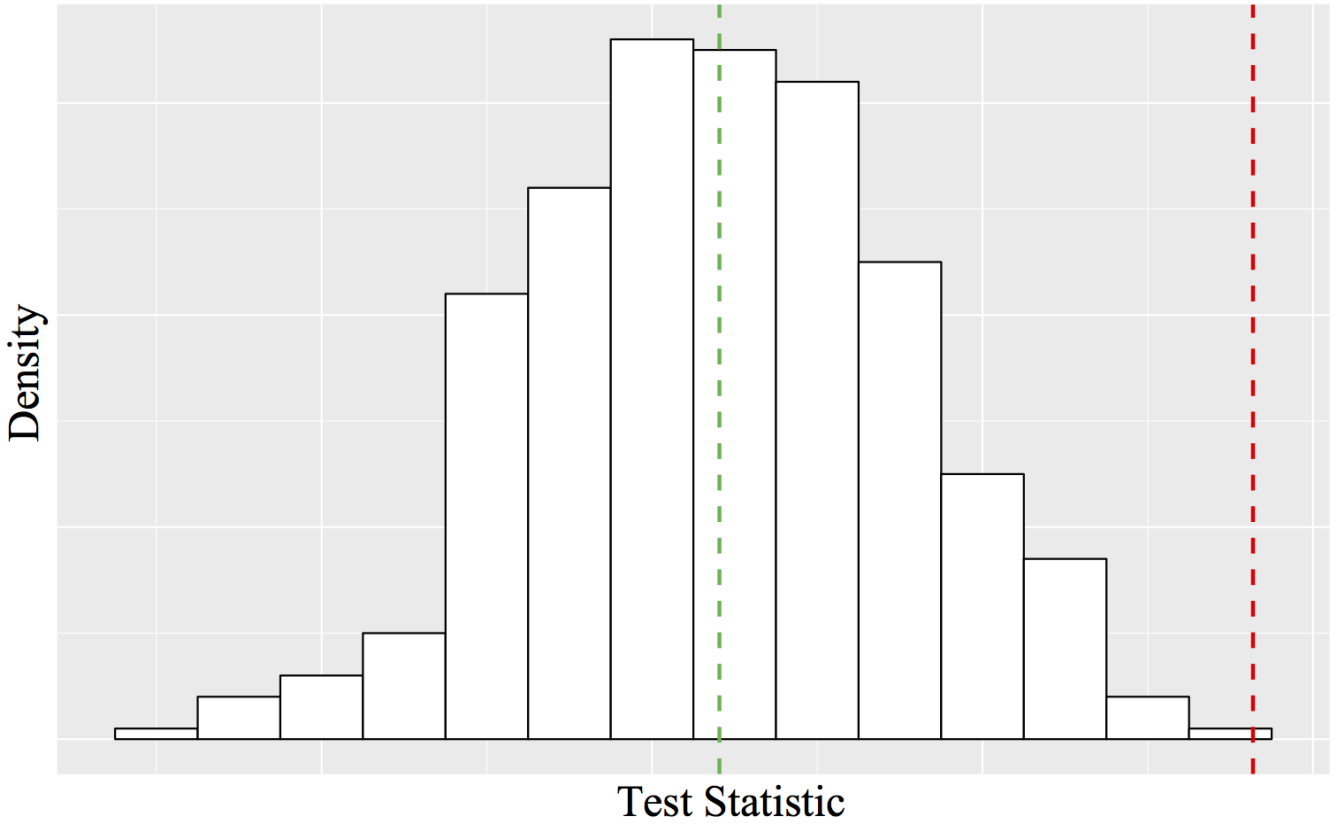

Figure 3: A cartoon showing a hypothesized distribution of a posterior predictive test statistic. The red line represents the empirical value in a poorly fitting model, the green line represents the empirical value in a reasonable model. Exploring the fit of a model to the data in a graphical way can often be the easiest way to identify a model with a poor fit.

6 For your consideration...

In this tutorial you have learned how to use RevBayes to assess the fit of a substitution model to a given sequence alignment. As you have discovered, the observed data should be plausible under the posterior predictive simulation if the model is reasonable. In phylogenetic analyses we choose a model, which explicitly assumes that it provides an reasonable explanation of the evolutionary process that generated our data. However, just because a model may be the 'best' model available, does not mean it is an appropriate model for the data. This distinction becomes both more critical and less obvious in modern analyses, where the number of genes often number in the thousands. Posterior predictive simulation in RevBayes, allows you to easily check model fit for a large number of genes by using global summaries to check the posterior predictive distributions with a comfortable goodness-of-fit style framework.

6.1 Batch Processing of Large Datasets

The process described above is for a single gene or alignment. However, batch processing a large number of genes with this method is a relatively straight forward process.

RevBayes has built in support for MPI so running RevBayes on more than a single processor, or on a cluster is as easy as calling it with openmpi.

For example:

```
mpirun -np 16 rb-mpi scripts/full_analysis.Rev
```

would run the entire posterior predictive simulation analysis on a single dataset using 16 processors instead of a single processor. Use of the MPI version of RevBayes will speed up the process dramatically.

Setting up the **full_analysis.Rev** script to cycle through a large number of alignments is relatively simple as well. One easy way is to provide a list of the data file names, and to loop through them. As an example:

```
data_file_list = "data_file_list.txt"
data_file_list <- readDiscreteCharacterData(data_file_list)
file_count = 0

for (n in 1:data_file_list.size()) {

FULL_ANALYSIS SCRIPT GOES HERE

file_count = file_count + 1
}
```

Then, anywhere in the full_analysis portion of the script that the lines

```
inFile = "data/8taxa_500chars_GTR.nex"
analysis_name = "pps_example"
```

appear, you would replace them with something along the lines of:

```
inFile = n
analysis_name = "pps_example_" + file_count
```

This should loop through all of the data files in the list provided, and run the full posterior predictive simulation analysis on each file. Using a method like this, and combining it with the MPI call above, you can scale this process up to multiple genes and spread the computational time across several cores to speed it up.

References

Bollback, J. P. 2002. Bayesian model adequacy and choice in phylogenetics. Mol. Biol. Evol. 19:1171–1180.

Brown, J. M. 2014a. Detection of implausible phylogenetic inferences using posterior predictive assessment of model fit. Syst. Biol. 63:334–348.

Brown, J. M. 2014b. Predictive approaches to assessing the fit of evolutionary models. Syst. Biol. 63:289–292.

Doyle, V. P., R. E. Young, G. J. P. Naylor, and J. M. Brown. 2015. Can we identify genes with increased phylogenetic reliability? Syst. Biol. .

Gelman, A., J. B. Carlin, H. S. Stern, and D. B. Rubin. 2014. Bayesian data analysis. Chapman & Hall/CRC.

Hasegawa, M., H. Kishino, and T. Yano. 1985. Dating of the human-ape splitting by a molecular clock of mitochondrial DNA. J. Mol. Evol. 22:160–174.

Jukes, T. H. and C. R. Cantor. 1969. Evolution of protein molecules. Pages 21–132 *in* Mammalian protein metabolism (H. N. Munro, ed.). Academic Press, New York.

Tavaré, S. 1986. Some probabilistic and statistical problems in the analysis of DNA sequences. Lectures on Mathematics in the Life Sciences (American Mathematical Society) 17:57–86.

Yang, Z. 1994. Maximum likelihood phylogenetic estimation from DNA sequences with variable rates over sites: approximate methods. J. Mol. Evol. 39:306–314.

Version dated: May 25, 2017