Phylogenetic Inference using RevBayes

Partitioned data analyses

(under construction)

Overview

This tutorial demonstrates how to accommodate variation in the substitution process across sites of an alignment. In the preceding tutorials, we assumed that all sites in an alignment evolved under an identical substitution process. This assumption is likely to be violated biologically, since different nucleotide sites are subject to different selection pressures, such as depending on which gene or codon position the site belongs to. Here, we will demonstrate how to specify—and select among—alternative *mixed models* using `RevBayes`. This is commonly referred to as partitioned-data analysis, where two or more subsets of sites in our alignment are assumed to evolve under distinct processes.

This tutorial will construct two multi-gene models. The first model, PS0, assumes all genes evolve under the same process parameters. The second model, PS1, assumes all genes evolve according to the same process, but each gene has it's own set of process parameters. The third model, PS2, partitions the data not only by gene, but also by codon position. Each analysis will generate a *maximum a posteriori* tree to summarize the inferred phylogeny. An advanced exercise introduces how to compute Bayes factors to select across various partitioning schemes.

All of the files for this analysis are provided for you and you can run these without significant effort using the **`source()`** function in the `RevBayes` console, e.g.

```
source("RevBayes_scripts/model_PS0.Rev")
```

If everything loaded properly, then you should see the program begin running the Markov chain Monte Carlo analysis needed for estimating the posterior distribution. If you continue to let this run, then you will see it output the states of the Markov chain once the MCMC analysis begins.

Requirements

We assume that you have previously completed the following tutorials:

- RB_Getting_Started
- RB_Data_Tutorial
- RB_CTMC_Tutorial
- RB_BayesFactor_Tutorial

Accordingly, 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 Bayes factor comparisons to select among competing substitution models.

Data and files

We provide several data files that we will use in this tutorial; these are the same datasets that we have used in previous tutorials. In the **`data`** folder, you will find the following files

- **`primates_cytb.nex`**: Alignment of the *cytochrome b* subunit from 23 primates representing 14 of the 16 families (*Indriidae* and *Callitrichidae* are missing).
- **`primates_cox2.nex`**: Alignment of the *COX-II* gene from the same 23 primates species.

Introduction

Variation in the evolutionary process across the sites of nucleotide sequence alignments is well established, and is an increasingly pervasive feature of datasets composed of gene regions sampled from multiple loci and/or different genomes. Inference of phylogeny from these data demands that we adequately model the underlying process heterogeneity; failure to do so can lead to biased estimates of phylogeny and other parameters (**?**).

Accounting for process heterogeneity involves adopting a 'mixed-model' approach, (**?**) in which the sequence alignment is first parsed into a number of partitions that are intended to capture plausible process heterogeneity within the data. The determination of the partitioning scheme is guided by biological considerations regarding the dataset at hand. For example, we might wish to evaluate possible variation in the evolutionary process within a single gene region (e.g., between stem and loop regions of ribosomal sequences), or among gene regions in a concatenated alignment (e.g., comprising multiple nuclear loci and/or gene regions sampled from different genomes). The choice of partitioning scheme is up to the investigator and many possible partitions might be considered for a typical dataset.

In this exercise, we assume that each partition evolved under an independent general-time reversible model with gamma-distributed rates across sites (GTR+Γ). Under this model the observed data are conditionally dependent on the exchangeability rates (θ), stationary base frequencies (π), and the degree of gamma-distributed among-site rate variation (α), as well as the unrooted tree topology (Ψ) and branch lengths (ν). We show the graphical model representation of the GTR+Γ mode in Figure **??**. When we assume different GTR+Γ models for each partitions, this results in a composite model, in which all sites are assumed to share a common, unrooted tree topology and proportional branch lengths, but subsets of sites ('data partitions') are assumed to have independent substitution model parameters. This composite model is referred to as a *mixed model*.

Finally, we perform a separate MCMC simulation to approximate the joint posterior probability density of the phylogeny and other parameters. Note that, in this approach, the mixed model is a fixed assumption of the inference (i.e., the parameter estimates are conditioned on the specified mixed model), and the parameters for each process partition are independently estimated.

For most sequence alignments, several (possibly many) partition schemes of varying complexity are plausible *a priori*, which therefore requires a way to objectively identify the partition scheme that balances estimation bias and error variance associated with under- and over-parameterized mixed models, respectively.

Increasingly, mixed-model selection is based on *Bayes factors* (*e.g.*, **?**), which involves first calculating the marginal likelihood under each candidate partition scheme and then comparing the ratio of the marginal likelihoods for the set of candidate partition schemes (**???**). The analysis pipeline that we will use in this tutorial is depicted in Figure 1.

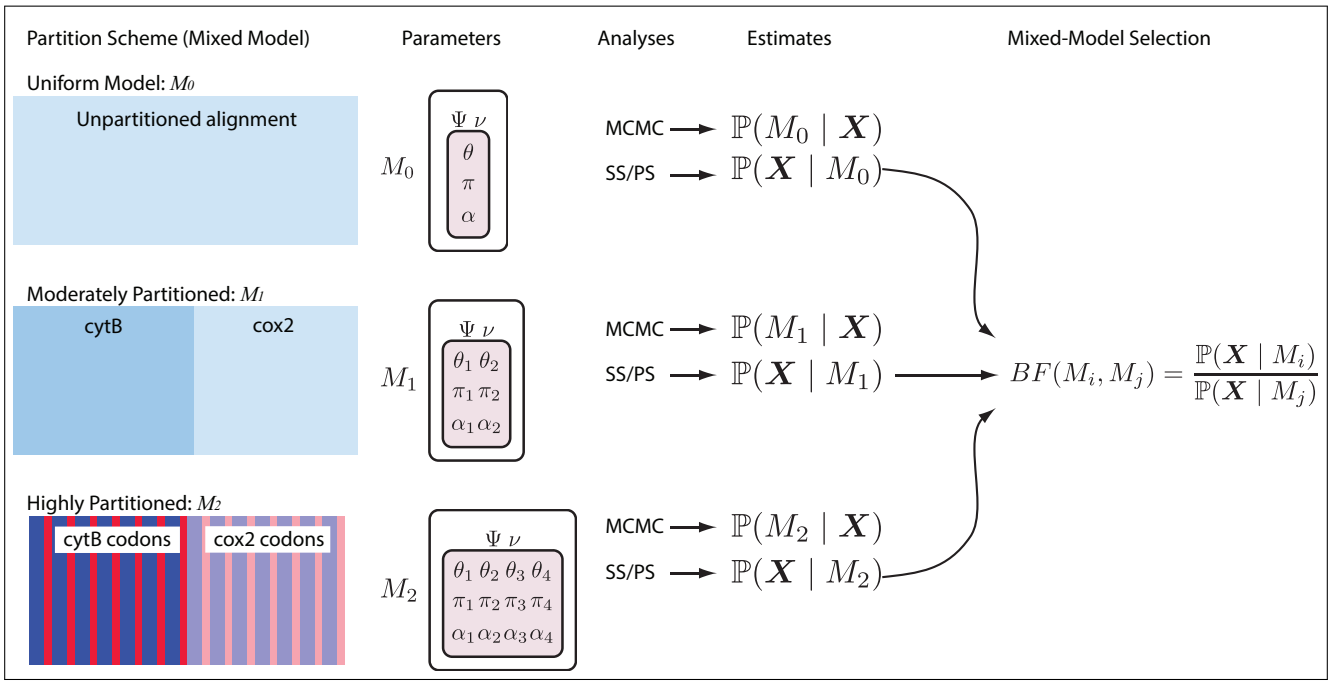

Figure 1: The analysis pipeline for Exercise 1. We will explore three partition schemes for the primates dataset. The first model (the 'uniform model', M_0) assumes that all sites evolved under a common GTR+Γ substitution model. The second model (the 'moderately partitioned' model, M_1) invokes two data partitions corresponding to the two gene regions (cytB and cox2), and assumes each subset of sites evolved under an independent GTR+Γ model. The final mixed model (the 'highly partitioned' model, M_2) invokes four data partitions—the first two partitions corresponds to the cytB gene region, where the first and second codon position sites share a partition distinct from the third codon position sites, and the cox2 has two partitions of its own, partitioned by codon positions in the same way—and each data partition is assumed evolved under an independent GTR+Γ substitution model. Note that we assume that all sites share a common tree topology, Ψ, and branch-length proportions, ν, for each of the candidate partition schemes. We perform two separate sets of analyses for each mixed model—a Metropolis-coupled MCMC simulation to approximate the joint posterior probability density of the mixed-model parameters, and a 'stepping-stone' MCMC simulation to approximate the marginal likelihood for each mixed model. The resulting marginal-likelihood estimates are then evaluated using Bayes factors to assess the fit of the data to the three candidate mixed models.

1 Concatenated, Non-partitioned

Our first exercise is to construct a multi-gene analysis where all genes evolve under the same process and parameters.

To begin, load in the sequences using the **readDiscreteCharacterData()** function.

```
data_cox2 = readDiscreteCharacterData("data/primates_cox2.nex")
data_cytb = readDiscreteCharacterData("data/primates_cytb.nex")
```

Since the first step in this exercise is to assume a single model across genes, we need to combine the two datasets using `concatenate()`

```
data = concatenate( data_cox2, data_cytb )
```

Typing **data** reports the dimensions of the concatenated matrix, this provides information about the alignment:

```
DNA character matrix with 23 taxa and 1852 characters
=====================================================
Origination:                primates_cox2.nex
Number of taxa:             23
Number of included taxa:    23
Number of characters:       1852
Number of included characters: 1852
Datatype:                   DNA
```

For later use, we will store the taxon labels (**names**), the number of species (**n_species**), the number of internal branches (**n_branches**), and the number of sites (**n_sites**).

```
names <- data.names()
n_species <- data.ntaxa()
n_branches <- 2 * n_species - 3
n_sites <- data.nchar()
```

Additionally, we will create some move and monitor index variables to create our move and monitor vectors.

```
mvi <- 1
mni <- 1
```

Now we can proceed with building our GTR+Γ model. First, we will define and specify a prior on the exchangeability rates of the GTR model

```
er_prior <- v(1,1,1,1,1,1)
er ~ dnDirichlet( er_prior )
```

and assign its move

```
moves[mvi++] = mvSimplexElementScale(er, alpha=10, tune=true, weight=15)
```

We can use the same type of distribution as a prior on the 4 stationary frequencies $(\pi_A, \pi_C, \pi_G, \pi_T)$ since these parameters also represent proportions. Specify a flat Dirichlet prior density on the base frequencies:

```
sf_prior <- v(1,1,1,1)
sf ~ dnDirichlet( sf_prior )
```

The node **sf** represents the π node in Figure **??**. Now add the simplex scale move on the stationary frequencies to the moves vector

```
moves[mvi++] = mvSimplexElementScale(sf, alpha=10, tune=true, weight=15)
```

We can finish setting up this part of the model by creating a deterministic node for the GTR rate matrix **Q**. The **fnGTR()** function takes a set of exchangeability rates and a set of base frequencies to compute the rate matrix used when calculating the likelihood of our model.

```
Q := fnGTR(er,sf)
```

We will also assume that the substitution rates vary among sites according to a gamma distribution, which has two parameters: the shape parameter, α, and the rate parameter, β. In order that we can interpret the branch lengths as the expected number of substitutions per site, this model assumes that the mean site rate is equal to 1. The mean of the gamma is equal to α/β, so a mean-one gamma is specified by setting the two parameters to be equal, $\alpha = \beta$. Therefore, we need only consider the single shape parameter, α (**?**). The degree of among-site substitution rate variation (ASRV) is inversely proportional to the value of the shape parameter—as the value of α-shape parameter increases, the gamma distribution increasingly resembles a normal distribution with decreasing variance, which corresponds to decreasing levels of ASRV (Figure 2). If $\alpha = 1$, then the gamma distribution collapses to an exponential distribution with a rate parameter equal to β. By contrast, when the value of the α-shape parameter is < 1, the gamma distribution assumes a concave distribution that places most of the prior density on low rates but allows some prior mass on sites with very high rates, which corresponds to high levels of ASRV (Figure 2).

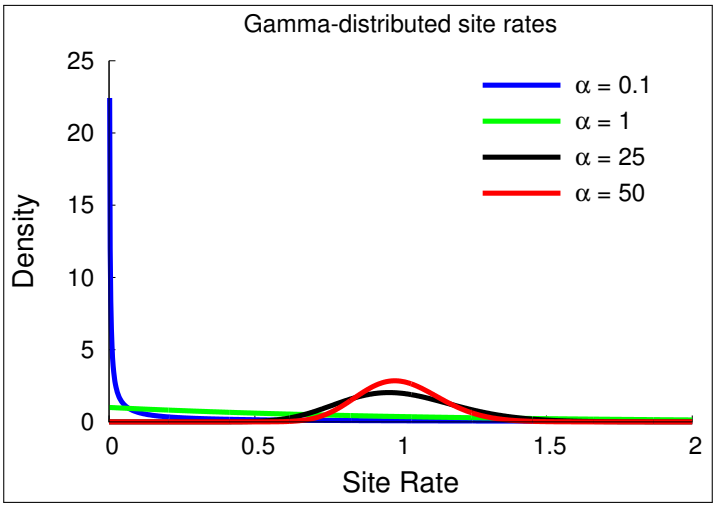

Figure 2: The probability density of mean-one gamma-distributed rates under different shape parameters.

Alternatively, we might not have good prior knowledge about the variance in site rates, thus we can place an uninformative, or diffuse prior on the shape parameter. For this analysis, we will use an exponential distribution with a rate parameter, **shape_prior**, equal to 0.05. Under an exponential prior, we are placing non-zero probability on values of α ranging from 0 to ∞. The rate parameter, often denoted λ, of an exponential distribution controls both the mean and variance of this prior such that the expected (or mean) value of α is: $\mathbb{E}[\alpha] = \frac{1}{\lambda}$. Thus, if we set $\lambda = 0.05$, then $\mathbb{E}[\alpha] = 20$.

Create a constant node called **shape_prior** for the rate parameter of the exponential prior on the gamma-shape parameter

```
alpha_prior_min <- 0.1
alpha_prior_max <- 50.0
```

Then create a stochastic node called **shape** to represent the α node in Figure **??**, with an exponential density as a prior:

```
alpha ~ dnUnif( alpha_prior_min, alpha_prior_max )
```

The way the ASRV model is implemented involves discretizing the mean-one gamma distribution into a set number of rate categories. Thus, we can analytically marginalize over the uncertainty in the rate at each site. To do this, we need a deterministic node that is a vector of rates calculated from the gamma distribution and the number of rate categories. The **fnDiscretizeGamma()** function returns this deterministic node and takes three arguments: the shape and rate of the gamma distribution and the number of categories. Since we want to discretize a mean-one gamma distribution, we can pass in **shape** for both the shape and rate.

Initialize the **gamma_rates** deterministic node vector using the **fnDiscretizeGamma()** function with **4** bins:

```
norm_gamma_rates := fnDiscretizeGamma( alpha, alpha, 4, false )
```

The random variable that controls the rate variation is the stochastic node **shape**. This variable is a single, real positive value (**RevType = RealPos**). We will apply a simple scale move to this parameter. The scale move's tuning parameter is called **lambda** and this value dictates the size of the proposal.

```
moves[mvi++] = mvScale(alpha, lambda=0.1, tune=false, weight=4.0)
```

Invariant sites (sites that remain fixed throughout their evolutionary history) may be seen as an extreme case of among-site rate variation. In contrast to $+\Gamma$ models, the $+I$ model allows site some probability of having substitution rate equal to zero. Only if a site has an invariant pattern (they are all are equal) there is some non-zero probability of the site being classified as invariant. (Seeing an invariant site pattern does not necessarily indicate the site evolved according to an invariant process: there is some probability of some substitution later being erased through a character reversion, which must be accounted for in the $+I$ model.)

Here, we give the probability of a site being invariant with `pinvar`

```
pinvar ~ dnBeta(1,1)
moves[mvi++] = mvScale(pinvar, lambda=0.1, tune=false, weight=2.0)
moves[mvi++] = mvSlide(pinvar, delta=10.0, tune=false, weight=2.0)
```

Note, if the $+\Gamma$ model allows for one rate category to take a very small value, it approximates the zero-rate value for the $+I$ model. This means when invariant site patterns are present, the $+\Gamma$ and $+I$ models vie to explain the same underlying evolutionary process of some sites evolving at a zero or near-zero rate.

The tree topology and branch lengths are also stochastic nodes in our model. In Figure **??**, the tree topology is denoted Ψ and the length of the branch leading to node i is ν_i.

We will assume the topology and divergence times are distributed by a birth-death process called **dnBDP()** in RevBayes. First, we will define the distribution's speciation rate, `lambda`, and an extinction rate `mu`

```
speciation ~ dnExponential(10.0)
extinction ~ dnExponential(10.0)
moves[mvi++] = mvScale(speciation, weight=5)
moves[mvi++] = mvScale(extinction, weight=5)
```

Since we know our analysis contains only 23 of 270 known primate species, we may inform the birth-death process only a portion of taxa were sampled. Failing to do so artificially inflates the extinction rate.

```
sampling_fraction <- 23. / 270 # 23 out of the ~ 270 primate species
```

To simplify the analysis, we assume the tree height is known to be 75, although in practice this quantity should be treated as a free parameter to estimate.

```
psi ~ dnBDP(lambda=speciation, mu=extinction, rho=sampling_fraction, rootAge=75, names
    =names)
```

Finally, we will provide moves so the MCMC may explore the space of tree topologies (`mvNNI` and `mvFNPR`, the nearest-neighbor interchange and fixed-node-height-prune-regraft proposals, resp.) and the space of divergence times (`mvNodeTimeSlideUniform`).

```
moves[mvi++] = mvNNI(psi, weight=10.0)
moves[mvi++] = mvFNPR(psi, weight=5.0)
moves[mvi++] = mvNodeTimeSlideUniform(psi, weight=10.0)
```

Also for simplicity, we assume a global molecular clock

```
clock ~ dnExponential(10.0)
moves[mvi++] = mvScale(clock, lambda=1, tune=true, weight=3.0)
```

We now have all the parameters needed to model the phylogenetic molecular substitution process

```
phyloSeq ~ dnPhyloCTMC(tree=psi, Q=Q, branchRates=clock, siteRates=norm_gamma_rates,
    pInv=pinvar, type="DNA")
```

To compute the likelihood, we condition the process on the data observed at the tips of the tree

```
phyloSeq.clamp(data)
```

Since the model is now specified, we wrap the components in a `Model` object.

```
mymodel = model(Q)
```

For our MCMC analysis we need to set up a vector of *monitors* to save the states of our Markov chain. The monitor functions are all called **mn***, where * is the wildcard representing the monitor type. First, we will initialize the model monitor using the **mnModel** function. This creates a new monitor variable that will output the states for all model parameters when passed into a MCMC function.

```
monitors[mni++] = mnModel(filename="output/PS0.params.txt",printgen=10)
```

The **mnFile** monitor will record the states for only the parameters passed in as arguments. We use this monitor to specify the output for our sampled trees and branch lengths.

```
monitors[mni++] = mnFile(psi, filename="output/PS0.tre", printgen=100)
```

Finally, create a screen monitor that will report the states of specified variables to the screen with **mnScreen**:

```
monitors[mni++] = mnScreen(er, sf, alpha, norm_gamma_rates, pinvar, speciation,
    extinction, clock, printgen=100)
```

With a fully specified model, a set of monitors, and a set of moves, we can now set up the MCMC algorithm that will sample parameter values in proportion to their posterior probability. The **mcmc()** function will create our MCMC object:

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

We can run the **.burnin()** member function if we wish to pre-run the chain and discard the initial states.

```
mymcmc.burnin(generations=10000,tuningInterval=100)
```

Now, run the MCMC:

```
mymcmc.run(generations=30000)
```

When the analysis is complete, you will have the monitor files in your output directory.

Methods for visualizing the marginal densities of parameter values are not currently available in `RevBayes`. Thus, it is important to use programs like Tracer (**?**) to evaluate mixing and non-convergence. (RevBayes does, however, have a tool for convergence assessment called **beca**.)

`RevBayes` can also summarize the tree samples by reading in the tree-trace file:

```
treetrace = readTreeTrace("output/model_PS0.tre")
treetrace.summarize()
```

The **mapTree()** function will summarize the tree samples and write the maximum a posteriori tree to file:

```
mapTree(treetrace,"output/model_PS0_map.tre")
```

This completes the uniform partition analysis. The next two sections will implement more complex partitioning schemes in a similar manner.

2 Partitioning by Gene Region

The uniform model used in the previous section assumes that all sites in the alignment evolved under the same process described by a shared tree, branch length proportions, and parameters of the GTR+Γ substitution model. However, our alignment contains two distinct gene regions—cytB and cox2—so we may wish to explore the possibility that the substitution process differs between these two gene regions. This requires that we first specify the data partitions corresponding to these two genes, then define an independent substitution model for each data partition.

First, we'll clear the workspace of all declared variables

```
clear()
```

Since we wish to avoid individually specifying each parameter of the GTR+Γ model for each of our data partitions, we can *loop* over our datasets and create vectors of nodes. To do this, we begin by creating a vector of data file names:

```
filenames <- v("data/primates_cox2.nex", "data/primates_cytb.nex")
```

Set a variable for the number of partitions:

```
n_parts <- filenames.size()
```

And create a vector of data matrices called **data**:

```
for (i in 1:n_parts){
   data[i] = readDiscreteCharacterData(filenames[i])
}
```

Next, we can initialize some important variables. This does require, however, that both of our alignments have the same number of species and matching tip names.

```
n_species <- data[1].ntaxa()
names <- data[1].names()
n_branches <- 2 * n_species - 3
```

```
mvi <- 1
mni <- 1
```

2.1 Specify the Parameters by Looping Over Partitions

We can avoid creating unique names for every node in our model if we use a **for** loop to iterate over our partitions. Thus, we will only have to type in our entire GTR+Γ model parameters once. This will produce a vector for each of the unlinked parameters — e.g., there will be a vector of **shape** nodes where the stochastic node for the first partition (cytB) will be **shape[1]** and the stochastic node for the second partition (cox2) will be called **shape[2]**.

```
for (i in 1:n_parts) {
  # exchangeability rates for partition i
  er_prior[i] <- v(1,1,1,1,1,1)
  er[i] ~ dnDirichlet(er_prior[i])
  moves[mvi++] = mvSimplexElementScale(er[i], alpha=10, tune=true, weight=3)

  # stationary frequencies for partition i
  pi_prior[i] <- v(1,1,1,1)
  pi[i] ~ dnDirichlet(pi_prior[i])
  moves[mvi++] = mvSimplexElementScale(pi[i], alpha=10, tune=true, weight=2)

  # rate matrix for partition i
  Q[i] := fnGTR(er[i],pi[i])

  # +Gamma for partition i
  alpha_prior_min[i] <- 0.1
  alpha_prior_max[i] <- 50.0
  alpha[i] ~ dnUnif( alpha_prior_min[i], alpha_prior_max[i] )
  norm_gamma_rates[i] := fnDiscretizeGamma( alpha[i], alpha[i], 4, false )
  moves[mvi++] = mvScale(alpha[i], lambda=0.8, tune=true, weight=3.0)

  # the probability of a site being invariable in partition i
  pinvar[i] ~ dnBeta(1,1)
  moves[mvi++] = mvScale(pinvar[i], lambda=0.1, tune=true, weight=2.0)
  moves[mvi++] = mvSlide(pinvar[i], delta=0.1, tune=true, weight=2.0)

  # specify a rate multiplier for each partition
  part_rate_mult[i] ~ dnExponential( 1.0 )
  moves[mvi++] = mvScale(part_rate_mult[i], lambda=1.0, tune=true, weight=2.0)
}
```

Our two genes evolve under different GTR rate matrices with different mean-one gamma distributions on the site rates. However, we do assume that they share a single topology and set of branch lengths.

```
speciation ~ dnExponential(10.)
extinction ~ dnExponential(10.)

# rescaling mv on speciation and extinction rates
moves[mvi++] = mvScale(speciation, lambda=1, tune=true, weight=3.0)
moves[mvi++] = mvScale(extinction, lambda=1, tune=true, weight=3.0)

sampling_fraction <- 23. / 270 # 23 out of the ~ 270 primate species

psi ~ dnBDP(lambda=speciation, mu=extinction, rho=sampling_fraction, rootAge=75, names
    =names)

# moves on the tree
moves[mvi++] = mvNNI(psi, weight=10.0)
moves[mvi++] = mvFNPR(psi, weight=5.0)
moves[mvi++] = mvNodeTimeSlideUniform(psi, weight=10.0)
```

which is the same as was specified for `model_PS0.Rev`.

Since we have a rate matrix and a site-rate model for each partition, we must create a phylogenetic CTMC for each gene. Additionally, we must fix the values of these nodes by attaching their respective data matrices. These two nodes are linked by the **`psi`** node and their log-likelihoods are added to get the likelihood of the whole DAG.

```
for (i in 1:n_parts) {
  phyloSeq[i] ~ dnPhyloCTMC(tree=psi, Q=Q[i], branchRates=part_rate_mult[i], siteRates=
      norm_gamma_rates[i], pInv=pinvar[i], type="DNA")
  phyloSeq[i].clamp(data[i])
}
```

The remaining steps should be familiar: wrap the model components in a model object

```
mymodel = model(psi)
```

create the monitors

```
monitors[mni++] = mnScreen(er, pi, alpha, norm_gamma_rates, pinvar, speciation,
    extinction, part_rate_mult, printgen=100)
monitors[mni++] = mnFile(psi, filename="output/PS1.tre", printgen=100)
monitors[mni++] = mnModel(filename="output/PS1.params.txt",printgen=10)
```

configure and run the MCMC analysis

```
mymcmc = mcmc(mymodel, moves, monitors)
mymcmc.burnin(10000,100)
mymcmc.run(30000)
```

and summarize the posterior density of trees with a MAP tree

```
treetrace = readTreeTrace("output/PS1.tre")
treetrace.summarize()
mapTree(treetrace,"output/PS1_map.tre")
```

3 Partitioning by Codon Position and by Gene

Because of the genetic code, we often find that different positions within a codon (first, second, and third) evolve at different rates. Thus, using our knowledge of biological data, we can devise a third approach that further partitions our alignment. For this exercise, we will partition sites within the cytB and cox2 gene by codon position.

```
clear()
data_cox2 <- readDiscreteCharacterData("data/primates_cox2.nex")
data_cytb <- readDiscreteCharacterData("data/primates_cytb.nex")
```

We must now add our codon-partitions to the **data** vector. The first and second elements in the **data** vector will describe cytB data, and the third and fourth elements will describe cox2 data. Moreover, the first and third elements will describe the evolutionary process for the first and second codon position sites, while the second and fourth elements describe the process for the third codon position sites alone.

We can create this by calling the helper function **setCodonPartition()**, which is a member function of the data matrix. We are assuming that the gene is *in frame*, meaning the first column in your alignment is a first codon position. The **setCodonPartition()** function takes a single argument, the position of the alignment you wish to extract. It then returns every third column, starting at the index provided as an argument.

Before we can use the use the **setCodonPartition()** function, we must first populate the position in the **data** matrix with some sequences. Then we call the member function of **data[1]** to exclude all but the 1^{st} and 2^{nd} positions for cox2.

```
data[1] <- data_cox2
data[1].setCodonPartition( v(1,2) )
```

Assign the 3^{rd} codon positions for cox2 to **data[2]**:

```
data[2] <- data_cox2
data[2].setCodonPartition( 3 )
```

Then repeat for cytB, being careful to store the subsetted data to elements 3 and 4:

```
data[3] <- data_cytb
data[3].setCodonPartition( v(1,2) )
data[4] <- data_cytb
data[4].setCodonPartition( 3 )
```

Now we can query the vector of data matrices to get the size, which is 4:

```
n_parts <- data.size()
```

Record the number of sites per partition:

```
for (i in 1:n_parts)
    n_sites[i] <- data[i].nchar()
```

And set the special variables from the data:

```
n_species <- data[1].ntaxa()
names <- data[1].names()
n_branches <- 2 * n_species - 3
```

Create index variables to populate the move and monitor vectors.

```
mvi <- 1
mni <- 1
```

Setting up the GTR+Γ model is just like in the two-gene analysis, except this time **n_parts** is equal to 4, so now our vectors of stochastic nodes should all contain nodes for the four pairs of codon-gene partitions.

```
for (i in 1:n_parts){

  # exchangeability rates for partition i
```

```
  er_prior[i] <- v(1,1,1,1,1,1)
  er[i] ~ dnDirichlet(er_prior[i])
  moves[mvi++] = mvSimplexElementScale(er[i], alpha=10.0, tune=true, weight=3.0)

  # stationary frequencies for partition i
  pi_prior[i] <- v(1,1,1,1)
  pi[i] ~ dnDirichlet(pi_prior[i])
  moves[mvi++] = mvSimplexElementScale(pi[i], alpha=10.0, tune=true, weight=2.0)

  # rate matrix for partition i
  Q[i] := fnGTR(er[i],pi[i])

  # +Gamma for partition i
  alpha_prior_min[i] <- 0.1
  alpha_prior_max[i] <- 50.0
  alpha[i] ~ dnUnif( alpha_prior_min[i], alpha_prior_max[i] )
  norm_gamma_rates[i] := fnDiscretizeGamma( alpha[i], alpha[i], 4, false )
  moves[mvi++] = mvScale(alpha[i], lambda=0.8, tune=true, weight=3.0)

  # the probability of a site being invariable
  pinvar[i] ~ dnBeta(1,1)
  moves[mvi++] = mvScale(pinvar[i], lambda=0.1, tune=true, weight=2.0)
  moves[mvi++] = mvSlide(pinvar[i], delta=0.1, tune=true, weight=2.0)

  # specify a rate multiplier for each partition
  part_rate_mult[i] ~ dnExponential( 1.0 )
  moves[mvi++] = mvScale(part_rate_mult[i], lambda=1.0, tune=true, weight=2.0)

}
```

Note that applying the per-partition model parameters only depends on the number of partitions, which means the script describing the partitioned model will work so long as you correctly populate the `data` vector.

We are still assuming that the genes share a single topology and branch lengths.

```
speciation ~ dnExponential(10.)
extinction ~ dnExponential(10.)

# rescaling mv on speciation and extinction rates
moves[mvi++] = mvScale(speciation, lambda=1, tune=true, weight=3.0)
moves[mvi++] = mvScale(extinction, lambda=1, tune=true, weight=3.0)

sampling_fraction <- 23. / 270 # 23 out of the ~ 270 primate species

psi ~ dnBDP(lambda=speciation, mu=extinction, rho=sampling_fraction, rootAge=75, names
    =names)

# moves on the tree
```

```
moves[mvi++] = mvNNI(psi, weight=10.0)
moves[mvi++] = mvFNPR(psi, weight=5.0)
moves[mvi++] = mvNodeTimeSlideUniform(psi, weight=10.0)
```

We must specify a phylogenetic CTMC node for each of our partition models.

```
for (i in 1:n_parts){
  phyloSeq[i] ~ dnPhyloCTMC(tree=psi, Q=Q[i], branchRates=part_rate_mult[i], siteRates=
     norm_gamma_rates[i], pInv=pinvar[i], type="DNA")
  phyloSeq[i].clamp(data[i])
}
```

And then wrap up the DAG using the **`model()`** function:

```
mymodel <- model(psi)
```

Monitor the model parameters, monitoring the tree seperately:

```
monitors[mni++] = mnScreen(er, pi, alpha, norm_gamma_rates, pinvar, speciation,
    extinction, part_rate_mult, printgen=100)
monitors[mni++] = mnFile(psi, filename="output/PS2.tre", printgen=100)
monitors[mni++] = mnModel(filename="output/PS2.params.txt",printgen=10)
```

Set up and run the MCMC analysis:

```
mymcmc = mcmc(mymodel, moves, monitors)
mymcmc.burnin(10000,100)
mymcmc.run(30000)
```

And, when MCMC completes, compute the MAP tree:

```
treetrace = readTreeTrace("output/PS2.tre")
treetrace.summarize()
mapTree(treetrace,"output/PS2_map.tre")
```

3.1 Exercises

- **1) Reviewing posterior estimates.** Open the `PS2.params.txt` file in Tracer. Remember that partitions 1 and 2 are for cox2, partitions 3 and 4 are for cytB, partitions 1 and 3 are for sites in the first and second codon positions (per gene), and partitions 2 and 4 are for sites in the third and fourth codon positions (per gene).

  Aside from the tree topology and divergence times, each partition is modeled to have its own set of parameters. However, the posterior estimates for some parameters appear quite similar between some pairs of partitions yet different between other pairs of partitions. For example, `part_rate_mult` is the per-partition molecular clock. This clock is approximately two orders of magnitude faster for partitions 2 and 4 (third codon position sites) than it is for partitions 1 and 3 (non-third codon position sites).

  Identify other parameter-partition relationships like this in the posterior. Under this model, would you consider the gene or the codon site position to hold greater influence over the site's evolutionary mode?

- **2) Comparison of MAP trees.** Open the three inferred MAP trees in FigTree. Check to enable "Node Labels", click "Display" and select "posterior" from the dropdown menu. Internal nodes now report the probability of the clade appearing in the posterior density of sampled trees. Do different models yield different tree topologies? Generally, do complex models provide higher or lower clade support?

- **3) Partitioned model selection.** Bayes factors are computed as the ratio of marginal likelihoods (see Section XXX for more details). Rather than constructing the analysis with an `mcmc` object, marginal likelihood computations rely on output from a `powerPosterior` object.

  Copy `model_PS0.Rev` to `marginal_PS0.Rev`. In `marginal_PS0.Rev`, delete all lines after the `model` function is called, so the MCMC is never run and the MAP tree is never computed.

  Instead, configure and run a power posterior analysis

```
pow_p = powerPosterior(mymodel, moves, "model1.out", cats=50)
pow_p.burnin(generations=10000,tuningInterval=1000)
pow_p.run(generations=1000)
```

  then compute the marginal likelihood using the stepping stone sampler

```
ss = steppingStoneSampler(file="PS0.out", powerColumnName="power",
    likelihoodColumnName="likelihood")
ss.marginal()
```

  and again using the path sampler

```
ps = pathSampler(file="PS0.out", powerColumnName="power", likelihoodColumnName="
    likelihood")
ps.marginal()
```

- **4) Customized partition models (advanced).** Create the partitioned model where first, second, and third codon positions are partitioned per gene. The substitution parameters for each partition are independent *except* the first and second codon positions per gene share stationary frequencies. These parameters would be $\pi_{12}^{(cytB)}$, $\pi_{3}^{(cytB)}$, $\pi_{12}^{(cox2)}$, $\pi_{3}^{(cox2)}$.

  The easiest way to accomplish this is to copy `model_PS2.Rev` to `model_PS3.Rev` and modify the new file's contents. Specifically, you will need to adjust how the codon position partitions are defined to yield six (instead of four) partitions. When looping over partitions, you will need an if-statement to assign stationary frequencies properly. Take some care when applying monitors and moves to the new model.