Statistical Phylogenetic Inference using RevBayes

Sebastian Höhna, Michael J. Landis, Tracy A. Heath, Bastien Boussau, Nicolas Lartillot, Brian R. Moore, Fredrik Ronquist, and John P. Huelsenbeck

July 15, 2015

Contents

Preface

This compilation of mostly independent exercises is our substitute for a complete manual of `RevBayes`. The first three chapters cover the basic ideas, installation requirements and commands for running `RevBayes`. The following chapters target specific parts of a phylogenetic analysis. Each chapter represents a 1.5 to 3 hour workshop session and is detailed enough to study independently at home. We have grouped the exercises together by topics into parts. We hope that you find the information provided helpful and welcome any feedback, criticism and suggestions.

Part I

Getting Started with RevBayes

Chapter 1

Introcution

1.1 Overview

RevBayes has as its central idea that any statistical model, for example a phylogenetic model, is composed of smaller parts that can be decomposed and put back together in a modular fashion. This comes from considering (phylogenetic) models as *probabilistic graphical models*, which lends flexibility and enhances the capabilities of the program. Users interact with RevBayes via an interactive shell. Users communicate commands using a language specifically designed for RevBayes, called Rev; an R-like language (complete with control statements, user-defined functions, and loops) that enables the user to build up (phylogenetic) models from simple parts (random variables, variable/parameter transformations, models, and constants of different sorts).

Here we assume that you have successfully installed RevBayes. If this isn't the case, then please consult our website on how to install RevBayes.

1.2 Getting Started

For the examples outlined in each tutorial, we will use RevBayes interactively by typing commands in the command-line console. For the exercises you can either use RevBayes interactively or run an entire script. Execute the RevBayes binary. If this program is in your path, then you can simply type in your Unix terminal:

- **$ rb**

When you execute the program, you will see a brief program information, including the current version number. Remember that more information can be obtained from ww.RevBayes.com. When you execute the program with an additional filename, *e.g.,*

- **$ rb my_analysis.Rev**

then RevBayes will run all commands specified in your file.

You may want to run RevBayes in parallel using multiple processes. This can be done by starting RevBayes with

- **$ mpirun -np 4 rb-mpi**

which starts 4 processes of RevBayes. You may want to change the number of processes depending on your available hardware.

The format of the exercises uses `lavender blush shaded boxes` to delineate code examples that you should type into RevBayes. For example, after opening the RevBayes program, you can load your data file:

```
data <- readDiscreteCharacterData("data/primates_cytb.nex")
```

The **RevBayes >** prompt that you see in your terminal is omitted so that the examples can be copied and pasted wholly. This is especially useful for larger command blocks, particularly loops, which will often be displayed in boxes

```
for( i in 1:12 ){
  x[i] ~ dnExponential(1.0)
}
```

The various `RevBayes` commands and syntax within the text are specified using **`typewriter text`**.

Most tutorials also includes hyperlinks: bibliographic citations are burnt orange and link to the full citation in the references, external URLs are cerulean, and internal references to figures and equations are purple.

The various exercises in this tutorial take you through the steps required to perform phylogenetic analyses of the example datasets. In addition, we have provided the `Rev` scripts and output files for some exercise so you can verify your results. (Note that since the MCMC runs you perform will start from different random seeds, the output files resulting from your analyses *will not* be identical to the ones we provide you.)

1.3 Probabilistic Graphical Models

`RevBayes` uses *probabilistic graphical models* for model specification, visualization, and implementation (Höhna et al. 2014). Graphical models are frequently used in machine learning and statistics to conceptually represent the conditional dependence structure of complex statistical models with many parameters (Gilks et al. 1994; Lunn et al. 2000; Jordan 2004; Koller and Friedman 2009; Lunn et al. 2009). The graphical model framework allows for flexible model specification and implementation and reduces redundant code. This framework provides a set of symbols for depicting a *directed acyclic graph* (DAG). Höhna et al. (2014) described the use of probabilistic graphical models for phylogenetics. The different nodes and components of a phylogenetic graphical model are shown in Figure 1.1 (Fig. 1 from Höhna et al. 2014).

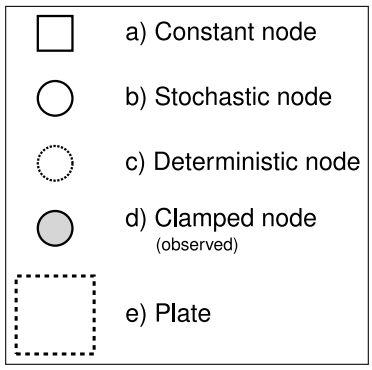


Figure 1.1: The symbols for a visual representation of a graphical model. a) Solid squares represent constant nodes, which specify fixed-valued variables. b) Stochastic nodes are represented by solid circles. These variables correspond to random variables and may depend on other variables. c) Deterministic nodes (dotted circles) indicate variables that are determined by a specific function applied to another variable. They can be thought of as variable transformations. d) Observed states are placed in clamped stochastic nodes, represented by gray-shaded circles. e) Replication over a set of variables is indicated by enclosing the replicated nodes in a plate (dashed rectangle). [Partially reproduced from Fig. 1 in Höhna et al. (2014).]

To represent the DAG, nodes are connected with arrows indicating dependency. A simple, albeit abstract, graphical model is shown in Figure 1.2. In this model, we observe a set of states for parameter x. We

assume that the values of x are samples from a lognormal distribution with a location parameter (log mean) μ and a standard deviation σ. It is more straightforward to model our uncertainty in the expectation of a lognormal distribution, rather than μ, thus we place a gamma distribution on the mean M. This gamma hyperprior has two parameters that we specify with fixed values (constant nodes): the shape α and rate β. The variable M is a stochastic node with this prior density. The standard deviation, σ, is also a stochastic node with an exponential prior density with rate parameter λ. For any value of M and any value of σ we can compute the deterministic variable μ using the formula $\mu = \ln(M) - \frac{\sigma^2}{2}$. This formula is known from using simple algebra on the equation for the mean of any lognormal distribution. With this model structure, we can then calculate the probability of the data conditional on the model and parameter values (the likelihood): $\mathbb{P}(\boldsymbol{x} \mid \mu, \sigma)$. Next we can get the posterior probability using Bayes' theorem:

$$\mathbb{P}(M, \sigma \mid \boldsymbol{x}, \alpha, \beta, \lambda) = \frac{\mathbb{P}(\boldsymbol{x} \mid \mu, \sigma)\mathbb{P}(M \mid \alpha, \beta)\mathbb{P}(\sigma \mid \lambda)}{\mathbb{P}(\boldsymbol{x})}.$$

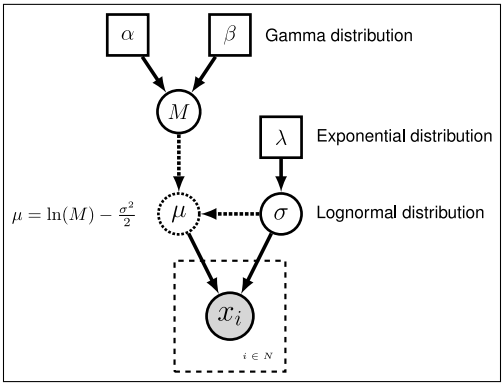


Figure 1.2: Graphical model representation of a simple lognormal model. A total of N observations of variable x are observed and occupy a clamped node. This parameter is log-normally distributed with parameters μ and σ (log mean and standard deviation, respectively). The parameter μ is a deterministic node that is calculated from the stochastic nodes M (the mean of the distribution) and σ. Dotted arrows indicate deterministic functions and are used to connect deterministic nodes to their parent variables. A gamma distribution is applied as a hyper prior on M with constant nodes for the shape α and rate β. The stochastic variable σ is exponentially distributed with fixed value for the rate λ.

1.4 Rev: The RevBayes Language

In `RevBayes` models and analyses are specified using an interpreted language called `Rev`. `Rev` bears similarities to the compiled language in WinBUGS and the interpreted `R` language. Setting up and executing a statistical analysis in `RevBayes` requires the user to specify all of the parameters of their model and the type of analysis (*e.g.,* an MCMC run). By using an interpreted language, `RevBayes` enables the practitioner to build complex, hierarchical models and to check the current states of variables while building the model. This will be very useful in the beginning. Later on you, when you run very complex analyses, you may want to write `Rev`-scripts.

Differently to `R` and BUGS, `Rev` is a strongly but implicitly typed language. It is implicitly typed, and thus similar to Python, because you do not need to provide the type of a variable (which you need to in languages such as C++ and Java). We do implicit typing to help users who do not know about the actual types of the variables. However, strongly typed means that every variable has a type and arguments of functions

need to match the required types. The strong type requirements ensures that you build meaningful model graphs. For example, the variance parameter of a normal distribution needs to be a positive number, and thus you can only use variables that are positive real numbers. `RevBayes` does automatic type conversion.

1.4.1 Specifying Models

Table 1.1: `Rev` assignment operators, clamp function, and plate/loop syntax.

Operator	Variable
`<-`	constant variable
`~`	stochastic variable
`:=`	deterministic variable
`node.clamp(data)`	clamped variable
`=`	inference (*i.e.,* non-model) variable
`for(i in 1:N){...}`	plate

The variables/parameters of a statistical model are created using different operators in `Rev` (Table 1.1). In Figure 1.3, the `Rev` syntax for creating the model in Figure 1.2 is provided. Because `Rev` is an interpreted language, it is important to consider the order in which you specify your variables (*cf.* BUGS where the order is not important). Thus, typically the first variables that are instantiated are *constant variables*. Constant variables require you to assign a fixed value using the `<-` operator. Stochastic variables are initialized using the `~` operator followed by the constructor function for a distribution. In `Rev`, the naming convention for distributions is `dn*`, where `*` is a wildcard representing the name of the distribution. Each distribution function requires hyper-parameters passed in as arguments. This is effectively linking nodes using arrows in the graphical model. The following code snippet creates a stochastic variable called `M` which is assigned a gamma-distributed hyperprior, with shape `alpha` and rate `beta`:

```
alpha <- 2.0
beta <- 4.0
M ~ dnGamma(alpha, beta)
```

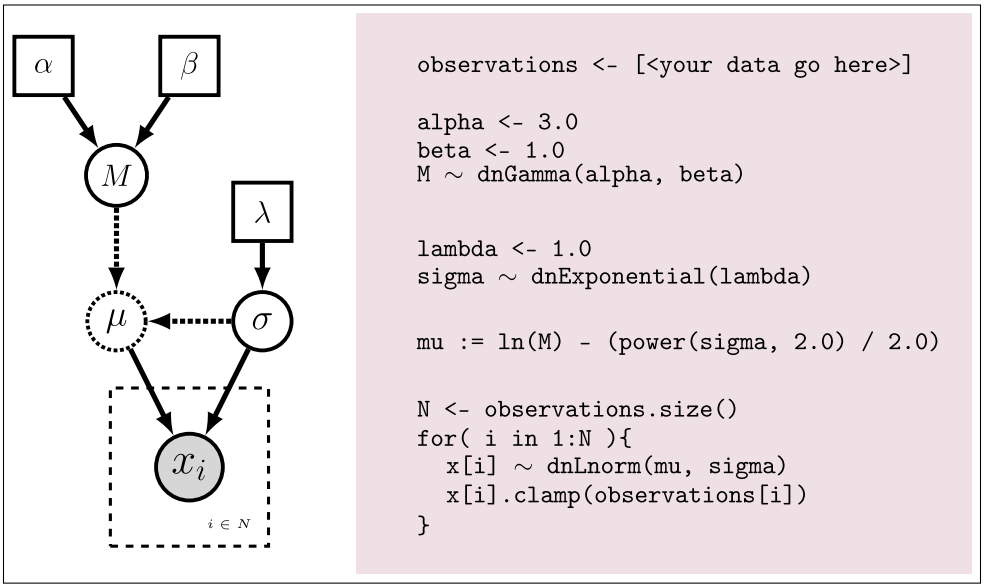
The flexibility gained from the graphical model framework and the interpreted language allows you to easily change a model by swapping components. For example, if you decide that a bimodal lognormal distribution is a better representation of your uncertainty in `M`, then you can simply change the distribution associated with `M` (after initializing the bimodal lognormal hyperparameters):

```
mean_1 <- 0.5
mean_2 <- 2.0
sd_1 <- 1.0
sd_2 <- 1.0
weight <- 0.5
M ~ dnBimodalLnorm(mean_1, mean_2, sd_1, sd_2, weight)
```

`Rev` does allow you to specify constant-variable values in the distribution constructor function, therefore this also works:

```
M ~ dnBimodalLnorm(0.5, 2.0, 1.0, 1.0, 0.5)
```

Both ways to specify priors are equivalent. The only difference is that one code may be more readable than the other.




```
observations <- [<your data go here>]

alpha <- 3.0
beta <- 1.0
M ~ dnGamma(alpha, beta)


lambda <- 1.0
sigma ~ dnExponential(lambda)

mu := ln(M) - (power(sigma, 2.0) / 2.0)


N <- observations.size()
for( i in 1:N ){
  x[i] ~ dnLnorm(mu, sigma)
  x[i].clamp(observations[i])
}
```

Figure 1.3: Specifying a model with `Rev`. The graphical model of the observed parameter x is shown on the left. In this example, x is log-normally distributed with a location parameter of μ and a standard deviation of σ, thus $x \sim \text{Lognormal}(\mu, \sigma)$. The expected value of x (or mean) is equal to M: $\mathbb{E}(x) = M$. In this model, M and σ are random variables and each are assigned hyperpriors. We assume that the mean is drawn from a gamma distribution with shape parameter α and rate parameter β: $M \sim \text{Gamma}(\alpha, \beta)$. The standard deviation of the lognormal distribution is assigned an exponential hyperprior with rate λ: $\sigma \sim \text{Exponential}(\lambda)$. Since we are conditioning our model on the *expectation*, we must compute the location parameter (μ) to calculate the probability of our model. Thus, μ is a deterministic node that is the result of the function* executed on M and σ: $\mu = \ln(M) - \frac{\sigma^2}{2}$. Since we observe values of x, we *clamp* this node.

Deterministic variables are parameter transformations and initialized using the **:=** operator followed by the function or formula for calculating the value. Previously we created a variable for the expectation of the lognormal distribution. Now, if you have an exponentially distributed stochastic variable σ, you can create a deterministic variable for the mean μ:

```
lambda <- 1.0
sigma ~ dnExponential(lambda)
mu := ln(M) - (sigma^2)/2.0
```

Replication over lists of variables as a plate object is specified using **for** loops. A for-loop is an iterator statement that performs a function a given number of times. In `Rev` you can use this syntax to create a vector of 7 stochastic variables, each drawn from a lognormal distribution:

```
for( i in 1:7 ) {
  x[i] ~ dnLognormal(mu, sigma)
}
```

The **for** loop executes the statement **x[i] ~ dnLognormal(mu, sigma)** for different values of i repeatedly, where i takes the values 1 to 7. Thus, we created a vector x of seven variables, each being independent and identically distributed (i.i.d.).

A clamped node/variable has observed data attached to it. Thus, you must first read in or input the data, then clamp it to a stochastic variable. In Figure 1.3 the observations are assigned and clamped to the stochastic variables. If we observed 7 values for **x** we would create 7 clamped variables:

```
observations <- [0.20, 0.21, 0.03, 0.40, 0.65, 0.87, 0.22]
N <- observations.size()
for( i in 1:N ){
  x[i].clamp(observations[i])
}
```

You may notice that the value of x has now changed and is equal to the observations.

1.5 Getting help in RevBayes

RevBayes provides an elaborate help system. Most of the help is found online on our website http://www.RevBayes.com. Within RevBayes you can display the help for a function, distribution or any other type using the **?** symbol followed by the command you want help for:

```
?dnNorm
?mcmc
?mcmc.run
```

Additionally, RevBayes will print the correct usage of a function if you only type in its name and hit return:

```
mcmc
   MCMC function (Model model, Monitor[] monitors, Move[] moves, String moveschedule = "
    sequential" | "random" | "single", Natural nruns)
```

If you typed in **?dnNorm** and you didn't see the help but got instead an error message then you have most likely an incorrect path variable to the help directory. You can check the current path to help directory by

```
getOption("helpdir")
   "/Users/hoehna/Software-Development/revbayes-development/help"
```

Check where the help files on your system are and then set the correct path

```
setOption("helpdir", "/Users/hoehna/Software-Development/revbayes-development/help")
```

1.5.1 RevBayes Users' Forum

An email list has been created for users of `RevBayes` to discuss `RevBayes`-related topics, including: `RevBayes` installation and use, scripting and programming, phylogenetics, population genetics, models of evolution, graphical models, etc. The forum is hosted by Google Groups:

- revbayes-users

Bibliography

Gilks, W., A. Thomas, and D. Spiegelhalter. 1994. A language and program for complex Bayesian modelling. The Statistician 43:169–177.

Höhna, S., T. A. Heath, B. Boussau, M. J. Landis, F. Ronquist, and J. P. Huelsenbeck. 2014. Probabilistic Graphical Model Representation in Phylogenetics. Systematic Biology 63:753–771.

Jordan, M. 2004. Graphical models. Statistical Science 19:140–155.

Koller, D. and N. Friedman. 2009. Probabilistic Graphical Models: Principles and Techniques. The MIT Press, Cambridge.

Lunn, D., D. Spiegelhalter, A. Thomas, and N. Best. 2009. The bugs project: Evolution, critique and future directions. Statistics in medicine 28:3049–3067.

Lunn, D. J., A. Thomas, N. Best, and D. Spiegelhalter. 2000. Winbugs-a bayesian modelling framework: concepts, structure, and extensibility. Statistics and computing 10:325–337.

Chapter 2

Introduction to RevBayes and Rev

2.1 Basic Rev Commands

2.1.1 Introduction

This tutorial demonstrates the basic syntactical features of RevBayes and Rev and shows how to set up and perform an analysis on "toy" statistical models for linear regression. This tutorial focuses on explaining probabilistic graphical models and the language Rev. A good reference for probabilistic graphical models for Bayesian phylogenetic inference is given in Höhna et al. (2014). The statistical examples are borrowed from a fourth year statistics course taught in the fall term 2011 at Stockholm University.

The first section of this tutorial involves

1. Creating different types of variables.
2. Learning about functions.

Then we will see how to perform statistical inference using RevBayes and Rev by implementing a Monte Carlo algorithm. Finally, we will see how RevBayes's built-in functions vastly simplify this inference task.

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:

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

Nevertheless, you will learn more if you type in the commands directly.

Let's start with the basic concepts for the interactive use of RevBayes with Rev (the language of RevBayes). You should try to execute the statements step by step, look at the output and try to understand what and why things are happening. We start with some simple concepts to get familiar and used to RevBayes. By now you should have executed RevBayes and you should see the command prompt waiting for input. The best exercise is to write these statements exactly in RevBayes.

Rev is an interpreted language for statistical computing and analyses in evolutionary biology. Therefore, the basics are simple mathematical operations, such as

```
# Simple mathematical operators:
1 + 1                          # Addition
10 - 5                         # Subtraction
5 * 5                          # Multiplication
10 / 2                         # Division
2^3                            # Exponentiation
5%2                            # Modulo
```

Just as a side note, you can also write multiple statements in the same line if you separate these by a semicolon (;). The statements will be executed as if you wrote each on a single line.

```
1 + 1; 2 + 2                # Multiple statements in one line
```

Here you can see that comments always start with the hash symbol (**#**). Everything after the '**#**'-symbol will be ignored. In addition to these simple mathematical operations, we provide some standard math functions which can be called by:

```
# Math-Functions
exp(1)                      # exponential function
ln(1)                       # logarithmic function with natural base
sqrt(16)                    # square root function
power(2,2)                  # power function: power(a,b) = a^b
```

Notice that `Rev` is case-sensitive. That means, `Rev` distinguishes upper and lower case letter for both variable names and function names. For example, only the first of these two calls will work

```
exp(1)                      # correct lower case name
Exp(1)                      # wrong upper case name
```

Moreover, we provide functions for the common statistical distributions.

```
# distribution functions
dexp(x=1,lambda=1)     # exponential distribution density function
qexp(0.5,1)            # exponential distribution quantile function
rexp(n=10,1)           # random draws from an exponential distribution
dnorm(-2.0,0.0,1.0)    # normal distribution density function
rnorm(n=10,0,1)        # random draws from a normal distribution
```

You may have noticed that we sometimes provided labels of the arguments and sometimes not. You can always provide the argument labels and then `RevBayes` will match the arguments based on the labels.

```
dnorm(x=0.5,mean=0.0,sd=1)    # normal distribution density function
```

If you do not provide the argument labels, then `RevBayes` will match the arguments by the best fitting types and the order in which you provided the arguments.

```
dnorm(0.5,0.5,1)       # correct order
dnorm(0.5,1,0.5)       # mismatched order
```

You may provide also just some arguments with labels and leave the other arguments without labels.

```
dnorm(0.0,x=0.5,sd=1)  # partially labeled
```

If you do not remember what the parameter name or parameter names of a function are, then you can simply type in the function name and RevBayes will tell you the possible parameters with their names.

```
dnorm
```

2.1.2 Variable Declaration

The next, and very important feature of RevBayes, is variable declaration. We have three types of (model) variables, namely constant, deterministic and stochastic variables, which represent the same three types of DAG nodes. Here we show how to construct the different variables and how they behave differently. First, we focus on the difference between constant and deterministic variables.

Let us begin by creating a constant variable with name **a** and assigned the value 1 to it. The left arrow assignment (**<-**) always creates a constant variable.

```
# Variable assignment: constant and deterministic
a <- 1                          # assignment of constant node 'a'
```

You see the value of 'a' by just typing in the variable name and pressing enter.

```
a                               # printing the value of 'a'
```

If you want to see which type of variable (constant, deterministic or stochastic) 'a' has, then call the structure function for it.

```
str(a)                          # printing the structure information of 'a'
   _variable    = a
   _RevType     = Natural
   _RevTypeSpec = [ Natural, Integer, RevObject ]
   _value       = 1
   _dagType     = Constant DAG node
   _children    = [ ]
   .methods = void function ()
```

An additional quite useful built-in function in RevBayes is the **type** function which gives you only the type information of the variable and thus is a subset of the **str** function.

```
type(a)                          # printing the type information of 'a'
   Natural
```

Next, we create a deterministic variable **b** using the **:=** assignment computed by **exp(a)** and another deterministic variable **c** computed by **ln(b)**. Deterministic variables are always created using the colon-equal assignment (**:=**).

```
b := exp(a)                      # assignment of deterministic node 'b' with the
   exponential function with parameter 'a'
b                                # printing the value of 'b'
c := ln(b)                       # assignment of deterministic node 'c' with logarithmic
   function with parameter 'b'
c                                # printing the value of 'c'
```

Again, you see the type of the variable and additional information such as which the parents and children are by calling the structure function on it.

```
str(b)                           # printing the structure information of 'b'
```

For example, see the difference to the creation of variable 'd', which is a constant variable.

```
d <- ln(b)                       # assignment of constant node 'd' with the value if the
   logarithmic function with parameter 'b'
d                                # printing the value of 'd'
str(d)                           # printing the structure information of 'd'
```

Currently, the variables **c** and **d** have the same value. We can check this using the equal comparison (**==**).

```
e := (c == d)
e
```

Now, if we assign a new value to variable **a**, then naturally the value of **a** changes. This has the consequence that all deterministic variables that use 'a' as a parameter, i.e., the variable **b**, change their value automatically too.

```
a <- 2                           # reassignment of variable a; every deterministic node
   which has 'a' as a parameter changes its value
```

```
a                               # printing the value of 'a'
b                               # printing the value of 'b'
c                               # printing the value of 'c'
d                               # printing the value of 'd'
e
```

Since variable **d** was a constant variable it did not change its value. This also means that **e** is now false.

Finally, we show you how to create the third type of variables in Rev: the stochastic variables. We will create a random variable **x** from an exponential distribution with parameter **lambda**. Stochastic assignments use the **~** operation.

```
# Variable assignment: stochastic
lambda <- 1                     # assign constant node 'lambda' with value '1'
x ~ dnExponential(lambda)       # create stochastic node with exponential distribution
    and parameter 'lambda'
```

The value of **x** is a random draw from the distribution. You can see the value and the probability (or log-probability) of the current value under the current parameter values by

```
x                               # print value of stochastic node 'x'
x.probability()                 # print the probability if 'x'
x.lnProbability()               # print the log-probability if 'x'
str(x)                          # printing all the information of 'x'
```

Similarly, we create a random variable **y** from a normal distribution by

```
mu <- 0
sigma <- 1
y ~ dnNorm(mu,sigma)
y.probability()                 # print the probability of 'y'
y.lnProbability()               # print the log-probability if 'y'
str(y)                          # printing all the information of 'y'
```

Now you know everything there is about creating the different types of variables and the different ways in which these variables behave.

Simple variable manipulation and other types of assignments

Rev provides some convenience variable manipulation operations that are equivalent to variable manipulations in other programming languages such as C/C++, Java and Python. You can increment (**++**) and

decrement (`-`) a variable. The increment operation increases the current value of a variable by 1 and the decrement operation decreases the value by 1. A post increment (`a++`) increases the value after returning the value, that is, the old value is returned. A pre increment (`++a`) increases the value before returning the value, that is, the new value is returned.

```
index <- 1
index++                         # post increment
++index                         # pre increment
index--                         # post decrement
--index                         # pre decrement
```

Additionally, you can use addition (`a += b`), subtraction (`a -= b`), multiplication (`a *= b`) and division (`a /= b`) to an existing variable.

```
index += 10                     # add 10 to the current value
index *= 2                      # double the current value
```

These variable manipulations will come in very handy for indices of vectors/arrays.

Vectors

Common values in `RevBayes` are of scalar types. That means, that not everything is a vector by default. Instead, you can create a vector using three different ways. First, you can call the vector function.

```
v <- v(1,2,3)                   # create a vector
```

Interestingly, we can use the same name for a variable as for a function: the variable `v` and the function `v(...)`. Both will still be fully functional and our interpreter checks if you asked for a function or a variable.

Second, you can use the square bracket notation.

```
w <- [1,2,3]                    # create a vector
```

And third, you can implicitly create the vector by assigning elements.

```
z[1] <-1                        # implicit creation of a vector
z[2] <-2
z[3] <-3
```

The implicit creation does not need to instantiate the variable beforehand. There are other useful built-in functions that produce vectors.

```
1:10                          # range function
rep(10,1)                     # replicate an element n times
seq(1,20,2)                   # built a sequence from a to b by c
```

Vectors in `Rev` belong to the class of objects that have methods. You can call a member method by

```
x.<method name>(<arguments>)
```

You have seen two methods previously, **`probability`** and **`lnProbability`**. If you don't remember what the methods were called, or if this object has any member methods, then you can get these by

```
v.methods()
```

In general, this is very, very useful. So for a vector we can get the size — the number of elements — by calling its member function:

```
v.size()
```

Control Structures

In this next part we will learn about control structures in `Rev`. The first control structure that we will look at is the **`for`** loop. **`for`** loop execute a single statement or a block of

```
# loops
for (<variable> in <set of value>) <single statement>

for (<variable> in <set of value>)
   <single statement>

for (<variable> in <set of value>) {
   <multiple statements>
   <multiple statements>
   <multiple statements>
}
```

The statement(s) will be execute for each value of variable of the **`for`** loop. A simple example is a **`for`** loop that computes the sum of

```
sum <- 0
for (i in 1:100) {
   sum <- sum + i
}
sum
```

Another example using a **for** loop is the computation of the Fibonacci number for a given integer.

```
# Fibonacci series using a for loop
fib[1] <- 1
fib[2] <- 1
for (j in 3:10) {
   fib[j] <- fib[j - 1] + fib[j - 2]
}
fib
```

We could also compute the Fibonacci numbers using a **while** loop. The **while** loop continues to execute the statement(s) until the condition is wrong.

```
# Fibonacci series using a while loop
fib[1] <- 1
fib[2] <- 1
j <- 3
while (j <= 10) {
   fib[j] <- fib[j - 1] + fib[j - 2]
   j++
}
fib
```

User Defined Functions

In Rev you can write your own functions as well. The syntax for writing function is:

```
function <return value type> <function name> (<list of arguments>) { <statements> }
```

As a simple example, let's write a function that computes the square of a number. We expect that the function takes in any real number. The type of real number is **Real**. Since the square is always a positive real number, we choose the return to be **RealPos**

```
# simple square function
function RealPos square ( Real x ) { x * x }
```

Now we can call our own function the same way as we call other already built-in function in `RevBayes`.

```
a <- square(5.0)
a
```

As an exercise, let's write a function that computes the factorial of a natural number.

```
# function for computing the factorial
function Natural fac(i) {
   if (i > 1) {
      return i * fac(i-1)
   } else {
      return 1
   }
}
b <- fac(6)
b
```

Here you see that within your own function you can call your function as well, which is commonly called recursive function calls.

Now let us write a recursive function for the sum of numbers which we computed before using a **`for`** loop.

```
# function for computing the sum
function Integer sum(Integer j) {
   if (j > 1) {
      return j + sum(j-1)
   } else {
      return 1
   }
}
c <- sum(100)
c
```

We can do the same for our favorite example, the Fibonacci series.

```
# function for computing the fibonacci series
function Integer fib(Integer k) {
   if (k > 1) {
      return fib(k-1) + fib(k-2)
   } else {
      return k
   }
}
d <- fib(6)
d
```

Now that should be enough to get you going with our first example analyses.

2.2 Exercise: Poisson Regression Model for Airline Fatalities

This exercise will demonstrate how to approximate the posterior distribution of some parameters using a simple Metropolis algorithm. The focus here lies in the Metropolis algorithm, Bayesian inference, and model specification—but not in the model or the data. After completing this computer exercise, you should be familiar with the basic Metropolis algorithm, analyzing output generated from a MCMC algorithm, and performing standard Bayesian inference.

2.2.1 Model and Data

We will use the data example from Gelman et al. (2003) (Table 2.1). A summary is given in table 2.1.

Table 2.1: Airline fatalities from 1976 to 1985. Reproduced from (**?**; Table 2.2 on p. 69).

Year	1976	1977	1978	1979	1980	1981	1982	1983	1984	1985
Fatalities	24	25	31	31	22	21	26	20	16	22

These data can be loaded into `RevBayes` by typing:

```
observed_fatalities <- v(24,25,31,31,22,21,26,20,16,22)
```

The model is a Poisson regression model with parameters α and β

$$y \sim \text{Poisson}(\exp(\alpha + \beta * x))$$

where y is the number of fatal accidents in year x. For simplicity, we choose uniform priors for α and β.

$$\begin{aligned} \alpha &\sim \text{Uniform}(-10, 10) \\ \beta &\sim \text{Uniform}(-10, 10) \end{aligned}$$

The probability density can be computed in `RevBayes` for a single year by

```
dpoisson(y[i],exp(alpha+beta*x[i]))
```

2.2.2 Problems

Metropolis Algorithm

The source file for this sub-exercise **`airline_fatalities_part1.Rev`**.

Let us construct a Metropolis algorithm that simulates from the posterior distribution $P(\alpha, \beta|y)$. We will construct this algorithm explicitly, without using the high-level functions existing in `RevBayes` to perform MCMC. In the next section, we will repeat the same analysis, this time using the high-level functions.

For simplicity of the calculations you can "normalize" the years, e.g.

```
x <- 1976:1985 - mean(1976:1985)
```

A common proposal distribution for $\alpha' \sim P(\alpha[i-1])$ is the normal distribution with mean $\mu = \alpha[i-1]$ and standard deviation $\sigma = \delta_\alpha$:

$$\alpha' \sim \text{norm}(alpha[i-1], delta\_alpha) \tag{2.1}$$

```
alpha_prime <- rnorm(1,alpha[i-1],delta_alpha)
```

A similar distribution should be used for β'.

```
delta_alpha <- 1.0
delta_beta <- 1.0
```

After you looked at the output of the MCMC, play around to find appropriate values for δ_α and δ_β.

Now we need to set starting values for the MCMC algorithm. Usually, these are drawn from the prior distribution, but sometimes if the prior is very uninformative, then these parameter values result into a likelihood of 0.0 (or log-likelihood of -Inf).

```
alpha[1] <- -0.01   # you can also use runif(-1.0,1.0)
beta[1] <- -0.01    # you can also use runif(-1.0,1.0)
```

Next, create some output for our MCMC algorithm. The output will be written into a file that can be read into R or Tracer (**?**).

```
# create a file output
write("iteration","alpha","beta",file="airline_fatalities.log")
write(0,alpha[1],beta[1],file="airline_fatalities.log",append=TRUE)
```

Note that we need a first iteration with value 0 so that Tracer can load in this file.

Finally, we set up a **for** loop over each iteration of the MCMC.

```
for (i in 2:10000) {
```

Within the **for** loop we propose new parameter values.

```
   alpha_prime <- rnorm(1,alpha[i-1],delta_alpha)[1]
   beta_prime <- rnorm(1,beta[i-1],delta_beta)[1]
```

For the newly proposed parameter values we compute the prior ratio. In this case we know that the prior ratio is 0.0 as long as the new parameters are within the limits.

```
   ln_prior_ratio <- dunif(alpha_prime,-10.0,10.0,log=TRUE) + dunif(beta_prime
       ,-10.0,10.0,log=TRUE) - dunif(alpha[i-1],-10.0,10.0,log=TRUE) - dunif(beta[i
       -1],-10.0,10.0,log=TRUE)
```

Similarly, we compute the likelihood ratio for each observation.

```
   ln_likelihood_ratio <- 0
   for (j in 1:x.size() ) {
      lambda_prime <- exp( alpha_prime + beta_prime * x[j] )
      lambda <- exp( alpha[i-1] + beta[i-1] * x[j] )
      ln_likelihood_ratio += dpoisson(observed_fatalities[j],lambda_prime) - dpoisson(
          observed_fatalities[j],lambda)
   }
   ratio <- ln_prior_ratio + ln_likelihood_ratio
```

And finally we accept or reject the newly proposed parameter values with probability **ratio**.

```
   if ( ln(runif(1)[1]) < ratio) {
      alpha[i] <- alpha_prime
      beta[i] <- beta_prime
   } else {
      alpha[i] <- alpha[i-1]
      beta[i] <- beta[i-1]
   }
```

Then we log the current parameter values to the file by appending the file.

```
   # output to a log-file
   write(i-1,alpha[i],beta[i],file="airline_fatalities.log",append=TRUE)
}
```

As a quick summary you can compute the posterior mean of the parameters.

```
mean(alpha)
mean(beta)
```

You can also load the file into R or Tracer to analyze the output.

In this section of the first exercise we wrote our own little Metropolis algorithm in Rev. This becomes very cumbersome, difficult and slow if we'ld need to do this for every model. Here we wanted to show you only the basic principle of any MCMC algorithm. In the next section we will use the built-in MCMC algorithm of RevBayes.

MCMC analysis using the built-in algorithm in RevBayes

Before starting with this new approach it would be good if you either start a new RevBayes session or clear all previous variables using the **clear** function. Currently we may have some minor memory problems and if you get stuck it may help to restart RevBayes.

We start by loading in the data to RevBayes.

```
observed_fatalities <- v(24,25,31,31,22,21,26,20,16,22)
x <- 1976:1985 - mean(1976:1985)
```

Then we create the parameters with their prior distributions.

```
alpha ~ dnUnif(-10,10)
beta ~ dnUnif(-10,10)
```

It may be good to set some reasonable starting values especially if you choose is very uninformative prior distribution. If by chance you had starting values that gave a likelihood of -Inf, then RevBayes will try several times to propose new starting values drawn from the prior distribution.

```
# let us use reasonable starting value
alpha.setValue(0.0)
beta.setValue(0.0)
```

Our next step is to set up the moves. Moves are algorithms that propose new values and know how the reset the values if the proposals are rejected. We use the same sliding window move as we implemented above by ourselves.

```
mi <- 0
moves[mi++] = mvSlide(alpha)
moves[mi++] = mvSlide(beta)
```

Then we set up the model. This means we create a stochastic variable for each observation and clamp its value with the observed data.

```
for (i in 1:x.size() ) {
    lambda[i] := exp( alpha + beta * x[i] )
    y[i] ~ dnPoisson(lambda[i])
    y[i].clamp(observed_fatalities[i])
}
```

We can now create the model by pulling the up the model graph from any variable that is connected to our model graph.

```
mymodel = model( alpha )
```

We also need some monitors that report the current values during the MCMC run. We create two monitors, one printing all numeric non-constant variables to a file and one printing some information to the screen.

```
monitors[1] = mnModel(filename="output/airline_fatalities.log",printgen=10, separator
    = " ")
monitors[2] = mnScreen(printgen=10, alpha, beta)
```

Finally we create an MCMC object. The MCMC object takes in a model object, the vector of monitors and the vector of moves.

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

On the MCMC object we call its member method **run** to run the MCMC.

```
mymcmc.run(generations=3000)
```

And now we are done ☺

Posterior Distribution of α and β

Report the posterior mean and 95% credible intervals for α and β. Additionally, plot the posterior distribution of α and β by plotting a histogram of the samples. You can use the `R` function

Plot the curve of $m(x) = \mathrm{E}[\exp(\alpha + \beta * x)|y]$ for $x = [1976, 1985]$. You can generate draws from the posterior distribution of the expected value for a specific x by recording the current expected value at a

iteration i of the Metropolis algorithm $m\_sample(x)[i] = \mathrm{E}[\exp(\alpha[i] + \beta[i] * x)|y]$ and taking the mean of those samples (`m(x) = mean(m_sample(x))`) afterwards. Since `RevBayes` provides you with the samples of $m(x) = \mathrm{E}[\exp(\alpha + \beta * x)|y] = \lambda_x$ you can simply plot these posterior curves.

Produce a histogram of the predictive distribution of the number of fatalities in 2014 and estimate the posterior mean. The predictive distribution can be approximated simultaneously with the Metropolis algorithm. This means, for any iteration i you simulate draws from the conditional distribution for $x = 2014$ and the current values of $\alpha[i]$ and $\beta[i]$.

Estimate the distribution of the mean of the posterior predictive distribution of the the number of fatalities in 2014. Therefore, let us denote the expected value of the posterior distribution by μ. Since we do not know this value μ exactly, we can follow the Bayesian approach and associate a probability for each value m as being the true expected value of the posterior distribution, given the observations y ($P(m = \mu|y)$). You can be approximate this distribution by recording the expected value for the number of fatalities in 2014 ($\mathrm{E}[\exp(\alpha + \beta * x)|y]$) in each iteration i of the Metropolis algorithm. Plot a histogram of the expected values, compute the mean of the expected values and compare it to the previously obtained estimate of the mean of the posterior predictive distribution.

Follow the same approach as for the posterior predictive distribution for $x = 2015$, but this time for $x = 2016$ and estimate the probability of no fatality.

2.3 Exercise: Poisson Regression Model for Coal-mine Accidents

We will analyze a dataset coal-mine accidents. The values are the dates of major (more than 10 casualties) coal-mining disasters in the UK from 1851 to 1962.

2.3.1 A model for disasters

A common model for the number of events that occur over a period of time is a Poisson process, in which the number of events in disjoint time-intervals are independent and Poisson-distributed. We will discretize and look at the yearly number of accidents.

In order to take into account the possible change of rate, we will allow for different rates before and after year θ, where θ is unknown to us. Thus, the observation distribution of our model is $y_t \sim Poisson(\lambda_t)$ with $t = 1851, \ldots, 1962$ and

$$\lambda_t = \begin{cases} \beta & \text{if } t < \theta \\ \gamma & \text{if } t \geq \theta \end{cases}$$

Thus, the rate t is defined by three unknown parameters: β, γ and θ. A hierarchical choice of priors is given by

$$\begin{aligned} \eta &\sim Gamma(10.0; 20.0) \\ \beta &\sim Gamma(2.0; \eta) \\ \gamma &\sim Gamma(2.0; \eta) \\ \theta &\sim Uniform(1852, \ldots, 1962) \end{aligned}$$

which brings an additional parameter η in the model. For θ we have used an uniform prior over the years, but excluded year 1851 in order to make sure at least one year has rate β. The hierarchical prior carry the belief that β and γ are somewhat similar in size, since they both depend on η.

The model in Rev

We start as usual by loading in the data.

```
observed_fatalities <- v(4, 5, 4, 1, 0, 4, 3, 4, 0, 6, 3, 3, 4, 0, 2, 6, 3, 3, 5, 4,
   5, 3, 1, 4, 4, 1, 5, 5, 3, 4, 2, 5, 2, 2, 3, 4, 2, 1, 3, 2, 2, 1, 1, 1, 1, 3, 0, 0,
    1, 0, 1, 1, 0, 0, 3, 1, 0, 3, 2, 2, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 2, 1, 0, 0,
   0, 1, 1, 0, 2, 3, 3, 1, 1, 2, 1, 1, 1, 1, 2, 3, 3, 0, 0, 0, 1, 4, 0, 0, 0, 1, 0, 0,
    0, 0, 0, 1, 0, 0, 1, 0, 1)
year <- 1851:1962
```

In Rev we specify this prior choice by

```
eta ~ dnGamma(10.0,20.0)
beta ~ dnGamma(2.0,eta)
gamma ~ dnGamma(2.0,eta)
theta ~ dnUnif(1852.0,1962.0)
```

Then we select moves for each parameter. For the rate parameters — which are defined only on the positive real line — we choose a scaling move. Only for **theta** we choose the sliding window proposal.

```
mi <- 0
moves[mi++] = mvScale(eta)
moves[mi++] = mvScale(beta)
moves[mi++] = mvScale(gamma)
moves[mi++] = mvSlide(theta)
```

Then, we set-up the model by computing the conditional rate of the Poisson distribution, creating random variables for each observation and attaching (clamping) data to the variables.

```
for (i in 1:year.size() ) {
    rate[i] := ifelse(theta > year[i], beta, gamma)
    y[i] ~ dnPoisson(rate[i])
    y[i].clamp(observed_fatalities[i])
}
```

Finally, we create the model object from the variables, add some monitors and run the MCMC algorithm.

```
mymodel = model( theta )

monitors[1] = mnModel(filename="output/coal_accidents.log",printgen=10, separator = "
    ")
monitors[2] = mnScreen(printgen=10, eta, lambda, gamma, theta)

mymcmc = mcmc(mymodel, monitors, moves)

mymcmc.run(generations=3000)
```

2.3.2 Batch Mode

If you wish to run this exercise in batch mode, the files are provided for you.

You can carry out these batch commands by providing the file name when you execute the **rb** binary in your unix terminal (this will overwrite all of your existing run files).

- **$ rb RevBayes_scripts airline_fatalities_part1.Rev**
- **$ rb RevBayes_scripts airline_fatalities_part2.Rev**
- **$ rb RevBayes_scripts coalmine_accidents.Rev**

Bibliography

Gelman, A., J. Carlin, H. Stern, and D. Rubin. 2003. Bayesian data analysis. Chapman & Hall/CRC.

Höhna, S., T. A. Heath, B. Boussau, M. J. Landis, F. Ronquist, and J. P. Huelsenbeck. 2014. Probabilistic Graphical Model Representation in Phylogenetics. Systematic Biology 63:753–771.

Chapter 3

Reading and manipulating data

3.1 Overview

This tutorial describes the data formats that are used in `RevBayes`.

Requirements

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

- RB_Getting_Started

3.2 Molecular Sequence Data

3.2.1 Getting Started

- Download data files from: http://revbayes.github.io/tutorials.html
- Open the file **`primates_cytb.nex`** in your text editor. This file contains the nucleotide sequences of the cytochrome B gene sampled from 13 species (Box 1). The elements of the **`DATA`** block indicate the data type, number of taxa, and sequence length.

Box 1: A fragment of the NEXUS file containing the cytochrome B sequences for this exercise.

```
#NEXUS

Begin data;
Dimensions ntax=13 nchar=673;
Format datatype=DNA missing=? gap=-;
Matrix
Trig_excelsa
TCGAAACCTG...
Fagus_engleriana
TCGAAACCTG...
Fagus_crenata1
TCGAAACCTG...
Fagus_japonica2
TCGAAACCTG...
Fagus_japonica1
TCGAAACCTG...
Fagus_orientalis
TCGAAACCTG...
Fagus_sylvatica
TCGAAACCTG...
Fagus_lucida1
TCGAAACCTG...
Fagus_lucida2
TCGAAACCTG...
Fagus_crenata2
TCGAAACCTG...
Fagus_grandifolia
TCGAAACCTG...
Fagus_mexicana
TCGAAACCTG...
Fagus_longipetiolata
TCGAAACCTG...
       ;
End;
```

3.2.2 Loading Molecular Sequence Data

We can read the data into RevBayes using the **readDiscreteCharacterData()** function.

```
data <- readDiscreteCharacterData("data/primates_cytb.nex")
```

3.2.3 Querrying Dataset Attributes

When a dataset has been loaded into RevBayes, we can query relevant Rev variables. To report the current value of any variable, simply type the variable name and press enter. For example, the **data** variable returns general information about the sequence alignment:

```
data
   DNA character matrix with 23 taxa and 1141 characters
   ======================================================
   Origination:                  primates_cytb.nex
   Number of taxa:               23
   Number of included taxa:      23
   Number of characters:         1141
   Number of included characters: 1141
   Datatype:                     DNA
```

The **data** variable has *member functions* that we can use to retrieve specific attributes of the dataset. These member functions include the number of taxa (**data.ntaxa()**), the sequence length (**data.nchar()**), etc.

```
data.ntaxa()
   23
```

Available *member functions* for the data variable are listed in Table 3.1.

3.2.4 Concatenating Sequences

We can combine two or more datasets using the **concatenate** function. First, we will read in two datasets; the first is an alignment of primate cytb sequences, the second is an alignment of cox2 sequences:

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

Next, we will concatenate these two alignments using the **concatenate** function. This returns a single data matrix that combines the sequences of both gene regions.

Table 3.1: Available member functions for the data variable.

Function name	Type
chartype	String function ()
excludeAll	void function ()
excludeCharacter	void function (Natural)
excludeCharacter	void function (Natural [ ])
getEmpiricalBaseFrequencies	Simplex function ()
getNumInvariantSites	Natural function ()
includeAll	void function ()
includeCharacter	void function (Natural)
includeCharacter	void function (Natural [ ])
ishomologous	Bool function ()
methods	void function ()
names	String [ ] function ()
nchar	Natural function ()
ntaxa	Natural function ()
removeTaxa	void function (String)
removeTaxa	void function (String [ ])
setCodonPartition	void function (Natural)
setCodonPartition	void function (Natural [ ])
setNumStatesPartition	void function (Natural)
setTaxonName	void function (String current, String new)
show	void function ()
size	Natural function ()

```
data <- concatenate(data_cytb, data_cox2)
```

We can confirm this by querying the **data** variable:

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

3.2.5 Excluding/Including Taxa

We can exclude species from an alignment that is currently in memory using the **removeTaxa** function. For example, we could exclude the outgroup species *Saimiri sciureus* from our concatenated primate alignment

(**data**) by typing:

```
data.removeTaxa("Saimiri_sciureus")
```

We can then confirm the removal of a species by checking the number of remaining taxa:

```
data.ntaxa()
   22
```

The number of species has decreased by one, as expected. We can confirm that we have excluded *Saimiri sciureus* by typing:

```
data.names()
    [ "Callicebus_donacophilus", "Cebus_albifrons", "Alouatta_palliata", ...]
```

3.2.6 Excluding/Including Sites or Genes

We can exclude a single site (or set of sites) from an alignment that is currently in memory using the **excludeCharacter** function. For example, we could exclude the first site in our concatenated primate alignment (**data**) by typing:

```
excludeCharacter([1])
```

[Note that sites of an alignment are indexed from 1 to N.] We can confirm the removal of a site by checking the number of remaining sites:

```
data.nchar()
   1851
```

The number of sites has decreased by one, as expected. We can return the excluded site to our alignment using the **includeCharacter** function:

```
includeCharacter([1])
```

We can similarly exclude/include a range of sites, *e.g.*, corresponding to a gene region. Here, we will exclude all 1141 sites comprising the cytb gene region from our concatenated alignment:

```
data.excludeCharacter(1:1141)
```

We can check the number of remaining sites, which comprise the cox2 gene region:

```
data.nchar()
   711
```

We can easily return the excluded cytb sequences by typing:

```
data.includeCharacter(1:1141)
```

It is also possible to exclude/include all sites using the **excludeAll** and **includeAll** commands.

3.3 Biogeographical Data

For concreteness, this section focuses on the Hawaiian *Psychotria* dataset used in **?**.

3.3.1 Nexus file

The data file contains a matrix of binary characters corresponding to the observed ranges of the study taxa.

```
#NEXUS

begin data;
  dimensions ntax=19 nchar=4;
  format datatype=standard symbols = "01";
  matrix
    P_mariniana_Kokee2 1000
    P_mariniana_Oahu  0100
    ...
    P_hexandra_Oahu   0100
  ;
end;
```

Geographic range data is stored in standard Nexus format. In the `data` block, the first line gives the dimensions of the data matrix and the second line indicates we will be using binary characters. The four characters correspond to areas defined by the geography file (next subsection). Rows in the `matrix` block correspond to taxa and their geographic range data, while columns give in which areas each taxon is present (1) or absent (0). For example, `P_hexandra_Oahu` is endemic to area 2.

3.3.2 Atlas file

The atlas file describes the biogeographic state space as it might change over time.

```
{
  "name":"HawaiianArchipelago5my",
  "epochs": [
  {
    "name":"epoch1",
    "start_age":100.0,
    "end_age":3.7,
    "areas":
      [{ "name":"K","latitude":19.6,"longitude":-155.5,"dispersalValues":[1,0,0,0]},
       { "name":"O","latitude":19.6,"longitude":-155.5,"dispersalValues":[0,0,0,0]},
       { "name":"M","latitude":19.6,"longitude":-155.5,"dispersalValues":[0,0,0,0]},
       { "name":"H","latitude":19.6,"longitude":-155.5,"dispersalValues":[0,0,0,0]}]
  },
  {
    "name":"epoch2",
    ...
  },
  {
    "name":"epoch3",
    ...
  },
  {
    "name":"epoch4",
    "start_age":0.5,
    "end_age":0.0,
    "areas":
      [{"name":"K","latitude":22.1,"longitude":-159.5,"dispersalValues":[1,1,1,1]},
       {"name":"O","latitude":21.5,"longitude":-158.0,"dispersalValues":[1,1,1,1]},
       {"name":"M","latitude":20.8,"longitude":-156.3,"dispersalValues":[1,1,1,1]},
       {"name":"H","latitude":19.6,"longitude":-155.5,"dispersalValues":[1,1,1,1]}]
    }
  ]
}
```

The atlas stores information in JSON (JavaScript Object Notation) format. JSON is a lightweight format used to assign values to variables in a hierarchical manner. There are three main tiers to the hierarchy in the Atlas file: the atlas, the epoch, and the area. In the lowest tier, each area corresponds to a character in the model and is assigned it's own properties. In the middle tier, each epoch contains the set of homologous areas (characters) that may be part of a species' range, but importantly the properties of these areas may take on different values during different intervals of time, as given by the `start_age` and `end_age` variables. Because the tree and range evolution model also operate on units of geological time, the rates of area gain

and loss can condition on areas' properties as a function of time. Sometimes these models are called stratified models or epochal models. Finally, the atlas contains the array of epochs in the highest tier.

Each area is assigned a `latitude` and `longitude` to represent its geographical coordinates, ideally the centroid of the area. If a centroid does not represent the distance between areas, splitting the area into multiple smaller areas is reasonable. Here, the `latitude` and `longitude` change in each of the four epochs, where they begin at the current location of Hawaii and drift northwesterly until they reach their current positions.

In addition, each area is marked as habitable or not using the `dispersalValues` array. The elements in the array correspond to the other areas defined in the analysis. For example, in `epoch1`, Kauai's `dispersalValues` is equal to `[ 1,0,0,0 ]`, which indicates Kauai exists at that point in time but it is not in contact with any other areas, i.e. the range in that area cannot expand into other areas. The `dispersalValues` for Oahu, Maui, and Hawaii are all equal to `[ 0,0,0,0 ]`, meaning no species may be present in that area during the time interval of epoch1 during ages from 10.0 to 3.7. In contrast, `epoch4`, from ages 0.5 to the present, range expansions may occur between any pair of areas and any area may be included in a species' range.

Part II

Models of Molecular Evolution

Chapter 4

Continuous Time Markov Model for Discrete Character Evolution

4.1 Overview

This tutorial demonstrates how to set up and perform analyses using common nucleotide substitution models. The substitution models used in molecular evolution are continuous time Markov models, which are fully characterized by their instantaneous-rate matrix:

$$Q = \begin{pmatrix} -\mu_A & \mu_{GA} & \mu_{CA} & \mu_{TA} \\ \mu_{AG} & -\mu_G & \mu_{CG} & \mu_{TG} \\ \mu_{AC} & \mu_{GC} & -\mu_C & \mu_{TC} \\ \mu_{AT} & \mu_{GT} & \mu_{CT} & -\mu_T \end{pmatrix} ,$$

where μ_{ij} represents the instantaneous rate of substitution from state i to state j. Given the instantaneous-rate matrix, Q, we can compute the corresponding transition probabilities for a branch of length t, $P(t)$, by exponentiating the rate matrix:

$$P(t) = \begin{pmatrix} p_{AA}(t) & p_{GA}(t) & p_{CA}(t) & p_{TA}(t) \\ p_{AG}(t) & p_{GG}(t) & p_{CG}(t) & p_{TG}(t) \\ p_{AC}(t) & p_{GC}(t) & p_{CC}(t) & p_{TC}(t) \\ p_{AT}(t) & p_{GT}(t) & p_{CT}(t) & p_{TT}(t) \end{pmatrix} = e^{Qt} = \sum_{j=0}^{\infty} \frac{tQ^j}{j!} .$$

Each specific substitution model has a uniquely defined instantaneous-rate matrix, Q.

In this tutorial you will perform phylogeny inference under common models of DNA sequence evolution: JC, F81, HKY85, GTR, GTR+Gamma and GTR+Gamma+I. For all of these substitution models, you will perform an MCMC analysis to estimate phylogeny and other model parameters. For the sake of simplicity we will assume a simple birth-death tree prior and a known, fixed clock rate. All the assumptions will be covered more in detail later in this tutorial.

4.1.1 Requirements

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

- RB_Getting_Started
- RB_Basics_Tutorial

Note that the RB_Basics_Tutorial introduces the basic syntax of `Rev` but does not cover any phylogenetic models. You may skip the RB_Basics_Tutorial if you have some familiarity with `R`. We tried to keep this tutorial very basic and introduce all the language concepts on the way. You may only need the RB_Basics_Tutorial for a more in-depth discussion of concepts in `Rev`.

4.2 Data and files

We provide the data file(s) which we will use in this tutorial. You may want to use your own data instead. 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).

4.3 Example: Character Evolution under the Jukes-Cantor Substitution Model

4.3.1 Getting Started

The first section of this exercise involves: (1) setting up a Jukes-Cantor (JC) substitution model for an alignment of the cytochrome b subunit; (2) approximating the posterior probability of the tree topology and node ages (and all other parameters) using MCMC, and; (3) summarizing the MCMC output by computing the maximum *a posteriori* tree.

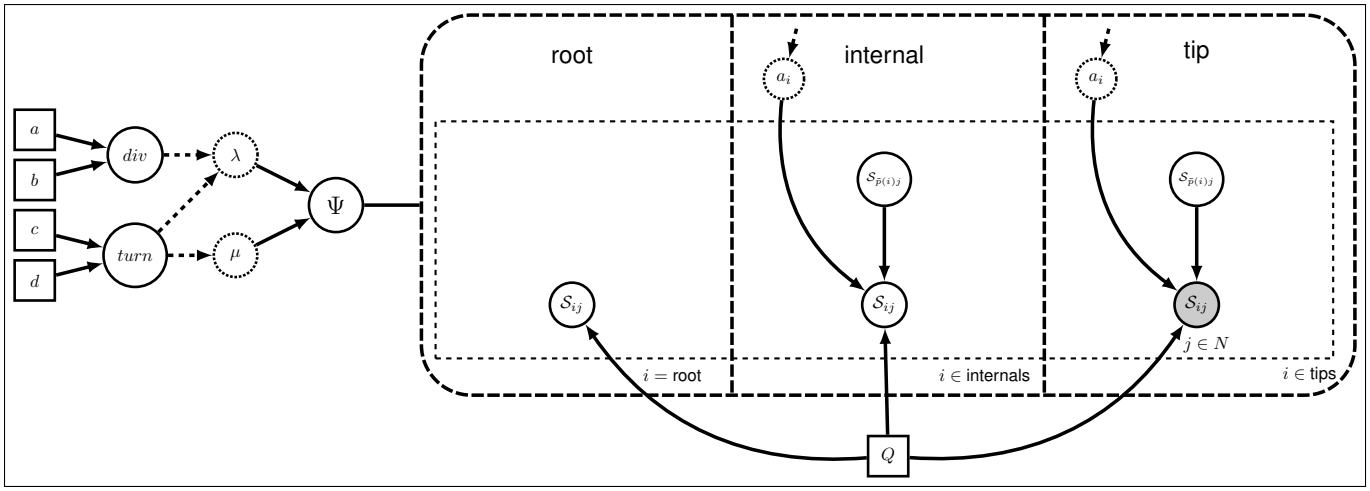


Figure 4.1: Graphical model representation of a simple phylogenetic model. The graphical model shows the dependencies between the parameters. Here, the rate matrix Q is a constant variable because it is fixed and does not depend on any parameters. The only free parameters of this model, the Jukes-Cantor model, are the tree Ψ including the node ages.

The general structure of the model is represented in Figure 7.1. This figure shows the full model graph. For simplicity and computational efficiency we will use a collapsed for of the same graphical model as shown in Figure 4.2

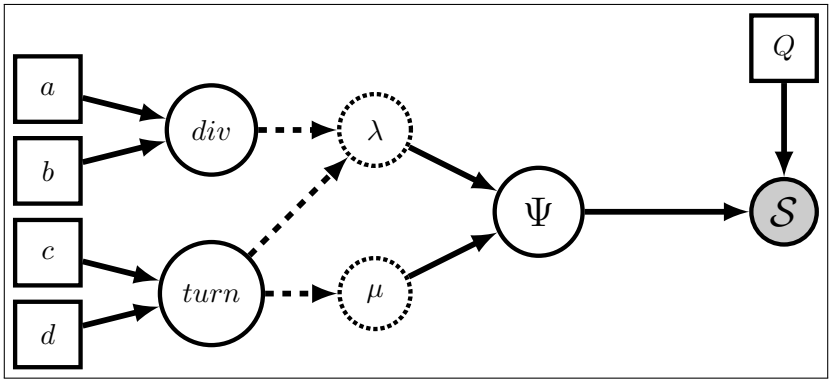


Figure 4.2: Graphical model representation of a simple phylogenetic model. The colapsed graphical model combining all the CTMC variable of the phylogeny. We still have the rate matrix Q as a constant variable and the stochastic variable for the tree Ψ.

We first consider the simplest substitution model described by Jukes and Cantor (1969). The instantaneous-rate matrix for the JC substitution model is defined as

$$Q_{JC69} = \begin{pmatrix} * & 1 & 1 & 1 \\ 1 & * & 1 & 1 \\ 1 & 1 & * & 1 \\ 1 & 1 & 1 & * \end{pmatrix},$$

which has the advantage that the transition probability matrix can be computed analytically:

$$P_{JC69} = \begin{pmatrix} \frac{1}{4} + \frac{3}{4}e^{-t\mu} & \frac{1}{4} - \frac{1}{4}e^{-t\mu} & \frac{1}{4} - \frac{1}{4}e^{-t\mu} & \frac{1}{4} - \frac{1}{4}e^{-t\mu} \\ \frac{1}{4} - \frac{1}{4}e^{-t\mu} & \frac{1}{4} + \frac{3}{4}e^{-t\mu} & \frac{1}{4} - \frac{1}{4}e^{-t\mu} & \frac{1}{4} - \frac{1}{4}e^{-t\mu} \\ \frac{1}{4} - \frac{1}{4}e^{-t\mu} & \frac{1}{4} - \frac{1}{4}e^{-t\mu} & \frac{1}{4} + \frac{3}{4}e^{-t\mu} & \frac{1}{4} - \frac{1}{4}e^{-t\mu} \\ \frac{1}{4} - \frac{1}{4}e^{-t\mu} & \frac{1}{4} - \frac{1}{4}e^{-t\mu} & \frac{1}{4} - \frac{1}{4}e^{-t\mu} & \frac{1}{4} + \frac{3}{4}e^{-t\mu} \end{pmatrix}.$$

In the later exercises you will be asked to specify more complex substitution models. **Don't be scared by the math!** `RevBayes` will take care of all the computations for you. Here we only provide some of the equations for the models in case you might be interested in the details. You will be able to complete the exercises without understanding the underlying math.

→ The files for this example analysis are provided for you, which can easily be run using the **`source()`** function in the `RevBayes` console:

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

If everything loaded properly, then you should see the program initiate the Markov chain Monte Carlo analysis that estimates 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.

Ultimately, this is how you will execute most analyses in `RevBayes`, with the full specification of the model and analyses contained in the sourced files. You could easily run this entire analysis on your own data by substituting your data file name for that in the model-specification file. However, it is important to understand the components of the model to be able to take full advantage of the flexibility and richness of `RevBayes`. Furthermore, without inspecting the `Rev` scripts sourced in **`JukesCantor.Rev`**, you may end up inadvertently performing inappropriate analyses on your dataset, which would be a waste of your time and CPU cycles. The next steps will walk you through the full specification of the model and MCMC analyses.

4.3.2 Loading the Data

→ Download data and output files (if you don't have them already) from: http://revbayes.github.io/tutorials.html

First load in the sequences using the **`readDiscreteCharacterData()`** function.

```
data <- readDiscreteCharacterData("data/primates_cytb.nex")
```

Executing these lines initializes the data matrix as the respective `Rev` variables. To report the current value of any variable, simply type the variable name and press enter. For the **data** matrix, this provides information about the alignment:

```
data
   DNA character matrix with 23 taxa and 1141 characters
   ====================================================
   Origination:                  primates_cytb.nex
   Number of taxa:               23
   Number of included taxa:      23
   Number of characters:         1141
   Number of included characters: 1141
   Datatype:                     DNA
```

Next we will specify some useful variables based on our dataset. The variable **data** has *member functions* that we can use to retrieve information about the dataset. These include the number of species (**n_species**) and the tip labels (**names**). Each of these variables will be necessary for setting up different parts of our model.

```
n_species <- data.ntaxa()
names <- data.names()
```

Additionally, we set up a counter variable for the number of moves that we already added to our analysis. [Recall that moves are algorithms used to propose new parameter values during the MCMC simulation.] This will make it much easier if we extend the model or analysis to include additional moves or to remove some moves.

```
mi = 0
```

You may have noticed that we used the **=** operator to create the move index. This simply means that the variable is not part of the model. You will later see that we use this operator more often, *e.g.*, when we create moves and monitors.

With the data loaded, we can now proceed to specify our Jukes-Cantor substitution model.

4.3.3 Jukes-Cantor Substitution Model

A given substitution model is defined by its corresponding instantaneous-rate matrix, Q. The Jukes-Cantor substitution model does not have any free parameters (as the substitution rates are all assumed to be equal),

so we can define it as a constant variable. The function **fnJC(n)** will create an instantaneous-rate matrix for character with n states. Since we use DNA data here, we create a 4x4 instantaneous-rate matrix:

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

You can see the rates of the Q matrix by typing

```
Q
   [ [ -1.0000, 0.3333, 0.3333, 0.3333 ] ,
     0.3333, -1.0000, 0.3333, 0.3333 ] ,
     0.3333, 0.3333, -1.0000, 0.3333 ] ,
     0.3333, 0.3333, 0.3333, -1.0000 ] ]
```

As you can see, all substitution rates are equal.

4.3.4 Tree Prior: Tree Topology and Node Ages

The tree (the topology and node ages) is a stochastic node in our phylogenetic model. In Figure 7.1, the tree is denoted Ψ.

We will assume a constant-rate birth-death process as the prior distribution on the tree. This means that all possible labeled, rooted tree topologies have equal probability. The distribution in `RevBayes` is **dnBDP()**. For the birth-death process we need a speciation rate and extinction rate parameter. Let us start with those two variables. We use a *gamma* distribution with rate $a = 5$ and shape $b = 1$ for both the **diversification** and **turnover** variables.

```
a <- 5
b <- 1
diversification ~ dnGamma(shape=a, rate=b)
c <- 5
d <- 1
turnover ~ dnGamma(shape=c,rate=d)
```

Now we can transform the **diversification** and **turnover** into the **speciation** rate and **extinction** rate.

```
speciation := diversification + turnover
extinction := turnover
```

We also need to specify a prior on the root age (our informed guess is about 75-80 mya). Nevertheless we use a very wide uniform distribution between 0 and 1000.

```
root ~ dnUniform(0.0,1000.0)
```

Additionally, we know that we do not have all primate species included in this data set. We only have 23 out of the approximately 450 primate species. Thus, we use a sampling fraction to represent this incomplete taxon sampling.

```
sampling_fraction <- 23 / 450
```

Here we have created our first three stochastic variables. For each one of them we need to create at least one moves that change the stochastic variables. In this case we use sliding window proposals but you could use scaling proposals for the rates too.

```
moves[++mi] = mvSlide(diversification,delta=1,tune=true,weight=1)
moves[++mi] = mvSlide(turnover,delta=1,tune=true,weight=1)
moves[++mi] = mvSlide(root,delta=1,tune=true,weight=1)
```

Next, specify the **tree** stochastic node by passing in the tip labels **names** to the **dnBDP()** distribution:

```
psi ~ dnBDP(lambda=speciation, mu=extinction, rootAge=abs(root), rho=sampling_fraction
    , nTaxa=n_species, names=names )
```

Some types of stochastic nodes can be updated by a number of alternative moves. Different moves may explore parameter space in different ways, and it is possible to use multiple different moves for a given parameter to improve mixing (the efficiency of the MCMC simulation). In the case of our rooted tree, for example, we can use both a nearest-neighbor interchange move without and with changing the node ages (**mvNarrow** and **mvNNI**) and a fixed-nodeheight subtree-prune and regrafting move (**mvFNPR**). We also need moves that change the ages of the internal nodes; which are for example the **mvSubtreeScale** and **mvNodeTimeSlideUniform**. These moves do not have tuning parameters associated with them, thus you only need to pass in the **psi** node and proposal **weight**.

```
moves[++mi] = mvNarrow(psi, weight=5.0)
moves[++mi] = mvNNI(psi, weight=1.0)
moves[++mi] = mvFNPR(psi, weight=3.0)
moves[++mi] = mvSubtreeScale(psi, weight=3.0)
moves[++mi] = mvNodeTimeSlideUniform(psi, weight=15.0)
```

The weight specifies how often the move will be applied either on average per iteration or relative to all other moves. Have a look at the MCMC tutorial for more details about moves and MCMC strategies: http://revbayes.github.io/tutorials.html

4.3.5 Putting it All Together

We have fully specified all of the parameters of our phylogenetic model—the tree topology with branch lengths, and the substitution model that describes how the sequence data evolved over the tree with branch lengths. Collectively, these parameters comprise a distribution called the *phylogenetic continuous-time Markov chain*, and we use the **PhyloCTMC** constructor function to create this node. This distribution requires several input arguments: (1) the **tree** with branch lengths; (2) the instantaneous-rate matrix **Q**; (3) the clock rate, and; (4) the **type** of character data.

For now we use an empirical estimate of the clock rate which is 0.01 (=1%) per million years per site.

```
clockRate <- 0.01
```

Build the random variable for the character data (sequence alignment).

```
# the sequence evolution model
seq ~ dnPhyloCTMC(tree=psi, Q=Q, branchRates=clockRate, type="DNA")
```

Once the **PhyloCTMC** model has been created, we can attach our sequence data to the tip nodes in the tree.

```
seq.clamp(data)
```

[Note that although we assume that our sequence data are random variables—they are realizations of our phylogenetic model—for the purposes of inference, we assume that the sequence data are "clamped".] When this function is called, **RevBayes** sets each of the stochastic nodes representing the tips of the tree to the corresponding nucleotide sequence in the alignment. This essentially tells the program that we have observed data for the sequences at the tips.

Finally, we wrap the entire model to provide convenient access to the DAG. To do this, we only need to give the **model()** function a single node. With this node, the **model()** function can find all of the other nodes by following the arrows in the graphical model:

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

Now we have specified a simple phylogenetic analysis—each parameter of the model will be estimated from every site in our alignment. If we inspect the contents of **mymodel** we can review all of the nodes in the DAG:

```
mymodel
```

4.3.6 Performing an MCMC Analysis Under the Jukes-Cantor Model

In this section, will describe how to set up the MCMC sampler and summarize the resulting posterior distribution of trees.

Specifying Monitors

For our MCMC analysis, we need to set up a vector of *monitors* to record 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[1] = mnModel(filename="output/primates_cytb_JC_posterior.log",printgen=10,
    separator = TAB)
```

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[2] = mnFile(filename="output/primates_cytb_JC_posterior.trees",printgen=10,
    separator = TAB, psi)
```

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

```
monitors[3] = mnScreen(printgen=1000, diversification, turnover)
```

Initializing and Running the MCMC Simulation

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 may wish to run the **.burnin()** member function. Recall that this function **does not** specify the number of states that we wish to discard from the MCMC analysis as burnin (i.e., the samples collected before the chain converges to the stationary distribution). Instead, the **.burnin()** function specifies a *completely separate* preliminary MCMC analysis that is used to tune the scale of the moves to improve mixing of the MCMC analysis.

```
mymcmc.burnin(generations=10000,tuningInterval=1000)
```

Now, run the MCMC:

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

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

Methods for visualizing the marginal densities of parameter values are not currently available in `RevBayes` itself. Thus, it is important to use programs like `Tracer` (Rambaut and Drummond 2011) to evaluate mixing and non-convergence. (`RevBayes` does, however, have a tool for convergence assessment called **`beca`**.)

→ Look at the file called **`output/primates_cytb_JC_posterior.log`** in `Tracer`.

4.3.7 Exercise 1

We are interested in the phylogenetic relationship of the Tarsiers. Therefore, we need to summarize the trees sampled from the posterior distribution. `RevBayes` can summarize the sampled trees by reading in the tree-trace file:

```
treetrace = readTreeTrace("output/primates_cytb_JC_posterior.trees",
                          treetype="clock")
treetrace.summarize()
```

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

```
mapTree(treetrace,"output/primates_cytb_JC.tree")
```

Fill in the following table as you go through the tutorial.

→ Look at the file called **`output/primates_cytb_JC.tree`** in `FigTree`.

Table 4.1: Posterior probabilities of phylogenetic relationship*.

Model	*Lemuroidea*	*Lorisoidea*	*Platyrrhini*	*Catarrhini*	*other*
Jukes-Cantor					
HKY85					
F81					
GTR					
GTR+Γ					
GTR+Γ+I					
Your model 1					
Your model 2					
Your model 3					

*you can edit this table

Table 4.2: Primate species and famaly relationships.

Species	**Family**	**Parvorder**	**Suborder**
Alouatta palliata	Atelidae	Platyrrhini (NWM)	Haplorrhini
Aotus trivirgatus	Aotidae	Platyrrhini (NWM)	Haplorrhini
Callicebus donacophilus	Pitheciidae	Platyrrhini (NWM)	Haplorrhini
Cebus albifrons	Cebidae	Platyrrhini (NWM)	Haplorrhini
Cheirogaleus major	Cheirogaleidae	Lemuroidea	Strepsirrhini
Chlorocebus aethiops	Cercopithecoidea	Catarrhini	Haplorrhini
Colobus guereza	Cercopithecoidea	Catarrhini	Haplorrhini
Daubentonia madagascariensis	Daubentoniidae	Lemuroidea	Strepsirrhini
Galago senegalensis	Galagidae	Lorisidae	Strepsirrhini
Hylobates lar	Hylobatidea	Catarrhini	Haplorrhini
Lemur catta	Lemuridae	Lemuroidea	Strepsirrhini
Lepilemur hubbardorum	Lepilemuridae	Lemuroidea	Strepsirrhini
Loris tardigradus	Lorisidae	Lorisidae	Strepsirrhini
Macaca mulatta	Cercopithecoidea	Catarrhini	Haplorrhini
Microcebus murinus	Cheirogaleidae	Lemuroidea	Strepsirrhini
Nycticebus coucang	Lorisidae	Lorisidae	Strepsirrhini
Otolemur crassicaudatus	Galagidae	Lorisidae	Strepsirrhini
Pan paniscus	Hominoidea	Catarrhini	Haplorrhini
Perodicticus potto	Lorisidae	Lorisidae	Strepsirrhini
Propithecus coquereli	Indriidae	Lemuroidea	Strepsirrhini
Saimiri sciureus	Cebidae	Platyrrhini (NWM)	Haplorrhini
Tarsius syrichta	Tarsiidae		Haplorrhini
Varecia variegata variegata	Lemuridae	Lemuroidea	Strepsirrhini

4.4 The Hasegawa-Kishino-Yano (HKY) 1985 Substitution Model

The Jukes-Cantor model assumes that all substitution rates are equal, which also implies that the stationary frequencies of the four nucleotide bases are equal. These assumptions are not very biologically reasonable, so we might wish to consider a more realistic substitution model that relaxes some of these assumptions. For example, we might allow stationary frequencies, π, to be unequal, and allow rates of transition and transversion substitutions to differ, κ. This corresponds to the substitution model proposed by Hasegawa et al. (1985; HKY), which is specified with the following instantaneous-rate matrix:

$$Q_{HKY} = \begin{pmatrix} \cdot & \pi_C & \kappa\pi_G & \pi_T \\ \pi_A & \cdot & \pi_C & \kappa\pi_T \\ \kappa\pi_A & \pi_C & \cdot & \pi_T \\ \pi_A & \kappa\pi_C & \pi_G & \cdot \end{pmatrix}.$$

[The diagonal $\cdot$ entries are equal to the negative sum of the elements in the corresponding row.]

→ Use the file **JukesCantor.Rev** as a starting point for the HKY analysis.

Note that we are adding two new variables to our model. We can define a variable **pi** for the stationary frequencies that are drawn from a flat Dirichlet distribution by

```
pi_prior <- v(1,1,1,1)
pi ~ dnDirichlet(pi_prior)
```

Since **pi** is a stochastic variable, we need to specify a move to propose updates to it. A good move on variables drawn from a Dirichlet distribution is the **mvSimplexElementScale**. This move randomly takes an element from the simplex, proposes a new value for it drawn from a Beta distribution, and then rescales all values of the simplex to sum to 1 again.

```
moves[++mi] = mvSimplexElementScale(pi)
```

The second new variable is κ, which specifies the ratio of transition-transversion rates. The κ parameter must be a positive-real number and a natural choice as the prior distribution is the lognormal distribution:

```
kappa ~ dnLnorm(0.0,1.25)
```

Again, we need to specify a move for this new stochastic variable. A simple scaling move should do the job.

```
moves[++mi] = mvScale(kappa)
```

Finally, we need to create the HKY instantaneous-rate matrix using the **fnHKY** function:

```
Q := fnHKY(kappa,pi)
```

This should be all for the HKY model.

→ Don't forget to change the output file names, otherwise your old analyses files will be overwritten.

4.4.1 Exercise 2

- Copy the file called **JukesCantor.Rev** and modify it by including the necessary parameters to specify the HKY substitution model.
- Run an MCMC analysis to estimate the posterior distribution under the HKY substitution model.
- Are the resulting estimates of the base frequencies equal? If not, how much do they differ? Are the estimated base frequencies similar to the empirical base frequencies? The empirical base frequencies are the frequencies of the characters in the alignment, which can be computed with **RevBayes** by **data.getEmpiricalBaseFrequencies()**.
- Is the inferred rate of transition substitutions higher than the rate of transversion substitutions? If so, by how much?
- Like the HKY model, the Felsenstein 1981 (F81) substitution model has unequal stationary frequencies, but it assumes equal transition-transversion rates (Felsenstein 1981). Can you set up the F81 model and run an analysis?
- Complete the table of the phylogenetic relationship of Tarsiers.

4.5 The General Time-Reversible (GTR) Substitution Model

The HKY substitution model can accommodate unequal base frequencies and different rates of transition and transversion substitutions. Despite these extensions, the HKY model may still be too simplistic for many real datasets. Here, we extend the HKY model to specify the General Time Reversible (GTR) substitution model (Tavaré 1986), which allows all six exchangeability rates to differ (Figure 4.3).

The instantaneous-rate matrix for the GTR substitution model is:

$$Q_{GTR} = \begin{pmatrix} \cdot & r_{AC}\pi_C & r_{AG}\pi_G & r_{AT}\pi_T \\ r_{AC}\pi_A & \cdot & r_{CG}\pi_G & r_{CT}\pi_T \\ r_{AC}\pi_A & r_{CG}\pi_C & \cdot & r_{GT}\pi_T \\ r_{AC}\pi_A & r_{CT}\pi_C & r_{GT}\pi_G & \cdot \end{pmatrix},$$

where the six exchangeability parameters, r_{ij}, specify the relative rates of change between states i and j.

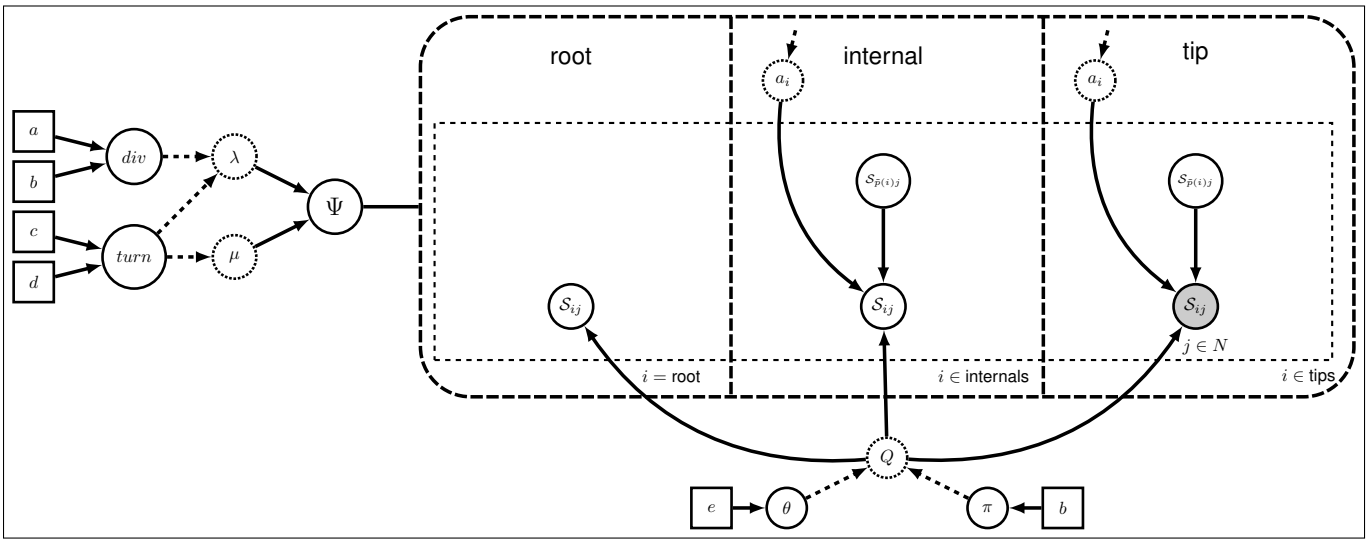


Figure 4.3: Graphical model representation of the General Time Reversible (GTR) phylogenetic model.

The GTR model requires that we define and specify a prior on the six exchangeability rates, which we will describe using a flat Dirichlet distribution. As we did previously for the Dirichlet prior on base frequencies, we first define a constant node specifying the vector of concentration-parameter values using the **v()** function:

```
er_prior <- v(1,1,1,1,1,1)
```

This node defines the concentration-parameter values of the Dirichlet prior distribution on the exchangeability rates. Now, we can create a stochastic node for the exchangeability rates using the **dnDirichlet()** function, which takes the vector of concentration-parameter values as an argument and the **~** operator. Together, these create a stochastic node named **er** (θ in Figure 4.3):

```
er ~ dnDirichlet(er_prior)
```

The Dirichlet distribution assigns probability densities to a group of parameters: e.g., those that measure proportions and must sum to 1. Here, we have specified a six-parameter Dirichlet prior, where each value describes one of the six relative rates of the GTR model: (1) $A \leftrightarrows C$; (2) $A \leftrightarrows G$; (3) $A \leftrightarrows T$; (4) $C \leftrightarrows G$; (5) $C \leftrightarrows T$; (6) $G \leftrightarrows T$. The input parameters of a Dirichlet distribution are called shape (or concentration) parameters. The expectation and variance for each variable are related to the sum of the shape parameters. The prior we specified above is a 'flat' or symmetric Dirichlet distribution; all of the shape parameters are equal (1,1,1,1,1,1). This describes a model that allows for equal rates of change between nucleotides, such that the expected rate for each is equal to $\frac{1}{6}$ (Figure 4.4a). We might also parameterize the Dirichlet distribution such that all of the shape parameters were equal to 100, which would also specify a prior with an expectation of equal exchangeability rates (Figure 4.4b). However, by increasing the values of the shape parameters, **`er_prior <- v(100,100,100,100,100,100)`**, the Dirichlet distribution will more strongly favor equal exchangeability rates; (*i.e.*, providing is a relatively *informative* prior). Alternatively, we might consider an asymmetric Dirichlet parameterization that could reflect a strong prior belief that transition and transversion substitutions occur at different rates. For example, we might specify the prior density **`er_prior <- v(4,8,4,4,8,4)`**. Under this model, the expected rate for transversions would be $\frac{4}{32}$ and that for transitions would be $\frac{8}{32}$, and there would be greater prior probability on sets of GTR rates that matched this configuration (Figure 4.4c). Yet another aymmetric prior could specify that each of the six GTR rates had a different value conforming to a Dirichlet(2,4,6,8,10,12). This would lead to a different prior probability density for each rate parameter (Figure 4.4d). Without strong prior knowledge about the pattern of relative rates, however, we can better reflect our uncertainty by using a vague prior on the GTR rates. Notably, all patterns of relative rates have the same probability density under **`er_prior <- v(1,1,1,1,1,1)`**.

For each stochastic node in our model, we must also specify a proposal mechanism if we wish to estimate that parameter. The Dirichlet prior on our parameter **`er`** creates a *simplex* of values that sum to 1.

```
moves[++mi] = mvSimplexElementScale(er)
```

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:

```
pi_prior <- v(1,1,1,1)
pi ~ dnDirichlet(pi_prior)
```

The node **`pi`** represents the π node in Figure 4.3. Now add the simplex scale move on the stationary frequencies to the moves vector:

```
moves[++mi] = mvSimplexElementScale(pi)
```

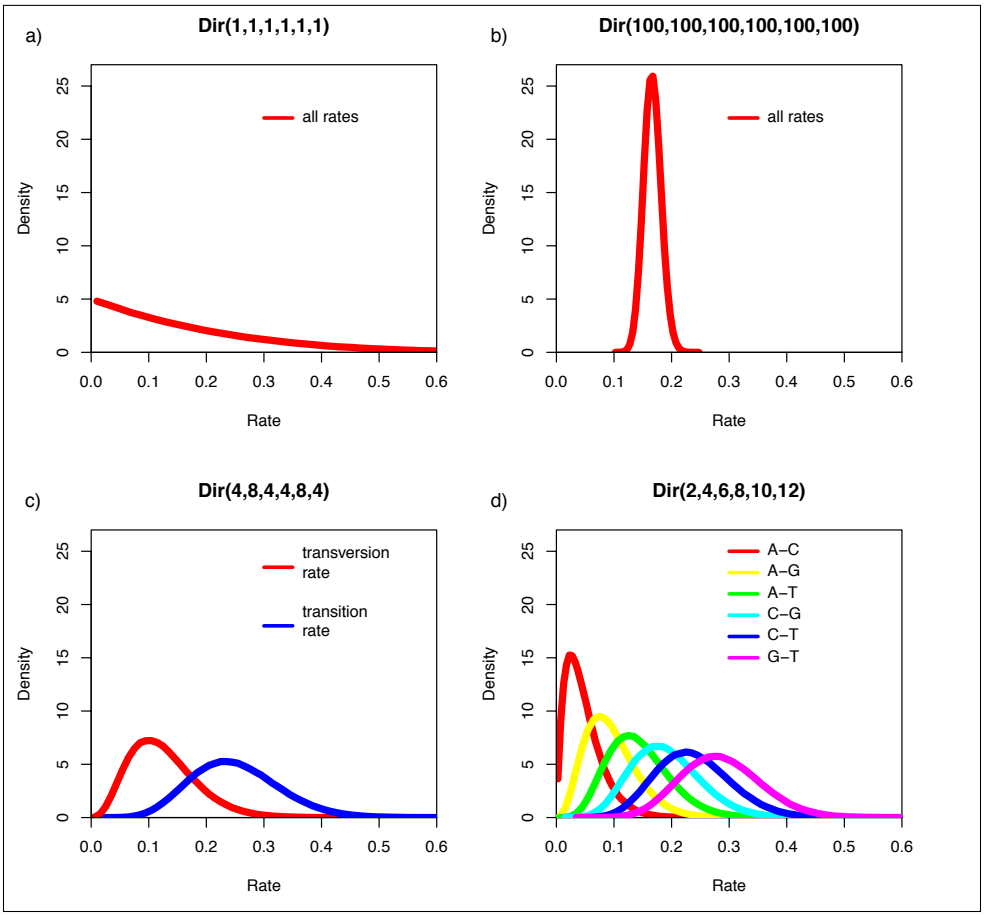


Figure 4.4: Four different examples of Dirichlet priors on exchangeability rates.

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

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

4.5.1 Execise 3

- Use one of your previous analysis files—either the **JukesCantor.Rev** or **HKY.Rev**—to specify a GTR analysis in a new file called **GTR.Rev**. Adapt the old analysis to be performed under the GTR substitution model.
- Run an MCMC analysis to estimate the posterior distribution.
- Complete the table of the phylogenetic relationship of Tarsiers.

4.6 The Discrete Gamma Model of Among Site Rate Variation

Members of the GTR family of substitution models assume that rates are homogeneous across sites, an assumption that is often violated by real data. We can accommodate variation in substitution rate among sites (ASRV) by adopting the discrete-gamma model (Yang 1994). This model assumes that the substitution rate at each site is a random variable that is described by a discretized 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, $\alpha = \beta$. This means that we can fully describe the gamma distribution with the single shape parameter, α. The degree of among-site substitution rate variation is inversely proportional to the value of the α-shape parameter. As the value of the α-shape increases, the gamma distribution increasingly resembles a normal distribution with decreasing variance, which therefore corresponds to decreasing levels of ASRV (Figure 5.2). By contrast, when the value of the α-shape parameter is < 1, the gamma distribution assumes a concave distribution that concentrates most of the prior density on low rates, but retains some prior mass on sites with very high rates, which therefore corresponds to high levels of ASRV (Figure 5.2). Note that, when $\alpha = 1$, the gamma distribution collapses to an exponential distribution with a rate parameter equal to β.

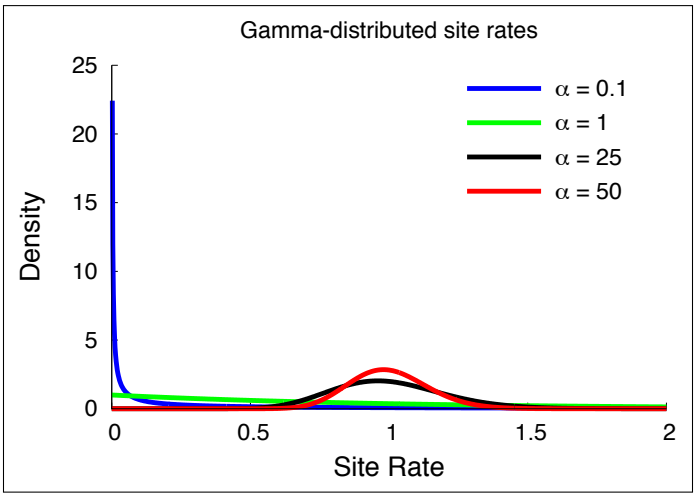


Figure 4.5: The probability density of mean-one gamma-distributed rates for different values of the α-shape parameter.

We typically lack prior knowledge regarding the degree of ASRV for a given alignment. Accordingly, rather than specifying a precise value of α, we can instead estimate the value of the α-shape parameter from the data. This requires that we specify a diffuse (relatively 'uninformative') prior on the α-shape parameter. For this analysis, we will use an exponential distribution with a rate parameter, **`shape_prior`**, equal to `0.05`. An exponential prior assigns non-zero probability on values of α ranging from 0 to ∞. The rate parameter of an exponential distribution, often denoted λ, controls both the mean and variance of this distribution, such that the expected (or mean) value of α is: $\mathbb{E}[\alpha] = \frac{1}{\lambda}$. Thus, if we set $\lambda = 0.05$, then $\mathbb{E}[\alpha] = 20$.

This approach for accommodating ASRV is another example of a hierarchical model (Figure 4.6). That is, variation in substitution rates across sites is addressed by applyig a site-specific rate multiplier to each of the j sites, r_j. These rate-multipliers are drawn from a discrete, mean-one gamma distribution; the shape of this prior distribution (and the corresponding degree of ASRV) is governed by the α-shape parameter.

The α-shape parameter, in turn, is treated as an exponentially distributed random variable. Finally, the shape of the exponential prior is governed by the rate parameter, λ, which is set to a fixed value.

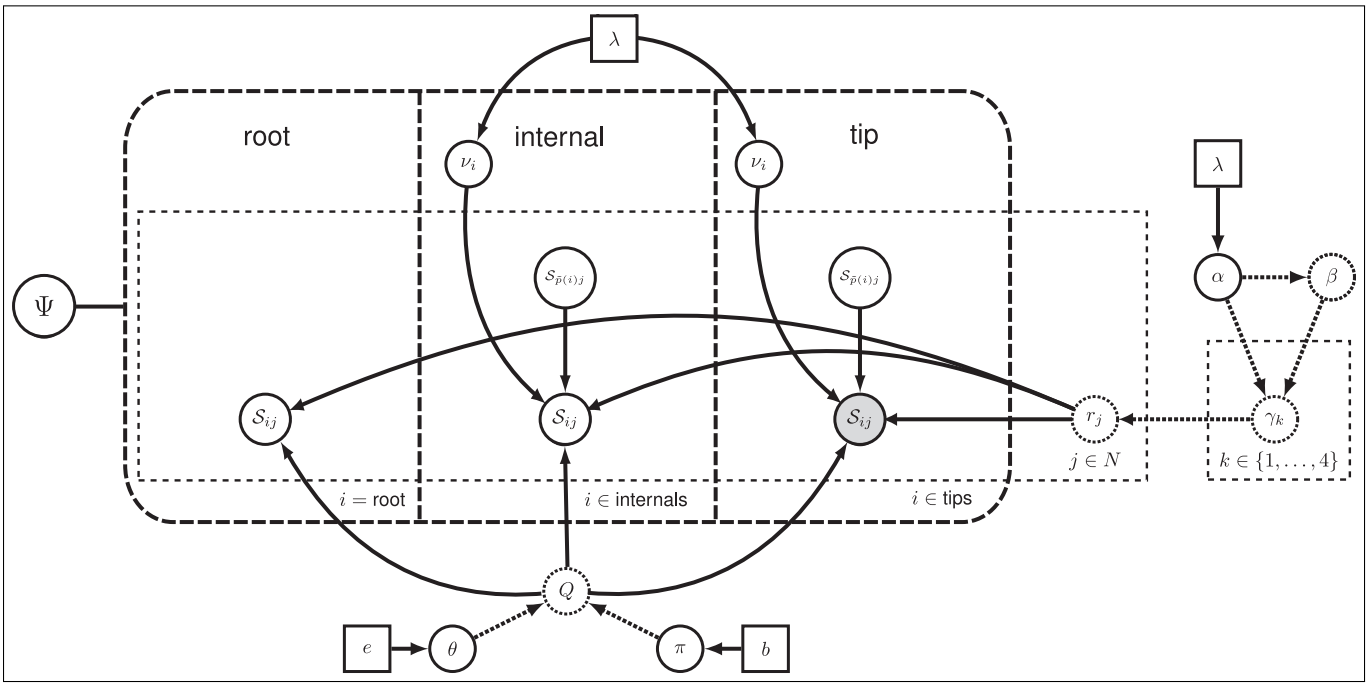


Figure 4.6: Graphical model representation of the General Time Reversible (GTR) + Gamma phylogenetic model.

4.6.1 Setting up the Gamma Model in RevBayes

Create a constant node called **shape_prior** for the rate parameter of the exponential prior on the gamma-shape parameter (this is represented as the constant λ-rate parameter in Figure 4.6):

```
shape_prior <- 0.05
```

Then create a stochastic node called **alpha** with an exponential prior (this represents the stochastic node for the α-shape parameter in Figure 4.6):

```
alpha ~ dnExponential(shape_prior)
```

The way the ASRV model is implemented involves discretizing the mean-one gamma distribution into a set number of rate categories, k. Thus, we can analytically marginalize over the uncertainty in the rate at each site. The likelihood of each site is averaged over the k rate categories, where the rate multiplier is the mean (or median) of each of the discrete k categories. To specify this, we need a deterministic node that is a vector that will hold the set of k rates drawn from the gamma distribution with k 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 **alpha** for both the shape and rate.

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

```
gamma_rates := fnDiscretizeGamma( alpha, alpha, 4 )
```

Note that here, by convention, we set $k = 4$. The random variable that controls the rate variation is the stochastic node **alpha**. We will apply a simple scale move to this parameter.

```
moves[++mi] = mvScale(alpha, weight=2.0)
```

Remember that you need to call the **PhyloCTMC** constructor to include the new site-rate parameter:

```
seq ~ dnPhyloCTMC(tree=psi, Q=Q, branchRates=clockRate, siteRates=gamma_rates, type="
    DNA")
```

4.6.2 Execise 4

Modify the previous GTR analysis to specify the GTR+Gamma model. Run an MCMC simulation to estimate the posterior distribution.

- Is there an impact on the estimated phylogeny compared with the previous analyses? Look at the MAP tree and the posterior probabilities of the clades.
- What is the estimated tree length? Is the estimate different to the previous analysis? What could cause this?
- Complete the table of the phylogenetic relationship of Tarsiers.

4.7 Modeling Invariable Sites

All of the substitution models described so far assume that the sequence data are potentially variable. That is, we assume that the sequence data are random variables; specifically, we assume that they are realizations of the specified **PhyloCTMC** distribution. However, some sites may not be free to vary—when the substitution rate of a site is zero, it is said to be *invariable*. Invariable sites are often confused with *invariant* sites—when each species exhibits the same state, it is said to be invariant. The concepts are related but distinct. If a site is truly invariable, it will necessarily give rise to an invariant site pattern, as such sites will always have a zero substitution rate. However, an invariant site pattern may be achieved via multiple substitutions that happen to end in the same state for every species.

Here we describe an extension to our phylogenetic model to accommodate invariable sites. Under the invariable-sites model (Hasegawa et al. 1985), each site is invariable with probability **pinvar**, and variable with probability 1−**pinvar**.

First, let's have a look at the data and see how many invariant sites we have:

```
data.getNumInvariantSites()
```

There seem to be a substantial number of invariant sites.

Now let's specify the invariable-sites model in **RevBayes**. We need to specify the prior probability that a site is invariable. A Beta distribution is a common choice for parameters representing probabilities.

```
pinvar ~ dnBeta(1,1)
```

The **Beta(1,1)** distribution is a flat prior distribution that specifies equal probability for all values between 0 and 1.

Then, as usual, we add a move to change this stochastic variable; we'll used a simple sliding window move.

```
moves[++mi] = mvSlide(pinvar)
```

Finally, that you need to call the **PhyloCTMC** constructor to include the new**pinvar** parameter:

```
seq ~ dnPhyloCTMC(tree=psi, Q=Q, branchRates=clockRate, siteRates=gamma_rates, pInv=
    pinvar, type="DNA")
```

4.7.1 Exercise 5

- Extend the GTR model to account for invariable sites and run an analysis.

- What is the estimated probability of invariable sites and how does it relate to the ratio of invariant sites to the total number of sites?
- Extend the GTR+Γ model to account for invariable sites and run an analysis.
- What is the estimated probability of invariable sites now?
- Complete the table of the phylogenetic relationship of Tarsiers.

Bibliography

Felsenstein, J. 1981. Evolutionary trees from DNA sequences: A maximum likelihood approach. Journal of Molecular Evolution 17:368–376.

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

Jukes, T. and C. Cantor. 1969. Evolution of protein molecules. Mammalian Protein Metabolism 3:21–132.

Rambaut, A. and A. J. Drummond. 2011. Tracer v1.5. http://tree.bio.ed.ac.uk/software/tracer/.

Tavaré, S. 1986. Some probabilistic and statistical problems in the analysis of DNA sequences. Some Mathematical Questions in Biologyï£¡DNA Sequence Analysis 17:57–86.

Yang, Z. 1994. Maximum likelihood phylogenetic estimation from DNA sequences with variable rates over sites: Approximate methods. Journal of Molecular Evolution 39:306–314.

Chapter 5

Partitioned Data Analysis

5.1 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.

5.1.1 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.

5.2 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.

5.3 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 (Brown and Lemmon 2007).

Accounting for process heterogeneity involves adopting a 'mixed-model' approach, (Ronquist and Huelsenbeck 2003) 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 5.1. 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.*, Suchard et al. 2001), 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 (Brandley et al. 2005; Nylander et al. 2004; McGuire et al. 2007). The analysis pipeline that we will use in this tutorial is depicted in Figure 5.1.

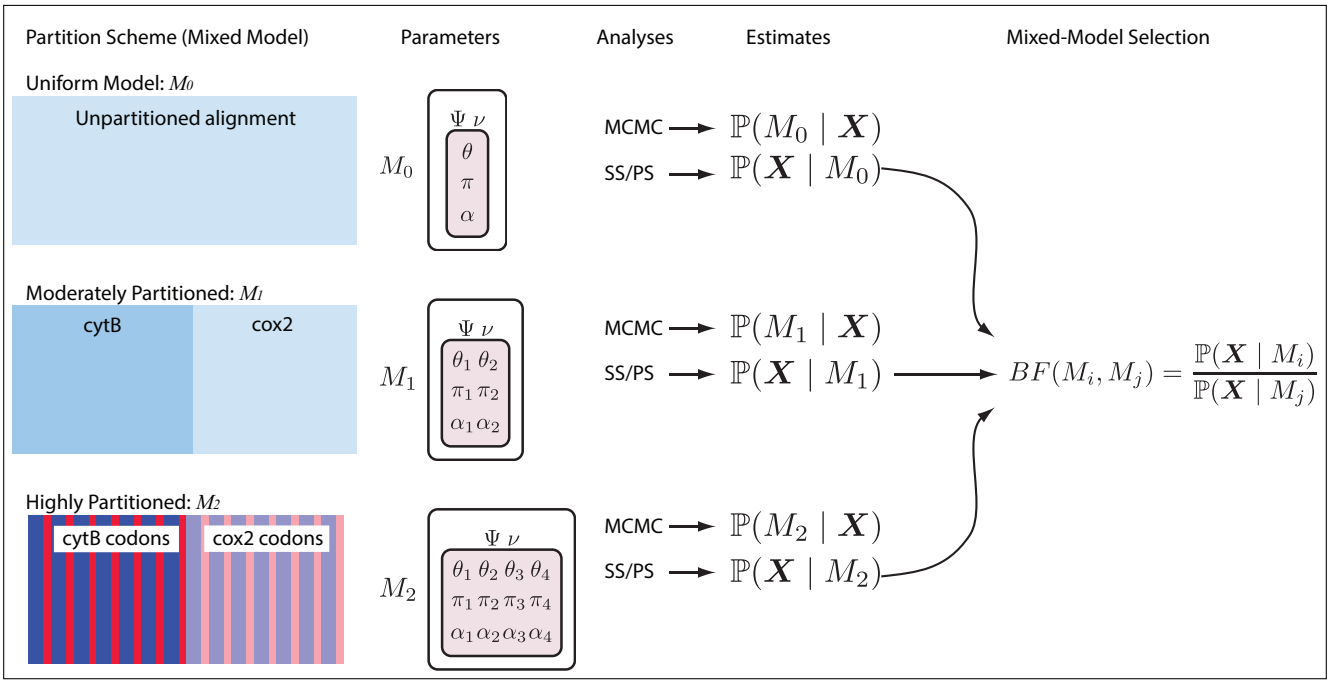


Figure 5.1: The analysis pipeline for Exercise 1. We will explore three partition schemes for the primates dataset. The first model (the 'uniform model', M_0) assumes that all sites evolved under a common GTR+Γ substitution model. The second model (the 'moderately partitioned' model, M_1) invokes two 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.

5.4 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=3)
```

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:

```
pi_prior <- v(1,1,1,1)
pi ~ dnDirichlet( pi_prior )
```

The node **pi** represents the π node in Figure 5.1. Now add the simplex scale move on the stationary frequencies to the moves vector

```
moves[mvi++] = mvSimplexElementScale(pi, alpha=10, tune=true, weight=2)
```

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,pi)
```

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 5.2). 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 5.2).

Alternatively, we might not have good prior knowledge about the variance in site rates, thus we can place an uninformative prior on the shape parameter. For this analysis, we will use an uniform distribution for values 0.1 to 50.0 to reflect our ignorance of what shape we expect.

Create a constant nodes called **alpha_prior_min** and **alpha_prior_max** to specify the interval for the uniform prior on the gamma-shape parameter

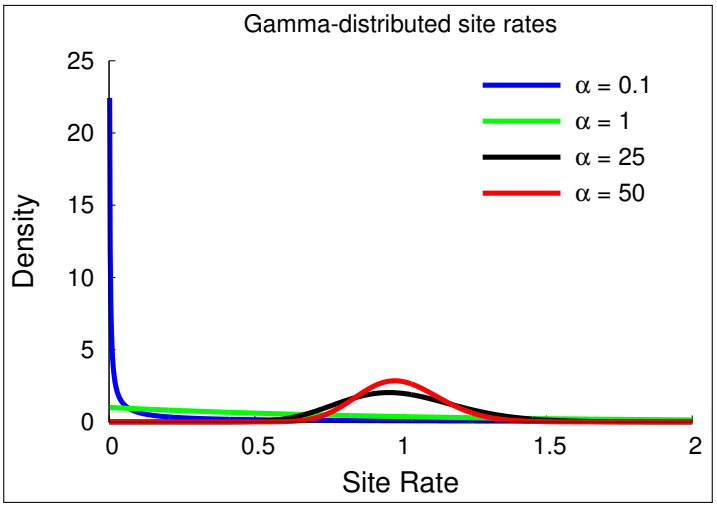


Figure 5.2: The probability density of mean-one gamma-distributed rates under different shape parameters.

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

Then create a stochastic node called **alpha** to represent the α node in Figure 5.1, 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 **alpha** 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 **alpha**. 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 5.1, 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, pi, 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 (Rambaut and Drummond 2011) 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/PS0.tre")
treetrace.summarize()
```

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

```
mapTree(treetrace,"output/PS0_map.tre")
```

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

5.5 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 parameterster 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
```

5.5.1 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 **alpha** nodes where the stochastic node for the first partition (cytB) will be **alpha[1]** and the stochastic node for the second partition (cox2) will be called **alpha[2]**.

The script for the model, `RevBayes_scripts/model_PS1.Rev`, creates the model parameters for each partition in one large loop. Here, we will split the loop into smaller parts to achieve the same end.

First, we will create the GTR rate matrix for partition i by first creating exchangeability rates

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

and stationary frequencies

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

then passing those parameters into a rate matrix function

```
for (i in 1:n_parts) {
  Q[i] := fnGTR(er[i],pi[i])
}
```

which states the rate matrix (`Q[i]`) for partition i is determined by the exchangeability rates (`er[i]`) and stationary frequencies (`pi[i]`) also defined for partition i. Following this format, we construct the remainin partition parameters: the +Γ mixture model

```
for (i in 1:n_parts) {
  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 $+I$ invariant sites model

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

and the per-partition molecular clock

```
for (i in 1:n_parts) {
  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++] = mnFile(er[1], pi[1], alpha[1], norm_gamma_rates[1], pinvar[1],
    part_rate_mult[1], er[2], pi[2], alpha[2], norm_gamma_rates[2], pinvar[2],
    part_rate_mult[2], speciation, extinction, 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")
```

5.6 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.

The script for the model, `RevBayes_scripts/model_PS2.Rev`, creates the model parameters for each partition in one large loop. Like before, we will split the loop into smaller parts to achieve the same end.

First, we will create the GTR rate matrix for partition i by first creating exchangeability rates

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

and stationary frequencies

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

then passing those parameters into a rate matrix function

```
for (i in 1:n_parts) {
  Q[i] := fnGTR(er[i],pi[i])
}
```

which states the rate matrix (`Q[i]`) for partition i is determined by the exchangeability rates (`er[i]`) and stationary frequencies (`pi[i]`) also defined for partition i. Following this format, we construct the remainin partition parameters: the $+\Gamma$ mixture model

```
for (i in 1:n_parts) {
  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 $+I$ invariant sites model

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

and the per-partition molecular clock

```
for (i in 1:n_parts) {
  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 mu 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++] = mnFile(er[1], pi[1], alpha[1], norm_gamma_rates[1], pinvar[1],
    part_rate_mult[1], er[2], pi[2], alpha[2], norm_gamma_rates[2], pinvar[2],
    part_rate_mult[2], er[3], pi[3], alpha[3], norm_gamma_rates[3], pinvar[3],
    part_rate_mult[3], er[4], pi[4], alpha[4], norm_gamma_rates[4], pinvar[4],
    part_rate_mult[4], speciation, extinction, 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(100000) # run for fewer iterations for classroom exercises
```

And, when MCMC completes, compute the MAP tree:

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

5.6.1 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 the marginal likelihood tutorial (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. Additionally, remove the line creating `mnScreen`, which will conflict with output produced by the power posterior analysis.

  Instead, configure and run a power posterior analysis

```
pow_p = powerPosterior(mymodel, moves, monitors, "output/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="output/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.

Bibliography

Brandley, M. C., A. Schmitz, and T. W. Reeder. 2005. Partitioned bayesian analyses, partition choice, and the phylogenetic relationships of scincid lizards. Systematic Biology 54:373–390.

Brown, J. M. and A. R. Lemmon. 2007. The importance of data partitioning and the utility of Bayes factors in Bayesian phylogenetics. Systematic Biology 56:643–655.

McGuire, J. A., C. C. Witt, D. L. Altshuler, and J. Remsen. 2007. Phylogenetic systematics and biogeography of hummingbirds: Bayesian and maximum likelihood analyses of partitioned data and selection of an appropriate partitioning strategy. Systematic Biology 56:837–856.

Nylander, J. A., F. Ronquist, J. P. Huelsenbeck, and J. Nieves-Aldrey. 2004. Bayesian phylogenetic analysis of combined data. Systematic Biology 53:47–67.

Rambaut, A. and A. J. Drummond. 2011. Tracer v1.5. http://tree.bio.ed.ac.uk/software/tracer/.

Ronquist, F. and J. Huelsenbeck. 2003. MrBayes 3: Bayesian phylogenetic inference under mixed models. Bioinformatics 19:1572–1574.

Suchard, M. A., R. E. Weiss, and J. S. Sinsheimer. 2001. Bayesian selection of continuous-time Markov chain evolutionary models. Molecular Biology and Evolution 18:1001–1013.

Part III

Inference

Chapter 6

Markov chain Monte Carlo Algorithms

This tutorial is currently under construction/revision.

6.1 Overview

This tutorial demonstrates how to diagnose the performance of MCMC simulations, such as the output from `RevBayes`. You will create a phylogenetic model for the evolution of DNA sequences under a JC, HKY85, GTR, GTR+Gamma and GTR+Gamma+I substitution model. For all these models you will perform an MCMC run to estimate phylogeny and other model parameters.

Requirements

We assume that you have completed the following tutorials:

- RB_MCMC_Tutorial

6.2 Exercise: Assessing Performance of MCMC Simulations

"*You can never be absolutely certain that the MCMC is reliable, you can only identify when something has gone wrong.*" Andrew Gelman

Model-based inference is, after all, based on the model. Careful research means being vigilant both regarding the choice of model and rigorously assessing our ability to estimate under the chosen model. The first issue—model specification, which actually entails three closely related issues—is critically important for the simple reason that unbiased estimates can only be obtained under a model that provides a reasonable description of the process that gave rise to our data. *Model selection* entails assessing the *relative fit* of our dataset to a pool of candidate models. Rankings are based on model-selection methods that compare the relative fit of candidate modes based either on their maximum-likelihood estimates (which measures the fit of the data to the model at a single point in parameter space), or on marginal likelihood of the candidate models (which measures the average fit of the candidate models to the data). *Model adequacy*—an equally important but relatively neglected issue—assesses the absolute fit of the data to the model. *Model uncertainty* is related to the common (and commonly ignored) scenario when multiple candidate models provide a similar fit to the data: in this scenario, conditioning on *any single* model (even the best) will lead to biased estimates, and so model averaging is required to accommodate uncertainty in the choice of model.

Much less concern is given to the second aspect of model-based inference: the ability to obtain reliable estimates under the chosen model(s). The implicit assumption, it seems, is that if a model is implemented correctly, and if that implementation is successfully used to obtain an estimate from a given dataset, then we must have performed valid inference under the model. This would be perfectly sound reasoning if inferences were based on analytical methods. Owing to the complexity of the models, however, it is not possible to estimate phylogenetic parameters analytically. Instead, parameter estimates are based on numerical methods. In the case of maximum-likelihood estimation, these are typically hill-climbing algorithms that attempt to search the profile likelihood to identify the vector of point estimates of all phylogenetic model parameters that jointly maximize the likelihood of observing the data under the model. The reliability of these algorithms can (and should) be assessed by comparing estimates obtained from repeated analyses that are initiated from random points in parameter space. Because there is only one *maximum* likelihood estimate, the terminal values estimated by replicate runs should be identical (within the precision of computer memory).

In the Bayesian statistical framework, inferences focus on the joint posterior probability density of phylogenetic model parameters, which is approximated by Markov chain Monte Carlo (MCMC) algorithms. It may be comforting to know that, in theory, an *appropriately constructed* and *adequately run* MCMC simulation is guaranteed to provide an arbitrarily precise description of the joint posterior probability density. In practice, however, even a given MCMC algorithm that provides reliable estimates in *most* cases will nevertheless fail in *some* cases and is not guaranteed to work for any given dataset. This raises an obvious question: "When do we know that an MCMC algorithm provides reliable estimates for a given empirical analyses". The answer is simple: *Never*.

Fortunately, this problem is not unique to the field of phylogenetics. Much of Bayesian inference outside our field also relies on MCMC algorithms to approximate the joint posterior probability density of parameters: similar concerns regarding the reliability of those inferences has motivated the development of a suite of diagnostic tools to assess MCMC performance. The trick is learning how to use these tools effectively and rigorously, especially for analyses that entail complex phylogenetic models and/or large datasets.

6.2.1 MCMC Basics

The ability to rigorously diagnose MCMC performance requires familiarity with some basic concepts from probability theory (discussed last time) and a strong intuitive understanding of the underlying mechanics—we need to know how the algorithms work in order to understand when they are not working. In this installment we'll briefly cover the mechanics of the Metropolis Hastings MCMC algorithm.

Recall that Bayesian inference is focused on the posterior probability density of parameters. The posterior probability of the parameters can, in principle, be solved using Bayes' theorem. However, (most) phylogenetic problems cannot be solved analytically, owing mainly to the denominator of Bayes' theorem—the marginal likelihood requires solving multiple integrals (for all of the continuous parameters, such as branch lengths, substitution rates, stationary frequencies, etc.) for each tree, and summing over all trees.

Accordingly, Bayesian inference of phylogeny typically resorts to numerical methods that approximate the posterior probability density. There are many flavors of Markov chain Monte Carlo (MCMC) algorithms—Gibbs samplers, Metropolis-coupled and reversible-jump MCMC, etc.—we will consider the Metropolis Hastings (MH) algorithm because it is commonly used for phylogenetic problems, and because it is similar to many other variants (which we will cover elsewhere). Note that MCMC and Bayesian inference are distinct animals: they have a relationship similar to that between 'optimization algorithms' and âĂŸmaximum-likelihood estimation.' Some Bayesian inference can be accomplished without MCMC algorithms, and MCMC algorithms can be used to solve problems in non-Bayesian statistical frameworks.

To introduce the MH algorithm, we will imagine a robot that is programed to explore an area. Specifically, the goal of our robot is to generate a topographic map of an unknown terrain. This terrain has a total surface area of one hectare. We deploy our robot by parachute at a random location within the terrain. We have programmed the robot with three simple rules:

1. If a proposed step will take the robot uphill, it automatically takes the proposed step.
2. If a proposed step will take the robot downhill, it divides the elevation of the proposed location by the elevation of the current location: we call this quotient R. It then generates a uniform random number, $U[0, 1]$. If $U < R$, the robot takes the proposed step; otherwise, it stays put.
3. The distribution for proposing steps is symmetrical. That is, the probability of proposing a step (but not necessarily accepting a proposed step) from point A to point B is equal to the probability of proposing a step from point B to point A.

We allow our little robot to wander through the terrain following these three simple rules. At prescribed intervals (*e.g.*, every 10 proposed steps), the robot records the details of his position (his elevation, latitude, longitude, etc.) in a log. If we allow our robot to wander long enough, his log is guaranteed to provide an arbitrarily precise description of the topography of the terrain.

Let's consider a slightly more formal description of the Metropolis-Hastings MCMC algorithm. First some preliminaries. This algorithm entails simulating a Markov chain that has a stationary distribution that is the joint posterior probability density of phylogenetic model parameters. The 'state' of the chain is a set of parameter values that fully specify phylogenetic model: *e.g.*, a tree topology, a vector of branch lengths, and a set of parameters specifying the stochastic model of trait change (*e.g.*, for the GTR + Γ substitution model, this comprises a vector of values for the four stationary frequencies, a vector of values

for the six exchangeability parameters, and the value of the alpha parameter describing the shape of the gamma-distributed among-site rate variation). Under the robot analogy, the state of the chain corresponds to a unique point in the terrain. The Markov property of the MCMC reflects the fact that the next state of the chain only depends on the current state of the chain, but not on previous states—that is, the past affects the future only through the present. The Monte Carlo aspect of the MCMC reflects the fact that it is a simulation: it is a numerical method that relies on repeated random sampling.

Now, on to the MH algorithm:

1. Initialize the chain with values for all parameters, including the tree topology, τ, the vector of branch lengths ν, and substitution model parameters π_i, q_i, α. We will call the set of model parameters θ. The initial parameter values might be specified arbitrarily, or might be drawn from the corresponding prior probability density for each parameter.

2. Select a single parameter (or set of parameters) to alter according to its proposal probability. For example, here are the default proposal probabilities used by `MrBayes`:

```
The MCMC sampler will use the following moves:
   With prob. Chain will use move
      1.00 %  Dirichlet(Revmat)
      1.00 %  Slider(Revmat)
      1.00 %  Dirichlet(Pi)
      1.00 %  Slider(Pi)
      2.00 %  Multiplier(Alpha)
     10.00 %  ExtSPR(Tau,V)
     10.00 %  ExtTBR(Tau,V)
     10.00 %  NNI(Tau,V)
     10.00 %  ParsSPR(Tau,V)
     40.00 %  Multiplier(V)
     10.00 %  Nodeslider(V)
      4.00 %  TLMultiplier(V)
```

   We can see that $\sim 1\%$ of the time (*i.e.*, with probability ~ 0.01), the MCMC will propose changes to the exchangeability ('revmat') and stationary frequency ('pi') parameters using a 'Dirichlet' proposal mechanism, and an equal effort proposing changes to the same parameters using something called a 'Slider' proposal mechanism. Conversely, $\sim 10\%$ of the time (*i.e.*, with probability ~ 0.1), the MCMC will propose changes to the tree and branch lengths ('Tau and V') using the 'extending SPR' proposal mechanism, and with equal probability using the 'extending TBR', 'extending NNI' and the 'parsimony SPR' proposal mechanisms.

   For the moment, we won't worry about the details of these proposal mechanisms—they basically involve different ways of 'poking' the current parameter value, θ, to generate a new (proposed) parameter value, θ'. The important question at the moment is: Where do these proposal probabilities come from? The answer is: experience. We want to design the MCMC such that it invests effort in a given parameter in proportion to the difficulty of approximating that parameter. Note that the MCMC summarized above will spend $\sim 2\%$ of its time proposing changes to the exchangeability and stationary frequency parameters, but will invest $\sim 40\%$ of its time proposing changes to the topology parameter. Experience suggests that the tree topology is a more difficult parameter for the MCMC to approximate relative to the exchangeability and stationary frequency parameters (in fact, it appears that the developers of `MrBayes` determined from their experience that the topology is ~ 20 times harder to approximate).

3. Propose a change to the selected parameter using the parameter-specific proposal mechanism. Different parameters may naturally have different prior probability densities—for example, the stationary frequencies are proportions (ranging between 0 and 1), and so are conveniently described using a Dirichlet prior probability density, whereas the alpha-shape parameter can range between zero and infinity, so is better described using a uniform prior probability density. Different kinds of prior probability densities will have specific proposal mechanisms—for example, there is something called a 'Dirichlet' proposal mechanism that is used to propose new values for parameters described by a Dirichlet prior, and something called a 'Slider' proposal mechanism that is used to propose new values for parameters described by a uniform prior. (Again, we'll go into the gory details of these proposal mechanisms another time. The main idea for now is that the proposal mechanisms generate a new parameter value by poking the current parameter value.)

   The design of proposal mechanisms is something of a dark art—trial and error are used to guide the development of mechanisms that work well. Nevertheless, there are basic criteria that the proposal mechanisms must meet. All proposal mechanisms must be:

   (a) stochastic (new parameter values must be proposed 'randomly')
   (b) irreducible (all parameter values must be potentially accessible by the chain)
   (c) aperiodic (the proposal mechanism must not induce epicycles of the chain)

4. Calculate the probability of accepting the proposed change, R:

$$R = \min\left[1,\ \underbrace{\frac{P(X \mid \theta')}{P(X \mid \theta)}}_{\text{likelihood ratio}} \cdot \underbrace{\frac{P(\theta')}{P(\theta)}}_{\text{prior ratio}} \cdot \underbrace{\frac{P(\theta \mid \theta')}{P(\theta' \mid \theta)}}_{\text{proposal ratio}}\right] \tag{6.1}$$

   The acceptance probability, R, is either equal to one or the product of three ratiosâĂŤso long as that product is less than one (because R is a probability, so it cannot be greater than one). So, what are these three ratios?

   Likelihood ratio: the likelihood ratio is simply the likelihood of the observations given the proposed value of the parameter, divided by the likelihood of the observations given the current value of the parameter. We can calculate the likelihood for any given parameterization of the phylogenetic model using the pruning (Felsenstein) algorithm.

   Prior ratio: in Bayesian inference, each parameter is a random variable, and so is described by a prior probability density. Accordingly, we can 'look up' (calculate) the prior probability of any specific parameter value. The prior ratio is simply the prior probability of the proposed and current parameter values.

   Proposal ratio: the proposal ratio is the probability of proposing the current parameter value given the proposed parameter value, divided by the converse. This is also called the Hastings ratio. This is equivalent to the rule that forces our robot to propose steps symmetrically; *i.e.*, that the probability of proposing a step from point A to point B in the terrain is equal to the probability of proposing a step from point B to point A. For inference problems in which the dimensions of the model are static, the proposal ratio is usually equal to one (we'll discuss exceptions to this in the context of reversible-jump MCMC in a future post).

   Note that if the proposal ratio is equal to one, the acceptance probability, R is based only on the product of the likelihood and prior ratios. Does that ring a bell? [Hint: think of Bayes' theorem.]

[Hint 2: think of the right side of Bayes' theorem.] [Hint 3: think of the numerator of the right side of Bayes' theorem.] That is, the posterior probability is proportional to the product of the likelihood and prior probability. Accordingly, the acceptance probability is the posterior probability of the proposed state divided by the posterior probability of the current state. Just as we programmed our robot, if the ratio of the proposed and current elevations (posterior probabilities) is greater than 1 (*i.e.*, it is an uphill step), we always accept the proposed change. When the ratio of the proposed and current elevations is less than 1 (*i.e.*, it is an downhill step), we may or may not take the proposed step (stay tuned).

5. Generate a uniform random number between zero and one, $U[0,1]$. If $U < R$, accept the proposed change; otherwise, the current state of the chain becomes the next state of the chain. Downhill moves will be accepted 'randomly' in proportion to the difference in elevation.

6. Repeat steps $2-5$ an 'adequate' number of times.

That's it! Notice that the decision to accept or reject proposed steps in the MCMC (and thus to sample from the joint posterior probability density) is based exclusively on the likelihood and prior probability of the proposed and current states—two quantities that are easy to calculate. The beautiful, fantabulous achievement of the Metropolis-Hastings algorithm is the slaying of the beastly denominator of Bayes' theorem. Specifically, the algorithm allows us to do inference while entirely avoiding the need to calculate the (completely intractable) marginal likelihood!

Approximating the joint posterior probability density is based on samples from the MCMC: a chain following the simple rules outlined above will sample parameter values in proportion to their posterior probability. That is, the proportion of time that the chain spends in any particular state is a valid approximation of the posterior probability of that state (*e.g.*, if a clade is present in 87% of the samples drawn from the chain, then the posterior probability of that clade is estimated to be 0.87).

Why do we accept proposals to states with lower posterior probability? People familiar with maximum-likelihood estimation often misunderstand the purpose of accepting proposals to states with a lower posterior probability. In maximum-likelihood inference, the game is to simply climb relentlessly and mindlessly up the likelihood surface in search of the very tip of the globally highest peak. Accordingly, the only reason these zombie hill climbers might consider a downward move would be motivated by concerns that they were currently climbing up a local (rather than global) optimum. By contrast, Bayesians are not just interested in the elevation of the very tip of the absolute peak of the posterior probability surface, but rather are interested in exploring and estimating the entire joint posterior probability density—we want topographical map of the entire posterior probability, not the elevation of the tip of the very highest peak in parameter space.

This survey of the joint posterior probability density results in a more robust inference procedure (inferences are averaged over the joint posterior probability of all model parameters), and allows us to estimate and evaluate the marginal posterior probability density for each of the parameters (these can be viewed by querying the MCMC samples with respect to the parameter of interest—we will do this in future tutorials). Accordingly, it is not sufficient for the chain to reach the peak of the joint posterior probability density; we need the chain to mix over the entire stationary distribution, spending time at each location in proportion to its posterior probability.

Next to the specification of priors (which are inspired by much more philosophical considerations), performance of the MCMC algorithm used to approximate the joint posterior probability distribution is the

greatest concern associated with Bayesian inference (it is certainly a more general concern, as it holds even when the priors are uncontroversial, and, in my opinion, is a more legitimate and practical concern). There are three closely related issues associated with MCMC performance: (1) convergence (is the robot sampling from the stationary distribution); (2) mixing (is the robot efficiently moving over the posterior probability density); and (3) adequacy (has the robot collected sufficient samples from the target distribution to describe it adequately).

6.3 Diagnosing MCMC performance

6.3.1 Running Markov chain Monte Carlo Simulations & Assessing Output

Data & Model

This is an analysis of 13 species within the genus *Fagus* (the beeches), based 2576 sites sampled from one nuclear gene region (ITS) and two chloroplast gene regions (rbcL and matK). Here we assume a GTR+Γ+I substitution model that is uniformly shared by all sites: we refer to this as the 'uniform' partition scheme, or PS0 for short. This is the same data and model as we used previously in the CTMC tutorial.

Look at Model file & MCMC file in a text editor.

Running MCMC

Let us now explore the behavior of our Markov chain. The aim here in these exercises is to visually inspect the output generated by `RevBayes`. You should look at the trace plots using `Tracer`. Watch out for badly mixing chains, poor performance of the MCMC and correlated parameter estimates.

The first analysis which we perform is specified as follows:

- uniform GTR+I+G
- single move per cycle
- large proposal for all parameters
- no-pre-burnin

You can run this analysis directly using

```
source("RevBayes_scripts/analysis/mcmc_run1_PS0.Rev")
mymcmc.operatorSummary()
```

The operator summary provides you with an overview of the moves that you have used for this MCMC run. It tells you what the weight for each move was, the variables it has been working on, the number of times the move was used, and how often the move was accepted. Good acceptance rates for continuous parameters are between 20% and 60%. For discrete characters, such as the phylogeny, there is no rule of thumb what good acceptance is. Instead, the acceptance rate of tree topology proposal strongly depends on the current data.

Now let us open the generated output file in `Tracer`. The file should be called output/fagus_ps0_posterior_run1.log.

What you may notice is that the ESS are very low and the MCMC is mixing very poor. This is not good. We'll try a new analysis but instead run multiple moves per iteration. The new setting is:

- uniform GTR+I+G
- *multiple moves per cycle*
- large proposal for all parameters
- flat & low proposal weights
- no-pre-burnin

You can run this analysis in `RevBayes` using

```
source("RevBayes_scripts/analysis/mcmc_run2_PS0.Rev")
mymcmc.operatorSummary()
```

Look at the output in `Tracer` again. You should see that the ESS values are still very low and the chain is still not mixing well.

One issue that you see is that the probability of a site being invariant is update not often enough. It is quite likely that our move proposed too many bad new parameters. So let us change the window size for the sliding window move applied on the probability of invariant sites from

```
moves[mi++] <- mvSlide(pinvar, delta=10.0, tune=false, weight=1.0)
```

to

```
moves[mi++] <- mvSlide(pinvar, delta=1.0, tune=false, weight=1.0)
```

This will give us now the following set-up:

- uniform GTR+I+G
- multiple moves per cycle
- *smaller proposals for Pinv*
- flat & low proposal weights
- no-pre-burnin

Run the analysis file and look how often the Sliding Move is now accepted.

```
source("RevBayes_scripts/analysis/mcmc_run3_PS0.Rev")
mymcmc.operatorSummary()
```

Open the output and look at it in `Tracer`. We are now getting better ESS values and better mixing of the MCMC.

That worked quite well but it seems to be cumbersome to adjust all the tuning parameters of the moves manually. Instead, let us allow `RevBayes` to auto-tune the moves. Just change **`tune=true`** for all moves where it was **`tune=false`** before. Now, also include a pre-burnin. The pre-burnin runs if you say **`mymcmc.burnin(generations=1000,tuningInterval=100)`**. This runs a chain for 1000 iterations and updates the tuning parameters, such as the window size, so that you achieve a good acceptance rate. That means, if previously a move accepted too few proposal it will decrease the window size.

Here is the new model set-up:

- uniform GTR+I+G
- multiple moves per cycle
- flat & low proposal weights
- *pre-burnin (with scale adjusted from previous)*

Run the analysis in `RevBayes`:

```
source("RevBayes_scripts/analysis/mcmc_run4_PS0.Rev")
mymcmc.operatorSummary()
```

Look at the output given by the operator summary. Notice how the tuning parameters have been updated. Also look at the output in `Tracer`. You will see that the ESS values are much better because the chain is mixing better.

In the next step we chain the proposal weights so that moves are used more often, especially for the parameters that are not mixing well. Our new setting is:

- uniform GTR+I+G
- multiple moves per cycle
- *variable proposal weights*
- pre-burnin (with scale adjusted from previous)

Run the `RevBayes` analysis.

```
source("RevBayes_scripts/analysis/mcmc_run5_PS0.Rev")
mymcmc.operatorSummary()
```

When you look at the output you will see that it now mixing fairly well. So let us focus on the approximated posterior distribution. Specifically, what is the posterior mean of the tree-length (**TL**)? This tree-length is surprisingly large. We could specify a prior distribution with a lower expectation (mean) on the tree-length.

- uniform GTR+I+G
- multiple moves per cycle
- variable proposal weights
- pre-burnin (with scale adjusted from previous)
- *branch length prior to* ***dnExponential(20.0)*** *(mean 0.05)*

Run the new analysis in `RevBayes`.

```
source("RevBayes_scripts/analysis/mcmc_run6_PS0.Rev")
mymcmc.operatorSummary()
```

If you look at the results from this analysis, you will notice that the tree-length estimates are better. However, if you look at the joint posterior density of the tree-length (**TL**) and the probability of a site being invariant (**pinvar**), then you will notice that there is a very high correlation between these two. It seems as if our model is over-parameterized. To resolve this issue we need to remove/fix some of our parameters. Open the data file and look how variable each site is. What is your impression on how many sites are invariant?

We remove the gamma distributed rate variation among sites because most sites either seem to be fixed or evolve under our GTR model. We also reset the branch length priors to **dnExponential(10.0)**.

- uniform GTR+I
- multiple moves per cycle
- variable proposal weights
- pre-burnin (with scale adjusted from previous)

Run the new script in `RevBayes`.

```
source("RevBayes_scripts/analysis/mcmc_run7_PS0.Rev")
mymcmc.operatorSummary()
```

Look again at output in `Tracer`. We seem to be doing a pretty good job with one chain. This is our visual method to inspect the MCMC output.

Next, let's compare it to the prior. Here we want to see how much information we actually have from the data and how much our prior influenced our posterior estimates. Run now an analysis that does not use the data. You can do this in `RevBayes` the exact same way as running an MCMC on the exact same model, you only need to call **`ymmcmc.run(1000,underPrior=true)`**.

```
source("RevBayes_scripts/analysis/mcmc_run8_1_prior_PS0.Rev")
mymcmc.operatorSummary()
```

Load both the MCMC output under the prior and the posterior into `Tracer`. Compare the approximated distributions on each parameter between both outputs.

Since we are doing fine for a single run, try multiple runs to check if the results are reproducible.

```
source("RevBayes_scripts/analysis/mcmc_run7_1_PS0.Rev")
source("RevBayes_scripts/analysis/mcmc_run7_2_PS0.Rev")
source("RevBayes_scripts/analysis/mcmc_run7_3_PS0.Rev")
```

...and the prior...

```
source("RevBayes_scripts/analysis/mcmc_run8_2_prior_PS0.Rev")
source("RevBayes_scripts/analysis/mcmc_run8_3_prior_PS0.Rev")
source("RevBayes_scripts/analysis/mcmc_run8_4_prior_PS0.Rev")
```

Apply multiple-run diagnostics:

- PSRF
- ASDSF
- comparetrees

6.4 Semi-automatic MCMC diagnosis using bonsai

Diagnosing MCMC performance manually can be extremely cumbersome, especially for more complex models. The R package bonsai performs a standard set of diagnostics on MCMC output and generates a report that highlights potentially pathological MCMC behaviors. To use bonsai, download the package source from https://bitbucket.org/mrmay/bonsai/downloads, launch the R console, and install the package from source (making sure to install the packages that bonsai relies on as well):

```
install.packages(packages=c('coda','tools','RColorBrewer','entropy','xtable'),
    dependencies=TRUE)
install.packages('< path to bonsai >',repos=NULL,type='source')
library(bonsai)
```

Move the output files from `run7` to the folder `.../RB_MCMC_Tutorial/output/output_for_bonsai`.

Going back to the R console, provide a name for the project.

```
project <- 'Fagus run 7'
```

Point bonsai at the directory where we placed our log files.

```
path <- '.../RB_MCMC_Tutorial/output/output_for_bonsai'
```

Finally, we create the bonsai object and execute the `runBonsai()` function. It will generate a report that draws your attention to potential issues.

```
fagus_proj <- bonsai(project=project,path=path)
fagus_proj$runBonsai()
```

Check the flags generated by bonsai (Figure 6.1; your results may differ). Many of these flags may be innocuous; it is up to the user to decide whether a particular flag is truly problematic. For example, a correlation between `Likelihood` and `Posterior` is not problematic; however, a significant p-value for Geweke's diagnostic for `pi[3]` indicates that this parameter has not converged to its stationary distribution.

There are additional bonsai reports for more complex models demonstrating a wider array of MCMC pathologies in the `.../RB_MCMC_Tutorial/bonsai_pre_cooked` directory.

Bibliography

2.2.1 Critical flags

- Run 2: Parameter `pi[3]` has critically low p-value for Geweke's diagnostic ($p = 0.002$)
- Parameters `Likelihood` and `Posterior` are strongly correlated ($\rho = 1$)
- Parameters `TL` and `Prior` are strongly correlated ($\rho = -1$)

2.2.2 Major flags

- Run 2: Parameter `Prior` has very low p-value for Geweke's diagnostic ($p = 0.023$)
- Run 2: Parameter `TL` has very low p-value for Geweke's diagnostic ($p = 0.023$)
- Parameters `Prior` and `Posterior` are correlated ($\rho = 0.292$)
- Parameters `TL` and `Posterior` are correlated ($\rho = -0.292$)
- Parameters `pinvar` and `Posterior` are correlated ($\rho = -0.268$)

Figure 6.1: Flags generated by bonsai.

Chapter 7

Model Selection and Bayes Factors

7.1 Overview

This tutorial demonstrates some general principles of Bayesian model comparison, which is based on estimating the marginal likelihood of competing models and then comparing their relative fit to the data using Bayes factors. We consider the specific case of calculating Bayes factors to select among different substitution models.

7.1.1 Requirements

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

- RB_Getting_Started
- RB_Data_Tutorial
- RB_CTMC_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).

7.2 Data and files

We provide several data files 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

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

7.3 Introduction

For most sequence alignments, several (possibly many) substitution models of varying complexity are plausible *a priori*. We therefore need a way to objectively identify the model that balances estimation bias and inflated error variance associated with under- and over-parameterized models, respectively. Increasingly, model selection is based on *Bayes factors* (*e.g.*, Suchard et al. 2001; Lartillot and Philippe 2006; Xie et al. 2011; Baele et al. 2012; 2013), which involves first calculating the marginal likelihood of each candidate model and then comparing the ratio of the marginal likelihoods for the set of candidate models.

Given two models, M_0 and M_1, the Bayes-factor comparison assessing the relative fit of each model to the data, $BF(M_0, M_1)$, is:

$$BF(M_0, M_1) = \frac{\text{posterior odds}}{\text{prior odds}}.$$

The posterior odds is the posterior probability of M_0 given the data, $\mathbf{X}$, divided by the posterior odds of M_1 given the data:

$$\text{posterior odds} = \frac{\mathbb{P}(M_0 \mid \mathbf{X})}{\mathbb{P}(M_1 \mid \mathbf{X})},$$

and the prior odds is the prior probability of M_0 divided by the prior probability of M_1:

$$\text{prior odds} = \frac{\mathbb{P}(M_0)}{\mathbb{P}(M_1)}.$$

Thus, the Bayes factor measures the degree to which the data alter our belief regarding the support for M_0 relative to M_1 (Lavine and Schervish 1999):

$$BF(M_0, M_1) = \frac{\mathbb{P}(M_0 \mid \mathbf{X}, \theta_0)}{\mathbb{P}(M_1 \mid \mathbf{X}, \theta_1)} \div \frac{\mathbb{P}(M_0)}{\mathbb{P}(M_1)}. \tag{7.1}$$

Note that interpreting Bayes factors involves some subjectivity. That is, it is up to *you* to decide the degree of your belief in M_0 relative to M_1. Despite the absence of an absolutely objective model-selection threshold, we can refer to the scale (outlined by Jeffreys 1961) that provides a "rule-of-thumb" for interpreting these measures (Table 7.1).

Table 7.1: The scale for interpreting Bayes factors by Harold Jeffreys (1961).

Strength of evidence	$BF(M_0, M_1)$	$\mathbf{log}(BF(M_0, M_1))$	$\mathbf{log_{10}}(BF(M_0, M_1))$
Negative (supports M_1)	< 1	< 0	< 0
Barely worth mentioning	1 to 3.2	0 to 1.16	0 to 0.5
Substantial	3.2 to 10	1.16 to 2.3	0.5 to 1
Strong	10 to 100	2.3 to 4.6	1 to 2
Decisive	> 100	> 4.6	> 2

For a detailed description of Bayes factors see Kass and Raftery (1995)

Unfortunately, it is generally not possible to directly calculate the posterior odds to prior odds ratio. However, we can further define the posterior odds ratio as:

$$\frac{\mathbb{P}(M_0 \mid \mathbf{X})}{\mathbb{P}(M_1 \mid \mathbf{X})} = \frac{\mathbb{P}(M_0)}{\mathbb{P}(M_1)} \frac{\mathbb{P}(\mathbf{X} \mid M_0)}{\mathbb{P}(\mathbf{X} \mid M_1)},$$

where $\mathbb{P}(\mathbf{X} \mid M_i)$ is the *marginal likelihood* of the data (this may be familiar to you as the denominator of Bayes Theorem, which is variously referred to as the *model evidence* or *integrated likelihood*). Formally, the marginal likelihood is the probability of the observed data ($\mathbf{X}$) under a given model (M_i) that is averaged over all possible values of the parameters of the model (θ_i) with respect to the prior density on θ_i

$$\mathbb{P}(\mathbf{X} \mid M_i) = \int \mathbb{P}(\mathbf{X} \mid \theta_i)\mathbb{P}(\theta_i)dt. \tag{7.2}$$

This makes it clear that more complex (parameter-rich) models are penalized by virtue of the associated prior: each additional parameter entails integration of the likelihood over the corresponding prior density. If you refer back to equation 7.1, you can see that, with very little algebra, the ratio of marginal likelihoods is equal to the Bayes factor:

$$BF(M_0, M_1) = \frac{\mathbb{P}(\mathbf{X} \mid M_0)}{\mathbb{P}(\mathbf{X} \mid M_1)} = \frac{\mathbb{P}(M_0 \mid \mathbf{X}, \theta_0)}{\mathbb{P}(M_1 \mid \mathbf{X}, \theta_1)} \div \frac{\mathbb{P}(M_0)}{\mathbb{P}(M_1)}. \tag{7.3}$$

Therefore, we can perform a Bayes factor comparison of two models by calculating the marginal likelihood for each one. Alas, exact solutions for calculating marginal likelihoods are not known for phylogenetic models (see equation 7.2), thus we must resort to numerical integration methods to estimate or approximate these values. In this exercise, we will estimate the marginal likelihood for each partition scheme using both the stepping-stone (Xie et al. 2011; Fan et al. 2011) and path sampling estimators (Lartillot and Philippe 2006; Baele et al. 2012).

7.3.1 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)
- Hasegawa-Kishino-Yano (HKY) substitution model (Hasegawa et al. 1985)
- 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)

7.3.2 Estimating the Marginal Likelihood

We will estimate the marginal likelihood of a given model using a 'stepping-stone' (or 'path-sampling') algorithm. These algorithms are similar to the familiar MCMC algorithms, which are intended to sample from (and estimate) the joint posterior probability of the model parameters. Stepping-stone algorithms are like a series of MCMC simulations that iteratively sample from a specified number of discrete steps between the posterior and the prior probability distributions. The basic idea is to estimate the probability of the data for all points between the posterior and the prior—effectively summing the probability of the data over the prior probability of the parameters to estimate the marginal likelihood. Technically, the steps correspond to a series of `powerPosteriors()`: a series of numbers between 1 and 0 that are iteratively applied to the posterior. When the posterior probability is raised to the power of 1 (typically the first stepping stone), samples are drawn from the (untransformed) posterior. By contrast, when the posterior probability is raised to the power of 0 (typically the last stepping stone), samples are drawn from the prior. To perform a stepping-stone simulation, we need to specify (1) the *number* of stepping stones (power posteriors) that we will use to traverse the path between the posterior and the prior (e.g., we specify 50 or 100 stones), (2) the *spacing* of the stones between the posterior and prior (e.g., we may specify that the stones are distributed according to a beta distribution), (3) the number of samples to (and thinning) of samples to be drawn from each stepping stone, and (4) the direction we will take (i.e., from the posterior to the prior or vice versa).

This method computes a vector of powers from a beta distribution, then executes an MCMC run for each power step while raising the likelihood to that power. In this implementation, the vector of powers starts with 1, sampling the likelihood close to the posterior and incrementally sampling closer and closer to the prior as the power decreases.

Just to be safe, it is better to clear the workspace (if you did not just restart `RevBayes`)

```
clear()
```

Now set up the model as in the previous exercise. You should start with the simple Jukes-Cantor substitution model. Setting up the model requires:

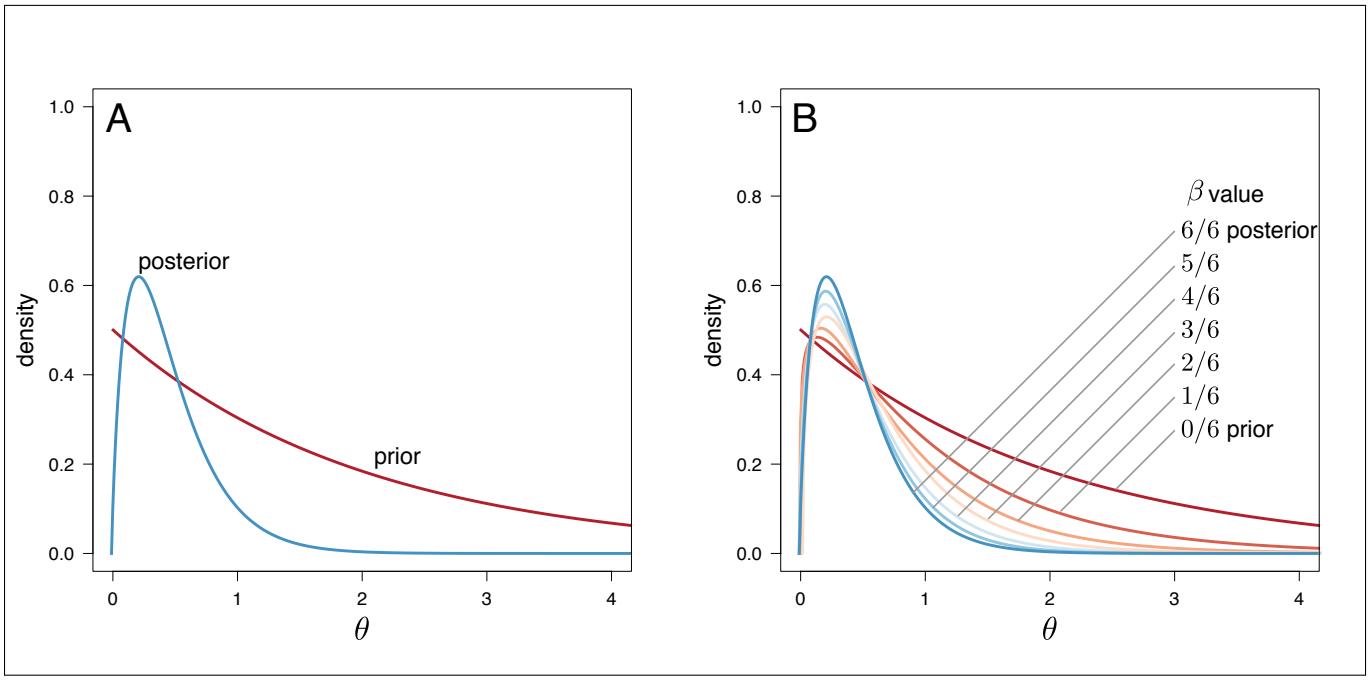


Figure 7.1: Estimating marginal likelihoods using stepping-stone simulation. Estimating the marginal likelihood involves integrating the likelihood of the data over the entire prior probability density for the model parameters. MCMC algorithms target the posterior probability density, which is typically concentrated in a small region of the prior probability density (A). Accordingly, standard MCMC simulation cannot provide unbiased estimates of the marginal likelihood because it will typically fail to explore most of the prior density. (B) Stepping-stone algorithms estimate the marginal likelihood by means of a series of MCMC-like simulations, where the likelihood is iterativey raised to a series of powers, effectivey forcing the simulation to more fully explore the prior density of the model parameters. Here, six uniformly spaced stones span the posterior, where the power posterior is $\beta = 6/6 = 1$, to the prior, where the power posterior is $\beta = 0/6 = 0$.

1. Loading the data and retrieving useful variables about it (*e.g.,* number of sequences and taxon names).
2. Specifying the instantaneous-rate matrix of the substitution model.
3. Specifying the tree model including branch-length variables.
4. Creating a random variable for the sequences that evolved under the `PhyloCTMC`.
5. Clamping the data.
6. Creating a model object.
7. Specifying the moves for parameter updates.

The following procedure for estimating marginal likelihoods is valid for any model in `RevBayes`. You will need to repeat this later for other models. First, we create the variable containing the power-posterior analysis. This requires that we provide a model and vector of moves, as well as an output file name. The `cats` argument sets the number of stepping stones.

```
pow_p = powerPosterior(mymodel, moves, monitors, "model1.out", cats=50)
```

We can start the power-posterior analysis by first burning in the chain and and discarding the first 10000 states. This will help ensure that analysis starts from a region of high posterior probability, rather than from some random point.

```
pow_p.burnin(generations=10000,tuningInterval=1000)
```

Now execute the run with the **.run()** function:

```
pow_p.run(generations=1000)
```

Once the power posteriors have been saved to file, create a stepping stone sampler. This function can read any file of power posteriors and compute the marginal likelihood using stepping-stone sampling.

```
ss = steppingStoneSampler(file="model1.out", powerColumnName="power", likelihoodColumnName="
    likelihood")
```

These commands will execute a stepping-stone simulation with 50 stepping stones, sampling 1000 states from each step. Compute the marginal likelihood under stepping-stone sampling using the member function **marginal()** of the **ss** variable and record the value in Table 7.2.

```
ss.marginal()
```

Path sampling is an alternative to stepping-stone sampling and also takes the same power posteriors as input.

```
ps = pathSampler(file="model1.out", powerColumnName="power", likelihoodColumnName="likelihood")
```

Compute the marginal likelihood under stepping-stone sampling using the member function **marginal()** of the **ps** variable and record the value in Table 7.2.

```
ps.marginal()
```

→ As an example we provide the file **RevBayes_scripts/marginalLikelihood_JukesCantor.Rev**.

7.3.3 Exercises

- Compute the marginal likelihoods of the *cytb* alignment for the following substitution models:
  - Jukes-Cantor (JC) substitution model
  - Hasegawa-Kishino-Yano (HKY) substitution model
  - General-Time-Reversible (GTR) substitution model
  - GTR with gamma distributed-rate model (GTR+G)
  - GTR with invariable-sites model (GTR+I)
  - GTR+I+G model
- Enter the marginal likelihood estimate for each model in the corresponding cell of Table 7.2.
- Repeat the above marginal likelihood analyses for the *mt-COX2 gene* and enter results in Table **??**.
- Which is the best fitting substitution model?

Table 7.2: Estimated marginal likelihoods for different substitution models for the cytb alignment*.

	Marginal lnL estimates	
Substitution Model	*Stepping-stone*	*Path sampling*
JC (M_1)		
HKY (M_2)		
GTR (M_3)		
GTR+Γ (M_4)		
GTR+I (M_5)		
GTR+Γ+I (M_6)		
Any other model (M_7)		
Any other model (M_8)		
Any other model (M_9)		

*you can edit this table

7.4 Computing Bayes Factors and Model Selection

Now that we have estimates of the marginal likelihood for each of our the candidate substitution models, we can evaluate their relative fit to the datasets using Bayes factors. Phylogenetic programs log-transform the likelihood values to avoid underflow: multiplying likelihoods (numbers < 1) generates numbers that are too small to be held in computer memory. Accordingly, we need to use a different form of equation 7.3 to calculate the ln-Bayes factor (we will denote this value $\mathcal{K}$):

$$\mathcal{K} = \ln[BF(M_0, M_1)] = \ln[\mathbb{P}(\mathbf{X} \mid M_0)] - \ln[\mathbb{P}(\mathbf{X} \mid M_1)], \tag{7.4}$$

where $\ln[\mathbb{P}(\mathbf{X} \mid M_0)]$ is the *marginal lnL* estimate for model M_0. The value resulting from equation 10.3 can be converted to a raw Bayes factor by simply taking the exponent of $\mathcal{K}$

$$BF(M_0, M_1) = e^{\mathcal{K}}. \tag{7.5}$$

Alternatively, you can directly interpret the strength of evidence in favor of M_0 in log space by comparing the values of $\mathcal{K}$ to the appropriate scale (Table refbftable, second column). In this case, we evaluate $\mathcal{K}$ in favor of model M_0 against model M_1 so that:

if $\mathcal{K} > 1$, model M_0 is preferred
if $\mathcal{K} < -1$, model M_1 is preferred.

Thus, values of $\mathcal{K}$ around 0 indicate that there is no preference for either model.

Using the values you entered in Table 7.2 and equation 10.3, calculate the ln-Bayes factors (using $\mathcal{K}$) for each model comparison. Enter your answers in Table 8.3 using the stepping-stone and the path-sampling estimates of the marginal log-likelihoods.

Table 7.3: Bayes factor calculation*.

Model comparison	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9
M_1	-								
M_2		-							
M_3			-						
M_4				-					
M_5					-				
M_6						-			
M_7							-		
M_8								-	
M_9									-

*you can edit this table

7.5 For your consideration...

In this tutorial you have learned how to use `RevBayes` to assess the *relative* fit of a pool of candidate substitution models to a given sequence alignment. Typically, once we have identified the "best" substitution model for our alignment, we would then proceed to use this model for inference. Technically, this is a decision to condition our inferences on the selected model, which explicitly assumes that it provides a reasonable description of the process that gave rise to our data. However, there are several additional issues to consider before proceeding along these lines, which we briefly mention below.

7.5.1 Accommodating Process Heterogeneity

In the analyses that we performed in this tutorial, we assumed that all sites in an alignment evolved under an identical substitution process. It is well established that this assumption is frequently violated in real datasets. Various aspects of the substitution process—the stationary frequencies, exchangeability rates, degree of ASRV, etc.—may vary across sites of our sequence alignment. For example, the nature of the substitution process may vary between codon positions of protein-coding genes, between stem and loop regions of ribosomal genes, or between different gene and/or genomic regions. It is equally well established that failure to accommodate this *process heterogeneity*—variation in the nature of the substitution process across the alignment—will cause biased estimates of the tree topology, branch lengths and other phylogenetic model parameters. We can accommodate process heterogeneity by adopting a *mixed-model* approach, where two or more subsets of sites are allowed to evolve under distinct substitution processes. We will demonstrate how to specify—and select among—alternative mixed models using `RevBayes` in a separate tutorial, RB_PartitionedData_Tutorial.

7.5.2 Assessing Model Adequacy

In this tutorial, we used Bayes factors to assess the fit of various substitution models to our sequence data, effectively establishing the *relative* rank of the candidate models. Even if we have successfully identified the very best model from the pool of candidates, however, the preferred model may nevertheless be woefully inadequate in an *absolute* sense. For this reason, it is important to consider *model adequacy*: whether a given model provides a reasonable description of the process that gave rise to our sequence data. We can assess the absolute fit of a model to a given dataset using *posterior predictive simulation*. This approach is based on the following premise: if the candidate model provides a reasonable description of the process that gave rise to our dataset, then we should be able to generate data under this model that resemble our observed data. We will demonstrate how to assess model adequacy using `RevBayes` in a separate tutorial, RB_ModelAdequacy_Tutorial.

7.5.3 Accommodating Model Uncertainty

Even when we have carefully assessed the relative and absolute fit of candidate models to our dataset, it may nevertheless be unwise to condition our inference on the best model. Imagine, for example, that there are several (possibly many) alternative models that provide a similarly good fit to our given dataset. In such scenarios, conditioning inference on *any* single model (even the 'best') ignores uncertainty in the chosen model, which will cause estimates to be biased. This is the issue of *model uncertainty*. The Bayesian framework provides a natural approach for accommodating model uncertainty by means of *model averaging*; we simply adopt the perspective that models (like standard parameters) are random variables, and integrate the inference over the distribution of candidate models. We will demonstrate how to accommodate model uncertainty using `RevBayes` in a separate tutorial, RB_ModelAveraging_Tutorial.

Bibliography

Baele, G., P. Lemey, T. Bedford, A. Rambaut, M. Suchard, and A. Alekseyenko. 2012. Improving the accuracy of demographic and molecular clock model comparison while accommodating phylogenetic uncertainty. Molecular Biology and Evolution 29:2157–2167.

Baele, G., W. Li, A. Drummond, M. Suchard, and P. Lemey. 2013. Accurate model selection of relaxed molecular clocks in bayesian phylogenetics. Molecular Biology and Evolution 30:239–243.

Fan, Y., R. Wu, M.-H. Chen, L. Kuo, and P. O. Lewis. 2011. Choosing among partition models in bayesian phylogenetics. Molecular Biology and Evolution 28:523–532.

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

Jeffreys, H. 1961. The theory of probability. Oxford University Press.

Jukes, T. and C. Cantor. 1969. Evolution of protein molecules. Mammalian Protein Metabolism 3:21–132.

Kass, R. and A. Raftery. 1995. Bayes factors. Journal of the American Statistical Association 90:773–795.

Lartillot, N. and H. Philippe. 2006. Computing Bayes factors using thermodynamic integration. Systematic Biology 55:195.

Lavine, M. and M. J. Schervish. 1999. Bayes factors: what they are and what they are not. The American Statistician 53:119–122.

Suchard, M. A., R. E. Weiss, and J. S. Sinsheimer. 2001. Bayesian selection of continuous-time markov chain evolutionary models. Molecular Biology and Evolution 18:1001–1013.

Tavaré, S. 1986. Some probabilistic and statistical problems in the analysis of DNA sequences. Some Mathematical Questions in Biologyï£¡DNA Sequence Analysis 17:57–86.

Xie, W., P. Lewis, Y. Fan, L. Kuo, and M. Chen. 2011. Improving marginal likelihood estimation for bayesian phylogenetic model selection. Systematic Biology 60:150–160.

Yang, Z. 1994. Maximum likelihood phylogenetic estimation from DNA sequences with variable rates over sites: Approximate methods. Journal of Molecular Evolution 39:306–314.

Part IV

Dating

Chapter 8

A Brief Introduction, Overview and Example of Dating, Calibration and Relaxed Clocks

8.1 Exercise: Comparing Relaxed-Clock Models & Estimating Time-Calibrated Phylogenies

8.1.1 Introduction

Central among the questions explored in biology are those that seek to understand the timing and rates of evolutionary processes. Accurate estimates of species divergence times are vital to understanding historical biogeography, estimating diversification rates, and identifying the causes of variation in rates of molecular evolution.

This tutorial will provide a general overview of divergence time estimation using fossil calibration and relaxed-clock model comparison in a Bayesian framework. The exercise will guide you through the steps necessary for estimating phylogenetic relationships and dating species divergences using the program `RevBayes`.

8.1.2 Getting Started

The various exercises in this tutorial take you through the steps required to perform phylogenetic analyses of the example datasets. In addition, we have provided the output files for every exercise so you can verify your results. (Note that since the MCMC runs you perform will start from different random seeds, the output files resulting from your analyses *will not* be identical to the ones we provide you.)

Download the starting tree file: http://bit.ly/1AbDPGK

Download the alignment file: http://bit.ly/1Gz4Drw

In this exercise, we will compare among different relaxed clock models and estimate a posterior distribution of calibrated time trees. The dataset we will use is an alignment of 10 caniform sequences, comprising 8 bears, 1 spotted seal, and 1 gray wolf. Additionally, we will use occurrence times from three caniform fossils to calibrate our analysis to absolute time (Table 8.1).

Table 8.1: Fossil species used for calibrating divergence times in the caniform tree.

Fossil species	Age range (My)	Citation
Hesperocyon gregarius	37.2–40	Wang (1994); Wang et al. (1999)
Parictis montanus	33.9–37.2	Clark and Guensburg (1972); Krause et al. (2008)
Kretzoiarctos beatrix	11.2–11.8	Abella et al. (2011; 2012)

The alignment in file **`data/bears_irbp.nex`** contains interphotoreceptor retinoid-binding protein (irbp) sequences for each extant species.

8.1.3 Creating Rev Files

This tutorial sets up three different relaxed clock models and a calibrated birth-death model. Because of the complexity of the various models, this exercise is best performed by specifying the models and samplers in different `Rev` files. At the beginning of each section, you will be given a suggested name for each component file; these names correspond to the provided `Rev` scripts that reproduce these commands.

Directory Structure

This tutorial assumes that you have a very specific directory structure when running `RevBayes`. First, you may want to put the `RevBayes` binary in your path if you're using a Unix-based operating system. Alternatively, you can place the binary in a directory from which you will execute `RevBayes`, e.g., the tutorial directory. The tutorial directory can be any directory on your file system, but you may want to create a new one so that you avoid conflicts with other `RevBayes` tutorials.

> Create a directory for this tutorial called `RB_Dating_Tutorial` (or any name you like), and navigate to that directory. This is the tutorial directory mentioned above.

For this exercise, the `Rev` code provided assumes that within the tutorial directory exists subdirectories. These directories must have the same names given here, unless you wish to also change the `Rev` code to conform to your specific directory names.

The first subdirectory will contain the data files (downloaded in Section 8.1.2).

> Create a directory called `data` in your tutorial directory.
>
> Save the tree and alignment files downloaded above (Section 8.1.2) in the **`data`** directory.

The second subdirectory will contain the `Rev` files you write to execute the exercises in this tutorial.

> Create a directory called `RevBayes_scripts` in your tutorial directory.

This tutorial will guide you through creating all of the files necessary to execute the analyses without typing the `Rev` language syntax directly in the `RevBayes` console. Since the scripts must point to model and analysis files in a modular way, it is important to be aware of you directory structure and if you choose to do something different, make sure that the file paths given throughout the tutorial are correct.

Finally, we'll need a directory for all of the files written by our analyses. For some operations, `RevBayes` can create this directory on the fly for you. However, it may be safer just to add it now.

> Create a directory called `output` in your tutorial directory.

The only files you need for this exercise are now in the **`data`** directory. Otherwise, you will create all of the `Rev` files specifying the models and analyses. All of the `Rev` files you write for this tutorial are available on the `RevBayes` Tutorial GitHub repository at this URL: http://bit.ly/1CWTRdF. You can refer to these examples to verify your own work.

8.1.4 Calibrating the Birth-Death Model

Fortunately, the fossil record for caniforms (and other carnivores) is quite good. We must formulate a birth-death model that accounts for the fossil occurrence times in Table 8.1. This part of the exercise will involve specifying a birth-death model with clamped stochastic nodes representing the observation times of two fossils descended from internal nodes in our tree: (1) *Parictis montanus*, the oldest fossil in the family Ursidae, a stem fossil bear, and (2) *Kretzoiarctos beatrix*, the fossil Ailuropodinae, a crown fossil bear. Additionally, we will use the canid fossil, *Hesperocyon gregarius*, to offset the age of the root of the tree.

In `RevBayes`, calibrated internal nodes are treated differently than in many other programs for estimating species divergence times (e.g., BEAST). This is because the graphical model structure used in `RevBayes` does not allow a stochastic node to be assigned more than one prior distribution. By contrast, the common approach to applying calibration densities as used in other dating softwares leads to incoherence in the calibration prior (for detailed explainations of this see Warnock et al. 2012; Heled and Drummond 2012; Heath et al. 2014). More explicitly, common calibration approaches assume that the age of a calibrated node is modeled by the tree-wide diversification process (e.g., birth-death model) *and* a parametric density parameterized by the occurrence time of a fossil (or other external prior information). This can induce a calibration prior density that is not consistent with the birth-death process or the parametric prior distribution. Thus, approaches that condition the birth-death process on the calibrated nodes are more statistically coherent (Yang and Rannala 2006).

In `RevBayes`, calibration densities are applied in a different way, treating fossil observation times like data. The graphical model in Figure 9.2 illustrates how calibrated nodes are specified in the directed acyclic graph (DAG). Here, the age of the calibration node (i.e., the internal node specified as the MRCA of the fossil and a set of living species) is a deterministic node—e.g., denoted o_1 for fossil $\mathcal{F}_1$—and acts as an offset on the stochastic node representing the age of the fossil specimen. The fossil age, $\mathcal{F}_i$, is specified as a stochastic node and clamped to its *observed* age in the fossil record. The node $\mathcal{F}_i$ is modeled using a distribution that describes the waiting time from the speciation event to the appearance of the observed fossil. Thus, if the MCMC samples any state of Ψ for which the age of $\mathcal{F}_i$ has a probability of 0, then that state will always be rejected, effectively calibrating the birth-death process without applying multiple prior densities to any calibrated node (Fig. 9.2).

The root age is treated differently, however. Here, we condition the birth-death process on the speciation time of the root, thus this variable is not part of the time-tree parameter. The root age can thus be given any parametric distribution over positive real numbers (Fig. 9.2).

Create the Rev File

Open your text editor and create the birth-death model file called `m_BDP_bears.Rev` in the `RevBayes_scripts` directory.

Enter the `Rev` code provided in this section in the new model file.

Read in the Starting Tree

When calibrating nodes in the birth-death process, it is very helpful to have a starting tree that is consistent with the topology constraints and calibration priors, otherwise, the probability of the model would be 0

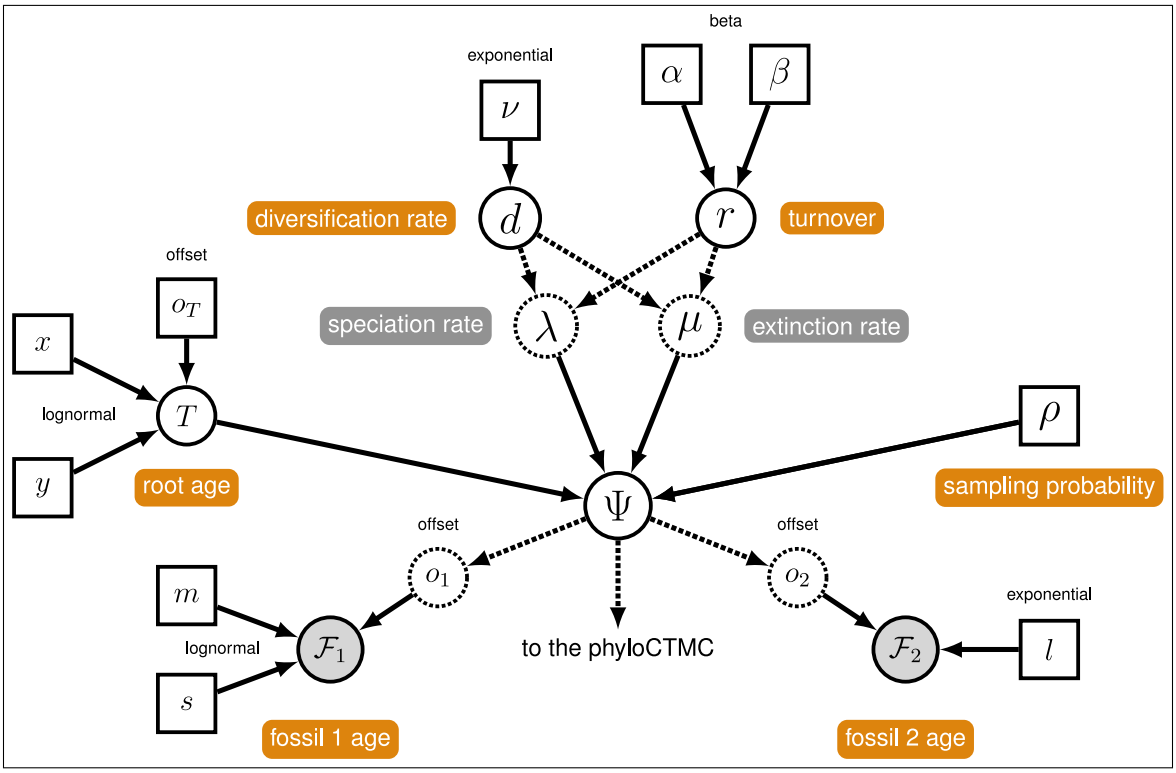


Figure 8.1: The graphical model representation of the node-calibrated birth-death process in RevBayes.

and the MCMC cannot run. For a starting tree we will use the tree estimated by dos Reis et al. (2012).

```
T <- readTrees("data/bears_dosReis.tre")[1]
```

From the tree we can initialize some useful variables.

```
n_taxa <- T.ntips()
names <- T.names()
```

Birth-Death Parameters

We will begin by setting up the model parameters and proposal mechanisms of the birth-death model. Note that we have not initialized the workspace iterator **mi** yet. Because of this, if you typed these lines in the RevBayes console, you would get an error. Since this code is intended to be in a sourced Rev file, we are assuming that you would initialize **mi** before calling **source("RevBayes_scripts/m_BDP_Tree_bears.Rev")**.

We will use the parameterization of the birth-death process specifying the diversification and turnover. For a more detailed tutorial on the simple birth-death model, please refer to the tutorial in the RevBayes repository: http://bit.ly/10UKeuq.

Diversification

Diversification (d) is the speciation rate (λ) minus the extinction rate (μ): $d = \lambda - \mu$.

```
diversification ~ dnExponential(10.0)
moves[mi++] = mvScale(diversification,lambda=1.0,tune=true,weight=3.0)
```

Turnover

Turnover is: $r = \mu/\lambda$.

```
turnover ~ dnBeta(2.0, 2.0)
moves[mi++] = mvSlide(turnover,delta=1.0,tune=true,weight=3.0)
```

Deterministic Nodes for Birth and Death Rates

The birth rate and death rate are deterministic functions of the diversification and turnover. First, create a deterministic node for $1 - r$, which is the denominator for each formula.

```
denom := abs(1.0 - turnover)
```

Now, the rates will both be positive real numbers that are variable transformations of the stochastic variables.

```
birth_rate := diversification / (denom)
death_rate := (turnover * diversification) / (denom)
```

Sampling Probability

Fix the probability of sampling to a known value. Since there are approximately 147 described caniform species, we will create a constant node for this parameter that is equal to 10/147.

```
rho <- 0.068
```

Prior on the Root Node

The fossil *Hesperocyon gregarius* is a fossil descendant of the most-recent common ancestor of all caniformes and has an occurrence time of ∼38 Mya. Thus, we can assume that the probability of the root age being younger than 38 Mya is equal to 0, using this value to offset a prior distribution on the root-age.

First specify the occurrence-time of the fossil.

```
tHesperocyon <- 38.0
```

We will assume a lognormal prior on the root age that is offset by the observed age of *Hesperocyon gregarius*. We can use the previous analysis by dos Reis et al. (2012) to parameterize the lognormal prior on the root time. The age for the MRCA of the caniformes reported in their study was ∼49 Mya. Therefore, we can specify the mean of our lognormal distribution to equal $49 - 38 = 11$ Mya. Given the expected value of the lognormal (**mean_ra**) and a standard deviation (**stdv_ra**), we can also compute the location parameter of the lognormal (**mu_ra**).

```
mean_ra <- 11.0
stdv_ra <- 0.25
mu_ra <- ln(mean_ra) - ((stdv_ra*stdv_ra) * 0.5)
```

With these parameters we can instantiate the root age stochastic node with the offset value.

```
root_time ~ dnLognormal(mu_ra, stdv_ra, offset=tHesperocyon)
```

Topology Constraints & Time Tree

To create the tree with calibrated nodes, we must constrain the topology such that the calibrated nodes always have the same descendants.

The two non-root nodes we are calibrating in this tree is the MRCA of all living bears:

```
clade_Ursidae <- clade("Ailuropoda_melanoleuca","Tremarctos_ornatus","
   Helarctos_malayanus", "Ursus_americanus","Ursus_thibetanus","Ursus_arctos","
   Ursus_maritimus","Melursus_ursinus")
```

And the MRCA of all bears and pinnipeds.

```
clade_UrsPinn <- clade("Ailuropoda_melanoleuca","Tremarctos_ornatus","
   Helarctos_malayanus", "Ursus_americanus","Ursus_thibetanus","Ursus_arctos","
   Ursus_maritimus","Melursus_ursinus", "Phoca_largha")
```

Once we have a set of constraints, we can use the vector function **v()** to bind them in a constant vector.

```
constraints <- v(clade_Ursidae, clade_UrsPinn)
```

Now we have all of the elements needed to specify the time-tree parameter.

```
timetree ~ dnBDP(lambda=birth_rate, mu=death_rate, rho=rho, rootAge=root_time,
    samplingStrategy="uniform", condition="nTaxa", nTaxa=n_taxa, names=names,
    constraints=constraints)
```

Calibrating Constrained Nodes

In order that our tree is consistent with the calibration ages, we must first set the value of the time-tree node to our starting tree.

```
timetree.setValue(T)
```

To begin specifying the calibration density on the MRCA of all ursids, we must first create the deterministic node representing the age of the MRCA. The way in which these densities work requires the offset to be negative. Therefore we are creating two deterministic variables, one positive for monitoring, and one negative for the off-set. We use the **tmrca()** function to create these nodes which require that you provide a clade constraint.

```
tmrca_Ursidae := tmrca(timetree,clade_Ursidae)
n_TMRCA_Ursidae := -(tmrca_Ursidae)
```

Now, we must specify our fossil occurrence time. This is the age for the fossil panda, *Kretzoiarctos beatrix*. Note that we also make this value negative.

```
tKretzoiarctos <- -11.2
```

Create the stochastic node for the age of the crown ursid fossil, using a lognormal distribution.

```
M <- 10
sdv <- 0.25
mu <- ln(M) - ((sdv * sdv) * 0.5)
crown_Ursid_fossil ~ dnLnorm(mu, sdv, offset=n_TMRCA_Ursidae)
```

Now clamp the fossil age stochastic node with the observation time of *Kretzoiarctos beatrix*

```
crown_Ursid_fossil.clamp(tKretzoiarctos)
```

Next we will create the variable for the age of the MRCA of all bears and pinnipeds.

```
tmrca_UrsidaePinn := tmrca(timetree,clade_UrsPinn)
n_TMRCA_UrsidaePinn := -(tmrca_UrsidaePinn)
```

Set the observed time for the stem fossil bear.

```
tParictis <- -33.9
```

Create the stochastic node using the exponential prior and clamp it with the observation time of the fossil.

```
stem_Ursid_fossil ~ dnExponential(lambda=0.0333, offset=n_TMRCA_UrsidaePinn)
stem_Ursid_fossil.clamp(tParictis)
```

Proposals on the Time Tree (Node Ages Only)

Next, create the vector of moves. These tree moves act on node ages:

```
moves[mi++] = mvNodeTimeSlideUniform(timetree, weight=30.0)
moves[mi++] = mvSlide(root_time, delta=2.0, tune=true, weight=10.0)
moves[mi++] = mvScale(root_time, lambda=2.0, tune=true, weight=10.0)
moves[mi++] = mvTreeScale(tree=timetree, rootAge=root_time, delta=1.0, tune=true,
    weight=3.0)
```

Now save and close the file called `m_BDP_bears.Rev`. This file, with all the model specifications will be loaded by other `Rev` files.

8.1.5 Specifying Branch-Rate Models

The next sections will walk you through setting up the files specifying different relaxed clock models. Each section will require you to create a separate `Rev` file for each relaxed clock model, as well as for each marginal-likelihood analysis.

The Global Molecular Clock Model

The global molecular clock assumes that the rate of substitution is constant over the tree and over time (Fig. 8.2).

Create the Rev File

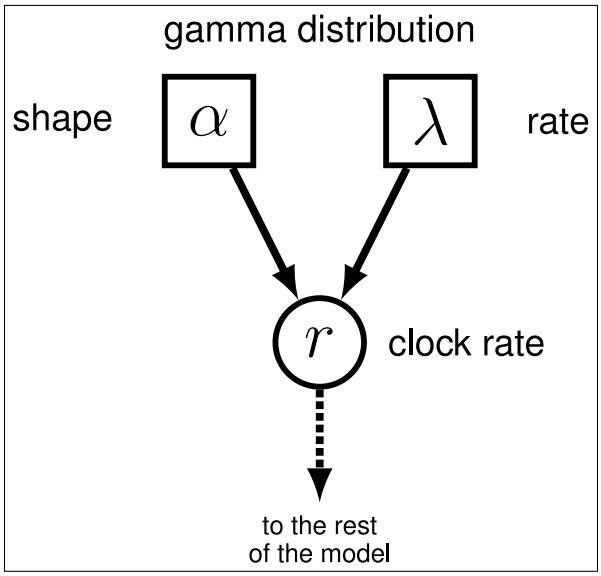


Figure 8.2: The graphical model representation of the global molecular clock model used in this exercise.

Open your text editor and create the global molecular clock model file called `m_GMC_bears.Rev` in the `RevBayes_scripts` directory.

Enter the `Rev` code provided in this section in the new model file. Keep in mind that we are creating modular model files that can be sourced by different analysis files. Thus, the `Rev` code below will still depend on variable initialized in different files.

The Clock-Rate

The clock-rate parameter is a stochastic node from a gamma distribution.

```
clock_rate ~ dnGamma(2.0,4.0)
moves[mi++] = mvScale(clock_rate,lambda=0.5,tune=true,weight=5.0)
```

The Sequence Model and Phylogenetic CTMC

Specify the parameters of the GTR model and the moves to operate on them.

```
sf ~ dnDirichlet(v(1,1,1,1))
er ~ dnDirichlet(v(1,1,1,1,1,1))
Q := fnGTR(er,sf)
moves[mi++] = mvSimplexElementScale(er, alpha=10.0, tune=true, weight=3.0)
moves[mi++] = mvSimplexElementScale(sf, alpha=10.0, tune=true, weight=3.0)
```

And instantiate the phyloCTMC.

```
phySeq ~ dnPhyloCTMC(tree=timetree, Q=Q, branchRates=abs_clock_rt, nSites=n_sites,
    type="DNA")
phySeq.clamp(D)
```

This is all we will include in the global molecular clock model file.

Save and close the file called `m_GMC_bears.Rev` in the **RevBayes_scripts** directory.

Estimate the Marginal Likelihood

Now we can use the model files we created and estimate the marginal likelihood under the global molecular clock model (and all other model settings). You can enter the following commands directly in the `RevBayes` console, or you can create another `Rev` script.

Open your text editor and create the marginal-likelihood analysis file under the global molecular clock model. Call the file: `mlnl_GMC_bears.Rev` and save it in the **RevBayes_scripts** directory.

Load Sequence Alignment — Read in the sequences and initialize important variables.

```
D <- readDiscreteCharacterData(file="data/bears_irbp.nex")
n_sites <- D.nchar(1)
mi = 1
```

The Calibrated Time-Tree Model — Load the calibrated tree model from file using the **source()** function. Note that this file does not have moves that operate on the tree topology, which is helpful when you plan to estimate the marginal likelihoods and compare different relaxed clock models.

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

Load the GMC Model File — Source the file containing all of the parameters of the global molecular clock model. This file is called `m_GMC_bears.Rev`.

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

We can now create our workspace model variable with our fully specified model DAG. We will do this with the **model()** function and provide a single node in the graph (**er**).

```
mymodel = model(er)
```

Run the Power-Posterior Sampler and Compute the Marginal Likelihoods — With a fully specified model, we can set up the **powerPosterior()** analysis to create a file of 'powers' and likelihoods from which we can estimate the marginal likelihood using stepping-stone or path sampling. This method computes a vector of powers from a beta distribution, then executes an MCMC run for each power step while raising the likelihood to that power. In this implementation, the vector of powers starts with 1, sampling the likelihood close to the posterior and incrementally sampling closer and closer to the prior as the power decreases.

First, we initialize a monitor which will log the MCMC samples for each parameter at every step in the power posterior.

```
monitors[1] = mnModel(filename="output/GMC_posterior_pp.log",printgen=10, separator =
    TAB)
```

Next, we create the variable containing the power posterior. This requires us to provide a model and vector of moves, as well as an output file name. The **cats** argument sets the number of power steps. Once we have specified the options for our sampler, we can then start the run after a burn-in/tuning period.

```
pow_p = powerPosterior(mymodel, moves, monitors, "output/GMC_bears_powp.out", cats=50,
     sampleFreq=10)
pow_p.burnin(generations=5000,tuningInterval=200)
pow_p.run(generations=1000)
```

Compute the marginal likelihood using two different methods, stepping-stone sampling and path sampling.

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

### use path sampling to calculate marginal likelihoods
ps = pathSampler(file="output/GMC_bears_powp.out", powerColumnName="power",
    likelihoodColumnName="likelihood")
ps.marginal()
```

If you have entered all of this directly in the RevBayes console, you will see the marginal likelihoods under each method printed to screen. Otherwise, if you have created the separate Rev file m_GMC_bears.Rev in the **RevBayes_scripts** directory, you now have to directly source this file in RevBayes (after saving the up-to-date content).

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

Once you have completed this analysis, record the marginal likelihoods under the global molecular clock model in Table 10.1.

The Uncorrelated Lognormal Rates Model

The uncorrelated lognormal (UCLN) model relaxes the assumption of a single-rate molecular clock. Under this model, the rate associated with each branch in the tree is a stochastic node. Each branch-rate variable is drawn from the same lognormal distribution (Fig. 8.3).

Given that we might not have prior information on the parameters of the lognormal distribution, we can assign hyper priors to these variables. Generally, it is more straightforward to construct a hyperprior on the expectation (i.e., the mean) of a lognormal density rather than the location parameter μ. Here, we will assume that the mean branch rate is exponentially distributed and as is the stochastic node representing the standard deviation. With these two parameters, we can get the location parameter of the lognormal by:

$$\mu = \log(M) - \frac{\sigma^2}{2}.$$

Thus, μ is a deterministic node, which is a function of M and σ. In Figure 8.3, we can represent the vector of N branch rates using the plate notation.

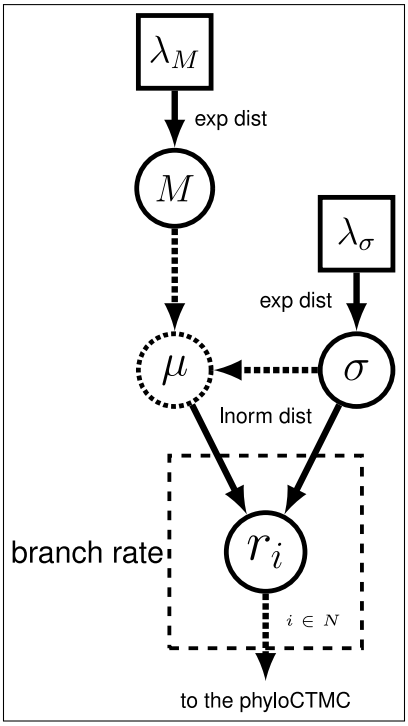


Figure 8.3: The graphical model representation of the UCLN model used in this exercise.

Create the Rev File

Open your text editor and create the uncorrelated-lognormal relaxed-clock model file called `m_UCLN_bears.Rev` in the **`RevBayes_scripts`** directory.

Enter the `Rev` code provided in this section in the new model file. Keep in mind that we are creating modular model files that can be sourced by different analysis files. Thus, the `Rev` code below will still depend on variable initialized in different files.

Independent Branch Rates

Before we can set up the variable of the branch-rate model, we must know how many branches exist in the tree.

```
n_branches <- 2 * n_taxa - 2
```

We will start with the mean of the lognormal distribution, M in Figure 8.3.

```
ucln_mean ~ dnExponential(2.0)
```

And the exponentially distributed node representing the standard deviation. We will also create a deterministic node, which is the variance, σ^2.

```
ucln_sigma ~ dnExponential(3.0)
ucln_var := ucln_sigma * ucln_sigma
```

Now we can declare the function that gives us the μ parameter of the lognormal distribution on branch rates.

```
ucln_mu := ln(ucln_mean) - (ucln_var * 0.5)
```

The only stochastic nodes we need to operate on for this part of the model are the lognormal mean (M or **`ucln_mean`**) and the standard deviation (σ or **`ucln_sigma`**).

```
moves[mi++] = mvScale(ucln_mean, lambda=1.0, tune=true, weight=4.0)
moves[mi++] = mvScale(ucln_sigma, lambda=0.5, tune=true, weight=4.0)
```

With our nodes representing the μ and σ of the lognormal distribution, we can create the vector of stochastic nodes for each of the branch rates using a **for** loop. Within this loop, we also add the move for each branch-rate stochastic node to our moves vector.

```
for(i in 1:n_branches){
   branch_rates[i] ~ dnLnorm(ucln_mu, ucln_sigma)
   moves[mi++] = mvScale(branch_rates[i], lambda=1, tune=true, weight=2.)
}
```

Sidebar: Other Uncorrelated-Rates Models

The choice in the branch-rate prior does not necessarily have to be a lognormal distribution. Depending on your prior beliefs about how branch rates vary across the tree, the rates can just as easily be assigned an exponential distribution (e.g., `branch_rates[i] ~ dnExponential(1.0)` defines each branch rate as an independent draw from an exponential distribution centered on 1) or a gamma distribution (e.g., `branch_rates[i] ~ dnGamma(2.0, 4.0)` defines each branch rate as an independent draw from a gamma distribution centered on 0.5) or any other distribution on positive-real numbers. The exercises outlined in this tutorial demonstrate how to compare different models of branch-rate variation using Bayes factors, and it may also be important to consider alternative priors on branch rates using these approaches. Importantly, `RevBayes` is flexible enough to make the process of comparing these models very straightforward. **For the purposes of this exercise, specify a lognormal prior on the branch rates.**

Because we are dealing with semi-identifiable parameters, it often helps to apply a range of moves to the variables representing the branch rates and branch times. This will help to improve the mixing of our MCMC. Here we will add 2 additional types of moves that act on vectors.

```
moves[mi++] = mvVectorScale(branch_rates,lambda=1.0,tune=true,weight=2.0)
moves[mi++] = mvVectorSingleElementScale(branch_rates,lambda=30.0,tune=true,weight
   =1.0)
```

The mean of the branch rates is a convenient deterministic node to monitor, particularly in the screen output when conducting MCMC.

```
mean_rt := mean(branch_rates)
```

The Sequence Model and Phylogenetic CTMC

Now, specify the stationary frequencies and exchangeability rates of the GTR matrix.

```
sf ~ dnDirichlet(v(1,1,1,1))
er ~ dnDirichlet(v(1,1,1,1,1,1))
Q := fnGTR(er,sf)
moves[mi++] = mvSimplexElementScale(er, alpha=10.0, tune=true, weight=3.0)
moves[mi++] = mvSimplexElementScale(sf, alpha=10.0, tune=true, weight=3.0)
```

Now, we can put the whole model together in the phylogenetic CTMC and clamp that node with our sequence data.

```
phySeq ~ dnPhyloCTMC(tree=timetree, Q=Q, branchRates=branch_subrates, nSites=n_sites,
    type="DNA")
attach the observed sequence data
phySeq.clamp(D)
```

Save and close the file called `m_UCLN_bears.Rev` in the **RevBayes_scripts** directory.

Estimate the Marginal Likelihood

Just as we did for the strict clock model, we can execute a power-posterior analysis to compute the marginal likelihood under the UCLN model.

Open your text editor and create the marginal-likelihood analysis file under the global molecular clock model. Call the file: `mlnl_UCLN_bears.Rev` and save it in the **RevBayes_scripts** directory.

Refer to the section describing this process for the GMC model above. Write your own **Rev** language script to estimate the marginal likelihood under the UCLN model. Be sure to change the file names in all of the relevant places (e.g., your output file for the **powerPosterior()** function should be `UCLN_bears_powp.out` and be sure to **source()** the correct model file `source("RevBayes_scripts/m_UCLN_bears.Rev")`).

Once you have completed this analysis, record the marginal likelihoods under the UCLN model in Table 10.1.

8.1.6 Compute Bayes Factors and Select Model

Now that we have estimates of the marginal likelihood under each of our different models, we can evaluate their relative plausibility using Bayes factors. Use Table 10.1 to summarize the marginal log-likelihoods estimated using the stepping-stone and path-sampling methods.

Phylogenetics software programs log-transform the likelihood to avoid underflow, because multiplying likelihoods results in numbers that are too small to be held in computer memory. Thus, we must calculate

Table 8.2: Estimated marginal likelihoods for different across-branch substitution rate models*.

	Marginal lnL estimates	
Clock Model	*Stepping-stone*	*Path sampling*
9.3.6 Global molecular clock (M_0)		
8.1.5 Uncorrelated lognormal (M_1)		

*you can edit this table

the ln-Bayes factor (we will denote this value $\mathcal{K}$):

$$\mathcal{K} = \ln[BF(M_0, M_1)] = \ln[\mathbb{P}(\mathbf{X} \mid M_0)] - \ln[\mathbb{P}(\mathbf{X} \mid M_1)], \tag{8.1}$$

where $\ln[\mathbb{P}(\mathbf{X} \mid M_0)]$ is the *marginal lnL* estimate for model M_0. The value resulting from equation 10.3 can be converted to a raw Bayes factor by simply taking the exponent of $\mathcal{K}$

$$BF(M_0, M_1) = e^{\mathcal{K}}. \tag{8.2}$$

Alternatively, you can interpret the strength of evidence in favor of M_0 using the $\mathcal{K}$ and skip equation 10.4. In this case, we evaluate the $\mathcal{K}$ in favor of model M_0 against model M_1 so that:

if $\mathcal{K} > 1$, then model M_0 wins
if $\mathcal{K} < -1$, then model M_1 wins.

Thus, values of $\mathcal{K}$ around 0 indicate ambiguous support.

Using the values you entered in Table 10.1 and equation 10.3, calculate the ln-Bayes factors (using $\mathcal{K}$) for the different model comparisons. Enter your answers in Table 8.3 using the stepping-stone and the path-sampling estimates of the marginal log likelihoods.

Table 8.3: Bayes factor calculation based on marginal likelihood estimates in Table 10.1*.

	ln-Bayes Factor ($\mathcal{K}$)	
Model comparison	*Stepping-stone*	*Path sampling*
M_0, M_1		
Supported model?		

*you can edit this table

8.1.7 Estimate the Topology and Branch Times

After computing the Bayes factors and determining the relative support of each model, you can choose your favorite model among the three tested in this tutorial. The next step, then, is to use MCMC to jointly estimate the tree topology and branch times.

Open your text editor and create the MCMC analysis file under the your favorite clock model. Call the file: `mcmc_TimeTree_bears.Rev` and save it in the **`RevBayes_scripts`** directory.

This file will contain much of the same initial `Rev` code as the files you wrote for the marginal-likelihood analyses.

```
### Load the sequence alignment
D <- readDiscreteCharacterData(file="data/bears_irbp.nex")

### get helpful variables from the data
n_sites <- D.nchar(1)

### initialize an iterator for the moves vector
mi = 1
```

This is how you should begin your MCMC analysis file. The next step is to source the birth-death model. However, if you're interested in estimating the tree topology, then you must add proposals that will do this. These moves can be added right after the birth-death model is sourced.

```
### set up the birth-death model from file
source("RevBayes_scripts/m_BDP_bears.Rev")

### and moves for the tree topology
moves[mi++] = mvNNI(timetree, weight=8.0)
moves[mi++] = mvNarrow(timetree, weight=8.0)
moves[mi++] = mvFNPR(timetree, weight=8.0)
```

Next load the file containing your favorite model (where the wildcard `*` indicates the name of the model you prefer: **`GMC`**, **`UCLN`**, or **`ACLN`**).

```
### load the model from file
source("RevBayes_scripts/m_*_bears.Rev")

### workspace model wrapper ###
mymodel = model(er)
```

MCMC Monitors

Before you instantiate the MCMC workspace object, you need to create a vector of "monitors" that are responsible for monitoring parameter values and saving those to file or printing them to the screen.

First, create a monitor of all the model parameters except the **`timetree`** using the model monitor: **`mnModel`**. This monitor takes *all* of the named parameters in the model DAG and saves their value to a file. Thus, every variable that you gave a name in your model files will be written to your log file. This makes it very easy to get an analysis going, but can generate very large files with a lot of redundant output.

```
monitors[1] = mnModel(filename="output/TimetTree_bears_mcmc.log", printgen=10)
```

If the model monitor is too verbose for your needs, you should use the file monitor instead: **mnFile**. For this monitor, you have to provide the names of all the parameters you're interested in after the file name and print interval. (Refer to the example files for how to set up the file monitor for model parameters.)

In fact, we use the file monitor for saving the sampled chronograms to file. It is important that you *do not* save the sampled trees in the same file with other numerical parameters you would like to summarize. That is because tools for reading MCMC log files—like Tracer (Rambaut and Drummond 2009)—cannot load files with non-numerical states. Therefore, you must save the sampled trees to a different file.

```
monitors[2] = mnFile(filename="output/TimeTree_bears_mcmc.trees", printgen=10,
    timetree)
```

Finally, we will create a monitor in charge of writing information to the screen: **mnScreen**. We will report the root age to the screen. If there is anything else you'd like to see in your screen output (e.g., the mean rate of the UCLN or ACLN model), feel free to add them to the list of parameters give to this model.

```
monitors[3] = mnScreen(printgen=10, root_time)
```

Setting-Up & Executing the MCMC

Now everything is in place to create the MCMC object in the workspace. This object allows you to perform a burn-in, execute a run of a given length, continue an analysis that might not have reached stationarity, and summarize the performance of the various proposals.

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

With this object instantiated, specify a burn-in period that will sample parameter space while re-tuning the proposals (only for the moves with **tune=true**). The monitors do not sample the states of the chain during burn-in.

```
mymcmc.burnin(generations=2000,tuningInterval=100)
```

Once the burn-in is complete, we want the analysis to run the full MCMC. Specify the length of the chain.

```
mymcmc.run(generations=5000)
```

When the MCMC run has completed, it's often good to evaluate the acceptance rates of the various proposal mechanisms. The **.operatorSummary()** member method of the MCMC object prints a table summarizing each of the parameter moves to the screen.

```
mymcmc.operatorSummary()
```

Summarize the Sampled Time-Trees

During the MCMC, the sampled trees will be written to a file that we will summarize using the **mapTree** function in **RevBayes**. This first requires that you add the code for reading in the tree-trace file and performing an analysis of those trees.

```
tt = readTreeTrace("output/TimeTree_bears_mcmc.trees", "clock")
tt.summarize()

### write MAP tree to file
mapTree(tt, "output/TimeTree_bears_mcmc_MAP.tre")
```

Save and close the file called **mcmc_TimeTree_bears.Rev** in the **RevBayes_scripts** directory. Then, execute the MCMC analysis using: **source("RevBayes_scripts/mcmc_TimeTree_bears.Rev")**

Bibliography

Abella, J., D. M. Alba, J. M. Robles, A. Valenciano, C. Rotgers, R. Carmona, P. Montoya, and J. Morales. 2012. *Kretzoiarctos* gen. nov., the oldest member of the giant panda clade. PLoS One 17:e48985.

Abella, J., P. Montoya, and J. Morales. 2011. Una nueva especie de *Agriarctos* (Ailuropodinae, Ursidae, Carnivora) en la localidad de Nombrevilla 2 (Zaragoza, España). Estudios Geológicos 67:187–191.

Clark, J. and T. E. Guensburg. 1972. Arctoid Genetic Characters as Related to the Genus *Paractis* vol. 1150. Field Museum of Natural History, Chicago, Ill.

dos Reis, M., J. Inoue, M. Hasegawa, R. Asher, P. Donoghue, and Z. Yang. 2012. Phylogenomic datasets provide both precision and accuracy in estimating the timescale of placental mammal phylogeny. Proceedings of the Royal Society B: Biological Sciences 279:3491–3500.

Heath, T. A., J. P. Huelsenbeck, and T. Stadler. 2014. The fossilized birth-death process for coherent calibration of divergence-time estimates. Proceedings of the National Academy of Sciences, USA 111:E2957–E2966.

Heled, J. and A. J. Drummond. 2012. Calibrated tree priors for relaxed phylogenetics and divergence time estimation. Systematic Biology 61:138–149.

Krause, J., T. Unger, A. Noçon, A.-S. Malaspinas, S.-O. Kolokotronis, M. Stiller, L. Soibelzon, H. Spriggs, P. H. Dear, A. W. Briggs, et al. 2008. Mitochondrial genomes reveal an explosive radiation of extinct and extant bears near the Miocene-Pliocene boundary. BMC Evolutionary Biology 8:220.

Rambaut, A. and A. J. Drummond. 2009. Tracer v1.5. Institute of Evolutionary Biology, University of Edinburgh, Edinburgh (United Kingdom) available from: http://beast.bio.ed.ac.uk/Tracer.

Wang, X. 1994. Phylogenetic Systematics of the Hesperocyoninae (Carnivora, Canidae) vol. 221.

Wang, X., R. H. Tedford, and B. E. Taylor. 1999. Phylogenetic Systematics of the Borophaginae (Carnivora, Canidae) vol. 243.

Warnock, R. C. M., Z. Yang, and P. C. J. Donoghue. 2012. Exploring the uncertainty in the calibration of the molecular clock. Biology Letters 8:156–159.

Yang, Z. and B. Rannala. 2006. Bayesian estimation of species divergence times under a molecular clock using multiple fossil calibrations with soft bounds. Molecular Biology and Evolution 23:212–226.

Chapter 9

Divergence Time Estimation

9.1 Overview

This tutorial demonstrates how to estimate divergence times and a dated phylogeny. The key concepts of this tutorial are the global molecular clock and node- and fossil-calibrations.

In this tutorial you will perform a Bayesian inference to estimate a time-calibrated phylogeny. In the first part we will demonstrate you how to set up a basic model for time-calibrated phylogeny inference. There we will use the global molecular clock rate to calibrate the phylogeny. The first analysis will use a fixed global molecular clock and the second analysis will use an informative prior distribution on the global molecular clock. In the second part you will use an informative prior distribution on the root age (crown age) to calibrate the phylogeny. In the third and final part you will use node- and fossil-calibrations instead of the informative prior on the root age. All the assumptions will be covered more in detail later in this tutorial.

Requirements

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

- RB_Getting_Started
- RB_Basics_Tutorial
- RB_CTMC_Tutorial

Note that the RB_Basics_Tutorial introduces the basic syntax of `Rev` but does not cover any phylogenetic models. You may skip the RB_Basics_Tutorial if you have some familiarity with `R`. We tried to keep this tutorial very basic and introduce all the language concepts on the way. You may only need the RB_Basics_Tutorial for a more in-depth discussion of concepts in `Rev`.

9.2 Data and files

We provide the data file(s) which we will use in this tutorial. You may want to use your own data instead. 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).

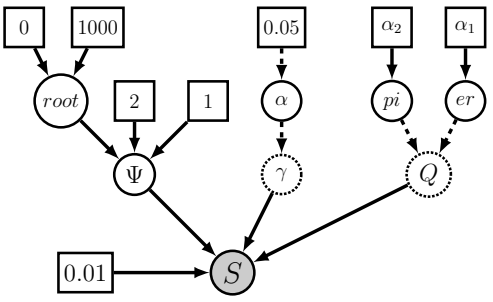
Table 9.1: Node information used for calibrating divergence times in the primate tree.

Clade	Age range (My)	Citation
Simiiformes	43 ± 4.5	
galagids and lorisids	40 ± 3	

9.3 Example: Divergence time estimation using a strict, global clock and an informative prior of the clock rate

9.3.1 Getting Started

The first section of this exercise involves: (1) setting up a general time reversible (GTR) substitution model with gamma distributed rate variation among sites for an alignment of the cytochrome b subunit; (2) use an informative prior on the clock rate to date the phylogeny; (3) approximating the posterior probability of the tree topology and node ages (and all other parameters) using MCMC, and; (4) summarizing the MCMC output by computing the maximum *a posteriori* tree. This analysis is mostly equivalent to the analysis performed in the RB_CTMC_Tutorial.




```
root ~ dnUniform(0,1000.0)
psi ~ dnBirthDeath(lambda=2.0, mu=1.0,
                   rootAge=root, taxon_names)


alpha1 <- v(1,1,1,1,1,1)
alpha2 <- v(1,1,1,1)
er ~ dnDirichlet( alpha1 )
pi ~ dnDirichlet( alpha2 )
Q_mol := fnGTR(er, pi)
alpha ~ dnExponential(0.05)
gamma := fnDiscretizeGamma( rate=alpha, shape=alpha, numCats=4 )


seq ~ dnPhyloCTMC( tree=psi, Q=Q_mol, siteRates=gamma,
                   branchRates=0.01, type="DNA" )
seq.clamp( data )
```

Figure 9.1: A simple phylogenetic model depicted in graphical-model notation (left) and the corresponding specification in the `Rev` language (right). In graphical-model notation, constant variables are enclosed in boxes, stochastic variables are enclosed in solid circles, deterministic variables in stippled circles, and observations in shaded circles, with arrows indicating variable dependencies. For example, the root age (*root*) is a random variable described by a uniform prior probability distribution with constant upper and lower bounds, the instantaneous-rate matrix, Q, is a deterministic function of the base frequencies and exchangeability rates (*pi* and *er*, respectively), and the observed sequences, S, are realizations of the phylogenetic model that are clamped for inference. This model is mirrored in `Rev` code, where the first two lines create the birth-death process (with fixed speciation and extinction rates; *lambda*=2.0 and *mu*=1.0), and a uniform prior distribution on the root age. The following three lines instantiate the instantaneous-rate matrix for the GTR model, where both the base frequencies and exchangeability rates are drawn from flat Dirichlet distributions. Finally, we create the stochastic variable representing the character data drawn from the Phylo-CTMC (continuous time Markov chain) process and attach (clamp) observations to the variable *seq*.

The general structure of the model is represented in Figure 9.1. This figure shows the full model graph.

9.3.2 Loading the Data

→ Download data and output files (if you don't have them already) from: http://revbayes.github.io/tutorials.html

First load in the sequences using the **`readDiscreteCharacterData()`** function.

```
data <- readDiscreteCharacterData("data/primates_cytb.nex")
```

Executing these lines initializes the data matrix as the respective `Rev` variable.

Next we will specify some useful variables based on our dataset. The variable **`data`** has *member functions* that we can use to retrieve information about the dataset. These include the number of species (**`n_species`**) and the tip labels (**`names`**). Each of these variables will be necessary for setting up different parts of our model (*e.g.,* the birth-death process prior).

```
n_species <- data.ntaxa()
names <- data.names()
```

Additionally, we set up a counter variable for the number of moves that we already added to our analysis. [Recall that moves are algorithms used to propose new parameter values during the MCMC simulation.] This will make it much easier if we extend the model or analysis to include additional moves or to remove some moves.

```
mi = 0
```

You may have noticed that we used the **`=`** operator to create the move index. This simply means that the variable is not part of the model. You will later see that we use this operator more often, *e.g.,* when we create moves and monitors.

With the data loaded, we can now proceed to specify our substitution model.

9.3.3 General Time Reversible (GTR) Substitution Model

The GTR model requires that we define and specify a prior on the six exchangeability rates, which we will describe using a flat Dirichlet distribution. As we did previously for the Dirichlet prior on base frequencies, we first define a constant node specifying the vector of concentration-parameter values using the **`v()`** function:

```
er_prior <- v(1,1,1,1,1,1)
```

This node defines the concentration-parameter values of the Dirichlet prior distribution on the exchangeability rates. Now, we can create a stochastic node for the exchangeability rates using the **`dnDirichlet()`** function, which takes the vector of concentration-parameter values as an argument and the **`~`** operator. Together, these create a stochastic node named **`er`**, see Figure 9.1:

```
er ~ dnDirichlet(er_prior)
```

The Dirichlet prior on our parameter **`er`** creates a *simplex* of values that sum to 1.

For each stochastic node in our model, we must also specify a proposal mechanism if we wish to estimate that parameter.

```
moves[++mi] = mvSimplexElementScale(er)
```

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:

```
pi_prior <- v(1,1,1,1)
pi ~ dnDirichlet(pi_prior)
```

The node **pi** represents the π node in Figure 9.1. Now add the simplex scale move on the stationary frequencies to the moves vector:

```
moves[++mi] = mvSimplexElementScale(pi)
```

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

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

9.3.4 Setting up the Gamma Model

Create a constant node called **shape_prior** for the rate parameter of the exponential prior on the gamma-shape parameter (this is represented as the constant rate parameter in Figure 9.1):

```
shape_prior <- 0.05
```

Then create a stochastic node called **alpha** with an exponential prior (this represents the stochastic node for the α-shape parameter in Figure 9.1):

```
alpha ~ dnExponential(shape_prior)
```

The way the ASRV model is implemented involves discretizing the mean-one gamma distribution into a set number of rate categories, k. Thus, we can analytically marginalize over the uncertainty in the rate at each site. The likelihood of each site is averaged over the k rate categories, where the rate multiplier is the mean (or median) of each of the discrete k categories. To specify this, we need a deterministic node that is a vector that will hold the set of k rates drawn from the gamma distribution with k 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 **alpha** for both the shape and rate.

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

```
gamma_rates := fnDiscretizeGamma( alpha, alpha, 4 )
```

Note that here, by convention, we set $k = 4$. The random variable that controls the rate variation is the stochastic node **alpha**. It may be useful to look at the values of the gamma rates directly to get an intuition of the different values for the rate categories. We will apply a simple scale move to this parameter.

```
moves[++mi] = mvScale(alpha, weight=2.0)
```

9.3.5 Tree Prior: Tree Topology and Node Ages

The tree (the topology and node ages) is a stochastic node in our phylogenetic model. In Figure 7.1, the tree is denoted Ψ.

We will assume a constant-rate birth-death process as the prior distribution on the tree. This means that all possible labeled, rooted tree topologies have equal probability. The distribution in `RevBayes` is **dnBDP()**. For the birth-death process we need a speciation rate and extinction rate parameter. Let us start with those two variables. We use a *gamma* distribution with rate $a = 5$ and shape $b = 1$ for both the **diversification** and **turnover** variables.

```
a <- 5
b <- 1
diversification ~ dnGamma(shape=a, rate=b)
c <- 5
d <- 1
turnover ~ dnGamma(shape=c,rate=d)
```

Now we can transform the **diversification** and **turnover** into the **speciation** rate and **extinction** rate.

```
speciation := diversification + turnover
extinction := turnover
```

We also need to specify a prior on the root age (our informed guess is about 75-80 mya). So we use a uniform distribution between 50 and 100.

```
root ~ dnUniform(0.0,1000.0)
```

Additionally, we know that we do not have all primate species included in this data set. We only have 23 out of the approximately 450 primate species. Thus, we use a sampling fraction to represent this incomplete taxon sampling.

```
sampling_fraction <- 23 / 450
```

Here we have created our first three stochastic variables. For each one of them we need to create at least one moves that change the stochastic variables. In this case we use sliding window proposals but you could use scaling proposals for the rates too.

```
moves[++mi] = mvSlide(diversification,delta=1,tune=true,weight=1)
moves[++mi] = mvSlide(turnover,delta=1,tune=true,weight=1)
moves[++mi] = mvSlide(root,delta=1,tune=true,weight=1)
```

Next, specify the **tree** stochastic node by passing in the tip labels **names** to the **dnBDP()** distribution:

```
psi ~ dnBDP(lambda=speciation, mu=extinction, rootAge=abs(root), rho=sampling_fraction
    , nTaxa=n_species, names=names )
```

Some types of stochastic nodes can be updated by a number of alternative moves. Different moves may explore parameter space in different ways, and it is possible to use multiple different moves for a given parameter to improve mixing (the efficiency of the MCMC simulation). In the case of our rooted tree, for example, we can use both a nearest-neighbor interchange move without and with changing the node ages (**mvNarrow** and **mvNNI**) and a fixed-nodeheight subtree-prune and regrafting move (**mvFNPR**). We also need moves that change the ages of the internal nodes; which are for example the **mvSubtreeScale** and **mvNodeTimeSlideUniform**. These moves do not have tuning parameters associated with them, thus you only need to pass in the **psi** node and proposal **weight**.

```
moves[++mi] = mvNarrow(psi, weight=5.0)
moves[++mi] = mvNNI(psi, weight=1.0)
moves[++mi] = mvFNPR(psi, weight=3.0)
moves[++mi] = mvSubtreeScale(psi, weight=3.0)
moves[++mi] = mvNodeTimeSlideUniform(psi, weight=15.0)
```

The weight specifies how often the move will be applied either on average per iteration or relative to all other moves. Have a look at the MCMC tutorial for more details about moves and MCMC strategies: http://revbayes.github.io/tutorials.html

9.3.6 The Global Molecular Clock Model

The global molecular clock assumes that the rate of substitution is constant over the tree and over time (Fig. 8.2).

Here we use an informative prior on the clock rate. In several studies it has been suggested that the rate of mitochondrial evolution is about 0.01 (=1%) per million years per site. Thus, we will specify a narrow prior distribution centered around 0.01.

```
clock_M <- 0.01
clock_s <- 1
clock_mu <- log(clock_M) - ((clock_s * clock_s) * 0.5)

clockRate ~ dnLognormal(clock_mu, clock_s)
```

As usual, we need a move on the stochastic variable, the **clockRate**. We use a scaling move because this is a rate parameter.

```
moves[++mi] = mvScale(clockRate,lambda=1,tune=true,weight=1)
```

9.3.7 Putting it All Together

We have fully specified all of the parameters of our phylogenetic model—the tree topology with branch lengths, and the substitution model that describes how the sequence data evolved over the tree with branch lengths. Collectively, these parameters comprise a distribution called the *phylogenetic continuous-time Markov chain*, and we use the **PhyloCTMC** constructor function to create this node. This distribution requires several input arguments: (1) the **tree** with branch lengths; (2) the instantaneous-rate matrix **Q**; (3) the clock rate, and; (4) the **type** of character data.

Build the random variable for the character data (sequence alignment).

```
# the sequence evolution model
seq ~ dnPhyloCTMC(tree=psi, Q=Q, branchRates=clockRate, siteRates=gamma_rates, type="
    DNA")
```

Once the **PhyloCTMC** model has been created, we can attach our sequence data to the tip nodes in the tree.

```
seq.clamp(data)
```

[Note that although we assume that our sequence data are random variables—they are realizations of our phylogenetic model—for the purposes of inference, we assume that the sequence data are "clamped".]

When this function is called, `RevBayes` sets each of the stochastic nodes representing the tips of the tree to the corresponding nucleotide sequence in the alignment. This essentially tells the program that we have observed data for the sequences at the tips.

Finally, we wrap the entire model to provide convenient access to the DAG. To do this, we only need to give the **`model()`** function a single node. With this node, the **`model()`** function can find all of the other nodes by following the arrows in the graphical model:

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

9.3.8 Performing an MCMC Analysis Under the Global Clock Model

In this section, will describe how to set up the MCMC sampler and summarize the resulting posterior distribution of trees.

Specifying Monitors

For our MCMC analysis, we need to set up a vector of *monitors* to record 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[1] = mnModel(filename="output/primates_cytb_global_clock.log",printgen=10,
    separator = TAB)
```

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[2] = mnFile(filename="output/primates_cytb_global_clock.trees",printgen=10,
    separator = TAB, psi)
```

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

```
monitors[3] = mnScreen(printgen=1000, clockRate)
```

Initializing and Running the MCMC Simulation

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 may wish to run the **.burnin()** member function. Recall that this function **does not** specify the number of states that we wish to discard from the MCMC analysis as burnin (i.e., the samples collected before the chain converges to the stationary distribution). Instead, the **.burnin()** function specifies a *completely separate* preliminary MCMC analysis that is used to tune the scale of the moves to improve mixing of the MCMC analysis.

```
mymcmc.burnin(generations=10000,tuningInterval=1000)
```

Now, run the MCMC:

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

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

→ Look at the file called **output/primates_cytb_global_clock.log** in Tracer.

9.3.9 Exercise 1

We are interested in the divergence time estimate between *New World Monkeys* and *Old World Monkeys*, which is the infraorder *Simiiformes*. We have provided the Table 9.2 of the primates species and their relationships for you convenience (if you forgot which primate species belongs to either of the two groups). To obtain an estimate of the divergence time we read in the tree trace and build the annotated maximum *a posteriori* tree.

```
treetrace = readTreeTrace("output/primates_cytb_global_clock.trees",
                          treetype="clock")
mapTree(treetrace,"output/primates_cytb_global_clock.tree")
```

Fill in the following table as you go through the tutorial.

→ Look at the file called **output/primates_cytb_global_clock.tree** in FigTree.

- Add a deterministic node monitoring the age of the *Simiiformes*:

Table 9.2: Primate species and famaly relationships.

Species	**Family**	**Parvorder**	**Suborder**
Alouatta palliata	Atelidae	Platyrrhini (NWM)	Haplorrhini
Aotus trivirgatus	Aotidae	Platyrrhini (NWM)	Haplorrhini
Callicebus donacophilus	Pitheciidae	Platyrrhini (NWM)	Haplorrhini
Cebus albifrons	Cebidae	Platyrrhini (NWM)	Haplorrhini
Cheirogaleus major	Cheirogaleidae	Lemuroidea	Strepsirrhini
Chlorocebus aethiops	Cercopithecoidea	Catarrhini	Haplorrhini
Colobus guereza	Cercopithecoidea	Catarrhini	Haplorrhini
Daubentonia madagascariensis	Daubentoniidae	Lemuroidea	Strepsirrhini
Galago senegalensis	Galagidae	Lorisidae	Strepsirrhini
Hylobates lar	Hylobatidea	Catarrhini	Haplorrhini
Lemur catta	Lemuridae	Lemuroidea	Strepsirrhini
Lepilemur hubbardorum	Lepilemuridae	Lemuroidea	Strepsirrhini
Loris tardigradus	Lorisidae	Lorisidae	Strepsirrhini
Macaca mulatta	Cercopithecoidea	Catarrhini	Haplorrhini
Microcebus murinus	Cheirogaleidae	Lemuroidea	Strepsirrhini
Nycticebus coucang	Lorisidae	Lorisidae	Strepsirrhini
Otolemur crassicaudatus	Galagidae	Lorisidae	Strepsirrhini
Pan paniscus	Hominoidea	Catarrhini	Haplorrhini
Perodicticus potto	Lorisidae	Lorisidae	Strepsirrhini
Propithecus coquereli	Indriidae	Lemuroidea	Strepsirrhini
Saimiri sciureus	Cebidae	Platyrrhini (NWM)	Haplorrhini
Tarsius syrichta	Tarsiidae		Haplorrhini
Varecia variegata variegata	Lemuridae	Lemuroidea	Strepsirrhini

Table 9.3: Estimated divergence times of the infraorder *Simiiformes**.

	Divergence time estimates		
Clock Model	*Mean Estimate*		*Credible interval*
9.3.6 Global molecular clock			
?? Root calibration (M_1)			
?? Node calibration (M_2)			

*you can edit this table

```
clade_simiiformes = clade("Cebus_albifrons", "Macaca_mulatta")
age_simiiformes := tmrca(psi, clade_simiiformes)
```

- Experiment with different prior distributions on the clock rate, *e.g.,* a uniform distribution between 0.005 and 0.02.

9.4 Root calibration

In the previous section we calibrated the phylogeny using an informative prior on the clock rate. Unfortunately, we don't always have a good estimate of the clock rate. A very common alternative approach is to use an informative prior on the root age. For example, another analysis might have dated the crown age of your group of interested.

In our reference publication, **?** used a normal distribution with mean of 90.0 MYA with stdev = 6.0 as the prior distribution on the root age.

```
root ~ dnNormal(90.0,6.0)
```

```
rootAge := abs(root)
```

Otherwise we will use the same priors and parameters for the tree prior, *e.g.,* a constant-rate birth-death process.

```
a <- 5
b <- 1
diversification ~ dnGamma(shape=a, rate=b)
c <- 5
d <- 1
turnover ~ dnGamma(shape=c,rate=d)

speciation := diversification + turnover
extinction := turnover

sampling_fraction <- 23 / 270

moves[++mi] = mvSlide(diversification,delta=1,tune=true,weight=1)
moves[++mi] = mvSlide(turnover,delta=1,tune=true,weight=1)
moves[++mi] = mvSlide(root,delta=1,tune=true,weight=1)
```

Next, specify the **tree** stochastic node by passing in the tip labels **names** to the **dnBDP()** distribution:

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

moves[++mi] = mvNarrow(psi, weight=5.0)
moves[++mi] = mvNNI(psi, weight=1.0)
moves[++mi] = mvFNPR(psi, weight=3.0)
moves[++mi] = mvSubtreeScale(psi, weight=3.0)
moves[++mi] = mvNodeTimeSlideUniform(psi, weight=15.0)
```

→ Don't forget to change the output file names, otherwise your old analyses files will be overwritten.

9.4.1 Exercise 2

- Run the analysis and fill in the table with the age estimates.
- Experiment again with different prior choices. What impact do you see of your choices on the prior distribution?

9.5 Node calibration

This part of the exercise will involve specifying a birth-death model with clamped stochastic nodes representing the prior information on internal nodes in our tree: (1) *Semiiformes*, the split between *New World Monkeys* and *OldWorld Monkeys*, and (2) *galagids and lorisids*, the split between *galagids* and *lorisids*.

In `RevBayes`, calibrated internal nodes are treated differently than in many other programs for estimating species divergence times (e.g., BEAST). This is because the graphical model structure used in `RevBayes` does not allow a stochastic node to be assigned more than one prior distribution. By contrast, the common approach to applying calibration densities as used in other dating softwares leads to incoherence in the calibration prior (for detailed explainations of this see **???**). More explicitly, common calibration approaches assume that the age of a calibrated node is modeled by the tree-wide diversification process (e.g., birth-death model) *and* a parametric density parameterized by the occurrence time of a fossil (or other external prior information). This can induce a calibration prior density that is not consistent with the birth-death process or the parametric prior distribution. Thus, approaches that condition the birth-death process on the calibrated nodes are more statistically coherent (**?**).

In `RevBayes`, calibration densities are applied in a different way, treating fossil observation times like data. The graphical model in Figure 9.2 illustrates how calibrated nodes are specified in the directed acyclic graph (DAG). Here, the age of the calibration node (i.e., the internal node specified as the MRCA of the fossil and a set of living species) is a deterministic node—e.g., denoted o_1 for fossil $\mathcal{F}_1$—and acts as an offset on the stochastic node representing the age of the fossil specimen. The fossil age, $\mathcal{F}_i$, is specified as a stochastic node and clamped to its *observed* age in the fossil record. The node $\mathcal{F}_i$ is modeled using a distribution that describes the waiting time from the speciation event to the appearance of the observed fossil. Thus, if the MCMC samples any state of Ψ for which the age of $\mathcal{F}_i$ has a probability of 0, then that state will always be rejected, effectively calibrating the birth-death process without applying multiple prior densities to any calibrated node (Fig. 9.2).

The root age is treated differently, however. Here, we condition the birth-death process on the speciation time of the root, thus this variable is not part of the time-tree parameter. The root age can thus be given any parametric distribution over positive real numbers (Fig. 9.2).

First, we specify the calibration for the split between *New World Monkeys* and *OldWorld Monkeys*

```
clade_simiiformes = clade("Cebus_albifrons", "Macaca_mulatta")
age_simiiformes := tmrca(psi, clade_simiiformes)
obs_age_simiiformes ~ dnNormal(age_simiiformes,4.5)
obs_age_simiiformes.clamp(43)
```

Next, we specify the calibration for the split between *galagids* and *lorisids*

```
clade_galago_loris = clade("Loris_tardigradus", "Galago_senegalensis")
age_galago_loris := tmrca(psi, clade_galago_loris)
obs_age_galago_loris ~ dnNormal(age_simiiformes,3)
obs_age_galago_loris.clamp(40)
```

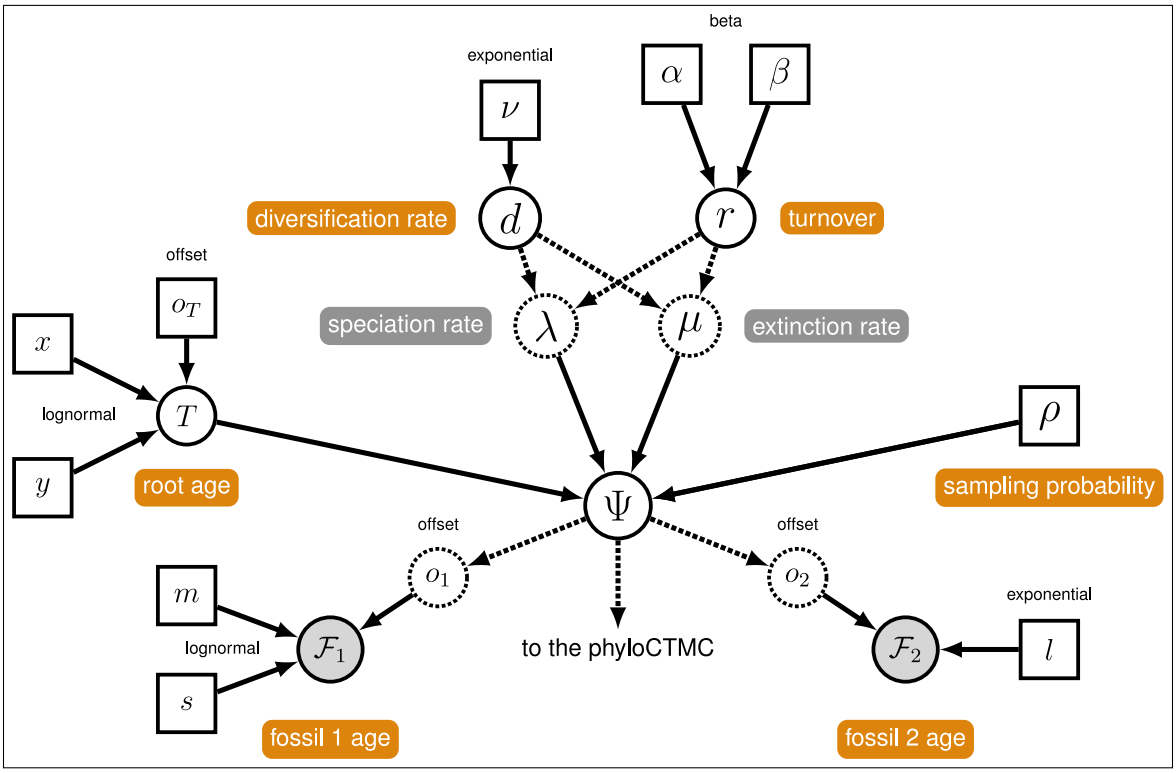


Figure 9.2: The graphical model representation of the node-calibrated birth-death process in `RevBayes`.

9.5.1 Exercise 3

- Make the nexessary modifications and runs the analysis.
- Fill in the table.
- Compare the divergence time estimates with the calibration prior.
- Try the **`dnSoftBoundUniformNormal`** instead of a normal distribution as a prior calibration density.

Bibliography

Part V

Diversification Rate Estimation

Chapter 10

Simple Diversification Rate Estimation

10.1 Exercise: Estimating Speciation & Extinction Rates

10.1.1 Introduction

Models of speciation and extinction are fundamental to any phylogenetic analysis of macroevolutionary processes. A prior describing the distribution of speciation events over time is critical to estimating phylogenies with branch lengths proportional to time. Moreover, stochastic branching models allow for inference of speciation and extinction rates. These inferences allow us to investigate key questions in evolutionary biology.

This tutorial describes how to specify simple branching-process models in `RevBayes`; two variants of the constant-rate birth-death process (Yule 1924; Kendall 1948a; Thompson 1975; Nee et al. 1994; Rannala and Yang 1996; Yang and Rannala 1997). The probabilistic graphical model is given for each component of this exercise. After each model is specified, you will estimate the marginal likelihood of the model and evaluate the relative support using Bayes factors. Finally, you will estimate speciation and extinction rates using Markov chain Monte Carlo (MCMC) under the model supported by the data.

10.1.2 The "Observed" Data

- Download the **`#NEXUS`** tree file from: http://bit.ly/1e5PwdW

- Create a directory called **`data`** and move the tree file **`bears_dosReis.nex`** into that directory

The tree in this exercise contains a subset of the taxa resulting from the divergence-time analysis of 274 placental mammal species by dos Reis et al. (2012). The tree comprises all living bear species (8 taxa) plus two outgroups—the gray wolf and spotted seal. Thus, the root of this tree represents the most-recent common ancestor of all living members of the suborder Caniformia (Fig. 10.1). The phylogenetic relationships and speciation times are the median estimates reported by dos Reis et al. (2012). In this exercise, this time tree is treated as an "observation", and we are estimating parameters of the birth-death model without accounting for uncertainty in the time tree.

- Open the tree **`data/bears_dosReis.nex`** in FigTree.

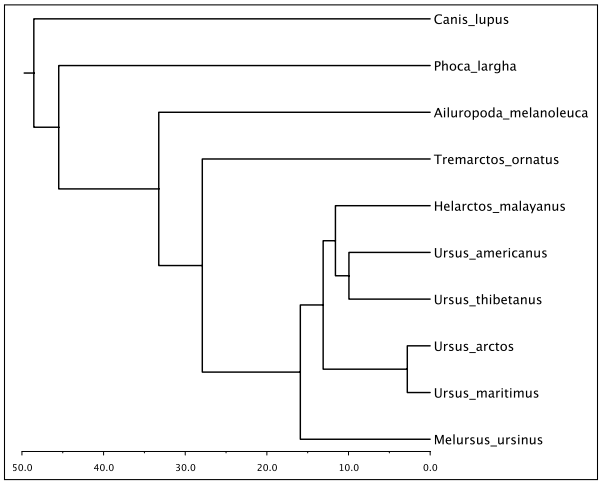


Figure 10.1: The relationships and median speciation times for bears and two outgroups estimated in the analysis by dos Reis et al. (2012).

10.2 Specifying Constant-Rate Birth-Death Models

Before evaluating the relative support for different models, we must first specify them in the `Rev` language. In this section, we will walk through specifying three different variants of the birth-death process model and estimating the marginal likelihood under each one.

10.2.1 Pure-Birth Model

The simplest branching model is the *pure-birth process* described by Yule (1924). Under this model, we assume at any instant in time, every lineage has the same speciation rate λ. In its simplest form, the speciation rate remains constant over time. As a result, the waiting time between speciation events is exponential, where the rate of the exponential distribution is the product of the number of extant lineages (n) at that time and the speciation rate: $n\lambda$ (Yule 1924; Aldous 2001; Hartmann et al. 2010). The pure-birth branching model does not allow for lineage extinction (this is similar to population-level coalescent models). However, the model depends on a second parameter ρ which is the probability of sampling a species in the present time as well as the time of the start of the process, whether that is the origin time or root age. Therefore, the probabilistic graphical model of the pure-birth process is quite simple, where the observed time tree topology and node ages are conditional on the speciation rate, sampling probability, and root age (Fig. 10.2).

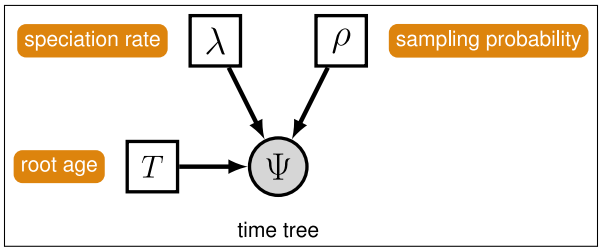


Figure 10.2: The graphical model representation of the pure-birth (Yule) process.

We can add hierarchical structure to this model and account for uncertainty in the value of the speciation rate by placing a hyperprior on λ (Fig. 10.3). The graphical models in Figures 10.2 and 10.3 demonstrate the simplicity of the Yule model. Ultimately, the pure birth model is just a special case of the birth-death process, where the extinction rate (typically denoted μ) is a constant node with the value 0.

For this exercise, we will specify a Yule model, such that the speciation rate is a stochastic node, drawn from an exponential distribution as in Figure 10.3. In a Bayesian framework, we are interested in estimating the posterior probability of λ given that we observe a time tree.

$$\mathbb{P}(\lambda \mid \Psi) = \frac{\mathbb{P}(\Psi \mid \lambda)\mathbb{P}(\lambda \mid \nu)}{\mathbb{P}(\Psi)} \tag{10.1}$$

In this example, we have a phylogeny of all living bears plus two outgroup species, the gray wolf and spotted seal. We are treating the time tree Ψ as an observation, thus clamping the model with an observed value. The time tree we are conditioning the process on is taken from the analysis by dos Reis et al. (2012) and shown in Figure 10.1. Furthermore, there are approximately 147 described caniform species, so we will fix the parameter ρ to 10/147.

- The full Yule-model specification is in the file called `m_Yule_bears.Rev` on the `RevBayes` tutorial

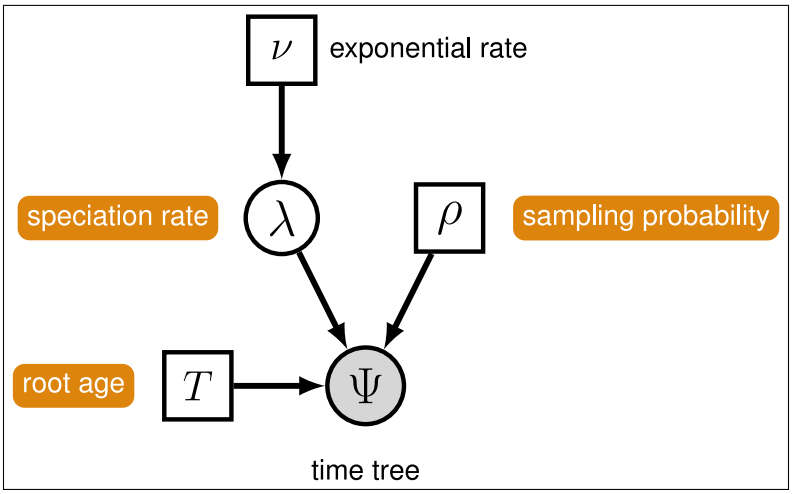


Figure 10.3: The graphical model representation of the pure-birth (Yule) process, where the speciation rate is treated as a random variable drawn from an exponential distribution with rate parameter ν.

repository.

Read the tree

Begin by reading in the observed tree from Figure 10.1.

```
T <- readTrees("data/bears_dosReis.tre")[1]
```

From this tree, we can get some helpful variables:

```
n_taxa <- T.ntips()
names <- T.names()
```

Additionally, we can initialize an iterator variable for our vector of moves:

```
mi = 1
```

Birth rate

The model we are specifying only has three nodes (Fig. 10.3). We can specify the birth rate λ, the rate-parameter ν of the exponential hyperprior on λ, and the conditional dependency of the two parameters all in one line of `Rev` code.

```
birth_rate ~ dnExponential(0.1)
```

Here, the stochastic node called **birth_rate** represents λ and the **0.1** is the constant node ν, given the value 0.1. Note that this value leads to an expected value for λ of 10:

$$\mathbb{E}[\lambda] = \nu^{-1} = 10$$

To estimate the value of λ, we assign a proposal mechanism to operate on this node. In **RevBayes** these MCMC sampling algorithms are called *moves*. We need to create a vector of moves and we can do this by using vector indexing and our pre-initialized iterator **mi**. We will use a scaling move on λ called **mvScale**.

```
moves[mi++] = mvScale(birth_rate,lambda=1,tune=true,weight=3)
```

Sampling probability

Our prior belief is that we have sampled 10 out of 147 living caniform species. To account for this we can set the sampling parameter as a constant node with a value of 0.068

```
rho <- 0.068
```

Root age

Any stochastic branching process must be conditioned on a time that represents the start of the process. Typically, this parameter is the *origin time* and it is assumed that the process started with *one* lineage. Thus, the origin of a birth-death process is the node that is *ancestral* to the root node of the tree. For macroevolutionary data, particularly without any sampled fossils, it is difficult to use the origin time. To accommodate this, we can condition on the age of the root by assuming the process started with *two* lineages that both originate at the time of the root.

We can get the value for the root from the dos Reis et al. (2012) tree.

```
root_time <- treeHeight(T)
```

The time tree

Now we have all of the parameters we need to specify the full pure-birth model. We can initialize the stochastic node representing the time tree. Note that we set the **mu** parameter to the constant value **0.0**.

```
timetree ~ dnBDP(lambda=birth_rate, mu=0.0, rho=rho, rootAge=root_time,
    samplingStrategy="uniform", condition="nTaxa", nTaxa=n_taxa, names=names)
```

If you refer back to Equation 10.1 and Figure 10.3, the time tree Ψ is the variable we observe, i.e., the data. We can set this in the **Rev** language by using the **clamp()** function.

```
timetree.clamp(T)
```

Here we are fixing the value of the time tree to our observed tree from dos Reis et al. (2012). If we did not clamp this node, and ran MCMC, we would simply simulate time trees under the model.

Finally, we can create a workspace object of our whole model using the **model()** function. Workspace objects are initialized using the **=** operator. This distinguishes the objects used by the program to run the MCMC analysis from the distinct nodes of our graphical model. The model workspace objects makes it easy to work with the model in the **Rev** language and creates a wrapper around our model DAG. Because our model is a directed, acyclic graph (DAG), we only need to give the model wrapper function a single node and it does the work to find all the other nodes through their connections.

```
mymodel = model(birth_rate)
```

The **model()** function traversed all of the connections and found all of the nodes we specified. We can now visualize our graphical model using the **.graph()** member method of the model object. This function writes a file in the DOT graph-description language. The contents of this file describes the nodes and edges of the model DAG and can be read by an interpreter program called Graphviz. First create the model graph file using the **.graph()** method. Set the flag for extra output (good for development debugging) to false: **verbose=false**. And specify a named RBG color for the background (for this graph, we like **"honeydew2"**).

```
mymodel.graph("output/m_Yule_bears_GM.dot", verbose=false, bg="honeydew2")
```

Open the **output/m_Yule_bears_GM.dot** file in the Graphviz program or paste the contents in an online viewer:

- http://graphviz-dev.appspot.com
- http://stamm-wilbrandt.de/GraphvizFiddle

Your graph should look like the one depicted in Figure 10.4. Compare this figure to the model in Figure 10.3. You should notice that there are two extra nodes in the Figure 10.4. The constant node with the value **0** connected to the **birth_rate** stochastic node represents the **offset** variable of the exponential distribution which is given the default value of 0 when no offset is provided. Additionally, there is a nameless constant node with the value of **0** pointing into the clamped **timetree** stochastic node. This constant node represents the death rate, **mu**, which we set to **0** when we initialized **timetree** using the **dnBDP()** constructor function. Viewing the model graph is helpful for identifying any problems prior to running MCMC.

Estimating the marginal likelihood of the model

With a fully specified model, we can set up the **powerPosterior()** analysis to create a file of 'powers' and likelihoods from which we can estimate the marginal likelihood using stepping-stone or path sampling.

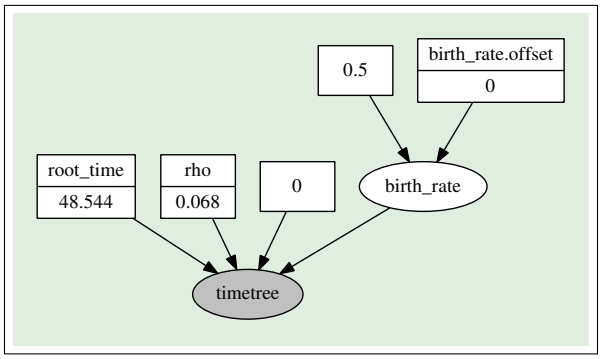


Figure 10.4: The graphical model representation of the pure-birth (Yule) process generated using the DOT language and the Graphviz program.

This method computes a vector of powers from a beta distribution, then executes an MCMC run for each power step while raising the likelihood to that power. In this implementation, the vector of powers starts with 1, sampling the likelihood close to the posterior and incrementally sampling closer and closer to the prior as the power decreases.

- The `Rev` file for performing this analysis: `mlnl_Yule_bears.Rev`.

First, we create the variable containing the power posterior. This requires us to provide a model and vector of moves, as well as an output file name. The **`cats`** argument sets the number of power steps.

```
pow_p = powerPosterior(mymodel, moves, "output/Yule_bears_powp.out", cats=50)
```

We can start the power posterior by first burning in the chain and and discarding the first 10000 states.

```
pow_p.burnin(generations=10000,tuningInterval=1000)
```

Now execute the run with the **`.run()`** function:

```
pow_p.run(generations=1000)
```

Once the power posteriors have been saved to file, create a stepping stone sampler. This function can read any file of power posteriors and compute the marginal likelihood using stepping-stone sampling.

```
ss = steppingStoneSampler(file="output/Yule_bears_powp.out", powerColumnName="power",
    likelihoodColumnName="likelihood")
```

Compute the marginal likelihood under stepping-stone sampling using the member function **`marginal()`** of the **`ss`** variable and record the value in Table 10.1.

```
ss.marginal()
```

Path sampling is an alternative to stepping-stone sampling and also takes the same power posteriors as input.

```
ps = pathSampler(file="output/Yule_bears_powp.out", powerColumnName="power",
    likelihoodColumnName="likelihood")
```

Compute the marginal likelihood under stepping-stone sampling using the member function **marginal()** of the **ps** variable and record the value in Table 10.1.

```
ps.marginal()
```

10.3 Birth-Death Process

The pure-birth model does not account for extinction, thus it assumes that every lineage at the start of the process will have sampled descendants at time 0. This assumption is fairly unrealistic for most phylogenetic datasets on a macroevolutionary time scale since the fossil record provides evidence of extinct lineages. Kendall (1948b) described a more general branching process model to account for lineage extinction called the *birth-death process*. Under this model, at any instant in time, every lineage has the same rate of speciation λ and the same rate of extinction μ. This is the *constant-rate* birth-death process, which considers the rates constant over time and over the tree (Nee et al. 1994). Importantly, this model assumes that all of the extant descendants of the process have been sampled at time 0.

Yang and Rannala (1997) derived the probability of time trees under an extension of the birth-death model that accounts for incomplete sampling of the tips (Fig. 10.5). Under this model, the parameter ρ accounts for the probability of sampling in the present time, and because it is a probability, this parameter can only take values between 0 and 1.

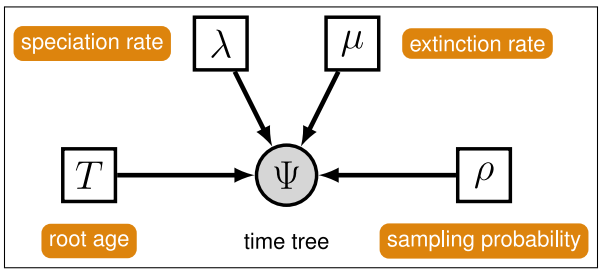


Figure 10.5: The graphical model representation of the birth-death process with uniform sampling and conditioned on the root age.

Ultimately, it is difficult to formulate prior densities on rate parameters, particularly when our uncertainty in the values of the speciation and extinction rates is quite large. Furthermore, without sampling the

process back in time, it is difficult to estimate extinction. Thus, we can re-parameterize the birth-death process to account for these issues. In this parameterization, we use the net diversification rate d and the turnover rate r (also called relative extinction rate) instead of the λ, μ parameters.

$$\begin{aligned} d &= \lambda - \mu && \text{Net diversification rate} \\ r &= \mu/\lambda && \text{Turnover} \end{aligned}$$

Importantly, we can recover λ and μ via:

$$\lambda = \frac{d}{1-r}, \quad \mu = \frac{rd}{1-r}. \tag{10.2}$$

Thus, λ and μ are deterministic nodes, transformed from d and r. By using the diversification and turnover parameters, we now have another variable, r that can only take values between 0 and 1. This is because, under the constant-rate birth-death process, μ can never be greater than λ (Fig. 10.6). Note that if $\mu = 0$, then $d = \lambda$ in this parameterization.

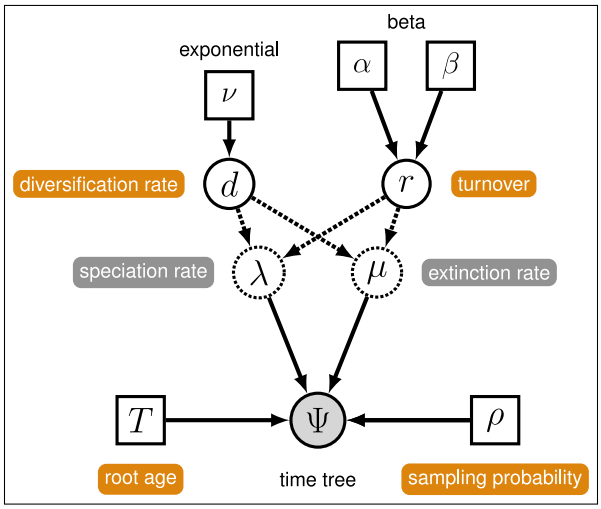


Figure 10.6: The graphical model representation of the birth-death process with uniform sampling parameterized using the diversification and turnover.

In this model, we will specify an exponential prior density on d and a beta prior on r. There are approximately 147 described caniform species, so we will fix the parameter ρ to 10/147.

- The full birth-death, fixed sampling model specification is in the file called `m_BD_bears.Rev`.

10.3.1 Clear the workspace and read the tree

It is best to remove all of the previous model variables created in the previous section.

```
clear()
```

Now read in the observed tree from Figure 10.1.

```
T <- readTrees("data/bears_dosReis.tre")[1]
```

Initialize the useful variables:

```
n_taxa <- T.ntips()
names <- T.names()
mi = 1
```

10.3.2 Diversification and turnover

The diversification and turnover are the parameters which we will treat as stochastic nodes in our model. We will assume an exponential prior on **diversification** and assign it a scale move.

```
diversification ~ dnExponential(10.0)
moves[mi++] = mvScale(diversification,lambda=1.0,tune=true,weight=3.0)
```

The **turnover** parameter can only take values between 0 and 1, thus we will assume a beta prior on this parameter and sample from the posterior distribution using a scale move.

```
turnover ~ dnBeta(2.0, 2.0)
moves[mi++] = mvSlide(turnover,delta=1.0,tune=true,weight=3.0)
```

10.3.3 Birth rate and death rate

The birth and death rates are both deterministic nodes. Refer to Equation 10.2. Note that both the birth rate and death rate are functions of d and r.

Because our variable transformations use the **-** operator, we must additionally use the **abs()** function to ensure that the rates are of type **RealPos**, which is required by the birth-death process model.

```
birth_rate := abs(diversification / (1.0 - turnover))
```

```
death_rate := abs(turnover * diversification / (1.0 - turnover))
```

10.3.4 The sampling probability

If we assume that the 147 described caniform species represent all of the living caniforms on Earth, then it is quite reasonable to fix the parameter ρ to a known value.

```
rho <- 0.068
```

10.3.5 Root age

Get the value for the root from the dos Reis et al. (2012) tree.

```
root_time <- treeHeight(T)
```

10.3.6 The time tree

Initialize the stochastic node representing the time tree.

```
timetree ~ dnBDP(lambda=birth_rate, mu=death_rate, rootAge=root_time, rho=rho,
    samplingStrategy="uniform", condition="nTaxa", nTaxa=n_taxa, names=names)
```

Since we are computing the likelihood on the dos Reis et al. (2012) tree, we can consider this time tree as an observation and clamp the stochastic node.

```
timetree.clamp(T)
```

Now set the workspace model variable.

```
mymodel = model(diversification)
```

```
mymodel.graph("output/m_BD_bears_GM.dot", bg="LightSteelBlue2")
```

10.3.7 Estimating the marginal likelihood of the model

Next, we will set up the file for running the power posteriors and computing the marginal likelihoods.

- The `Rev` file for performing this analysis: `mlnl_BD_bears.Rev`.

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

Compute the marginal likelihood under stepping-stone sampling using the member function **`marginal()`** of the **`ss`** variable and record the value in Table 10.1.

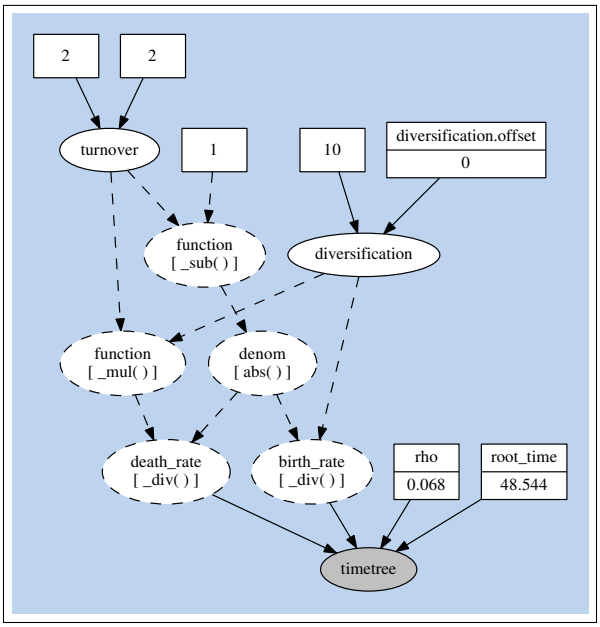


Figure 10.7: The graphical model representation of the birth-death process.

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

Compute the marginal likelihood under stepping-stone sampling using the member function **marginal()** of the **ps** variable and record the value in Table 10.1.

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

10.4 Compute Bayes Factors and Select Model

Now that we have estimates of the marginal likelihood under each of our different models, we can evaluate their relative plausibility using Bayes factors. Use Table 10.1 to summarize the marginal log-likelihoods estimated using the stepping-stone and path-sampling methods.

Phylogenetics software programs log-transform the likelihood to avoid underflow, because multiplying likelihoods results in numbers that are too small to be held in computer memory. Thus, we must calculate the ln-Bayes factor (we will denote this value $\mathcal{K}$):

$$\mathcal{K} = \ln[BF(M_0, M_1)] = \ln[\mathbb{P}(\mathbf{X} \mid M_0)] - \ln[\mathbb{P}(\mathbf{X} \mid M_1)], \tag{10.3}$$

where $\ln[\mathbb{P}(\mathbf{X} \mid M_0)]$ is the *marginal lnL* estimate for model M_0. The value resulting from equation 10.3 can be converted to a raw Bayes factor by simply taking the exponent of $\mathcal{K}$

$$BF(M_0, M_1) = e^{\mathcal{K}}. \tag{10.4}$$

Alternatively, you can interpret the strength of evidence in favor of M_0 using the $\mathcal{K}$ and skip equation 10.4. In this case, we evaluate the $\mathcal{K}$ in favor of model M_0 against model M_1 so that:

$$\begin{array}{c} \text{if } \mathcal{K} > 1, \text{ then model } M_0 \text{ wins} \\ \text{if } \mathcal{K} < -1, \text{ then model } M_1 \text{ wins.} \end{array}$$

Thus, values of $\mathcal{K}$ around 0 indicate ambiguous support.

Using the values you entered in Table 10.1 and equation 10.3, calculate the ln-Bayes factors (using $\mathcal{K}$) for the different model comparisons. Enter your answers in Table 10.1 using the stepping-stone and the path-sampling estimates of the marginal log likelihoods.

Table 10.1: Marginal likelihoods and Bayes factors*.

Estimate	*Stepping-stone*	*Path sampling*
10.2.1 Marginal likelihood Yule (M_0)		
10.3 Marginal likelihood birth-death (M_1)		
Eq. 10.3: $BF(M_0, M_1)$		
Supported model?		

*you can edit this table

Do these data support a model without extinction ($\mu = 0$)?

10.5 Estimate Speciation and Extinction Rates

After comparing the marginal likelihoods using Bayes factors, you will discover which model is best supported by the data. With this model, we can now estimate posterior probability of the the global rate of speciation (and extinction if the birth-death model is used) given our observed tree.

$$\mathbb{P}(\lambda, \mu \mid \Psi) = \frac{\mathbb{P}(\Psi \mid \lambda, \mu)\mathbb{P}(d \mid \nu)\mathbb{P}(r \mid \alpha, \beta)}{\mathbb{P}(\Psi)} \tag{10.5}$$

- The `Rev` file for performing this analysis: `mcmc_BD_bears.Rev` or `mcmc_Yule_bears.Rev`.

10.5.1 Clear the workspace and load the preferred model

It is best to remove all of the previous model variables created in the previous section.

```
clear()
```

Now read in the observed tree from Figure 10.1.

```
T <- readTrees("data/bears_dosReis.tre")[1]
```

Initialize the useful variables:

```
n_taxa <- T.ntips()
names <- T.names()
mi = 1
```

Source the model file of your favorite model ($*$ = **Yule** or **BD**).

```
source("RevBayes_scripts/m_*_bears.Rev")
```

```
mymodel = model(birth_rate)
```

10.5.2 Set up parameter monitors

```
monitors[1] = mnFile(filename="output/BDR_mcmc_bears.log",printgen=10, diversification
    , birth_rate, turnover, death_rate)
monitors[2] = mnScreen(printgen=1000, birth_rate)
```

Note that if your preferred model was the Yule process, then you would only have the **birth_rate** parameter listed in the **mnFile** monitor.

10.5.3 Run MCMC

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

```
mymcmc.burnin(generations=10000,tuningInterval=1000)
mymcmc.run(generations=50000)
```

```
mymcmc.operatorSummary()
```

- Visualize the MCMC samples of the birth rate and death rate parameters in Tracer.

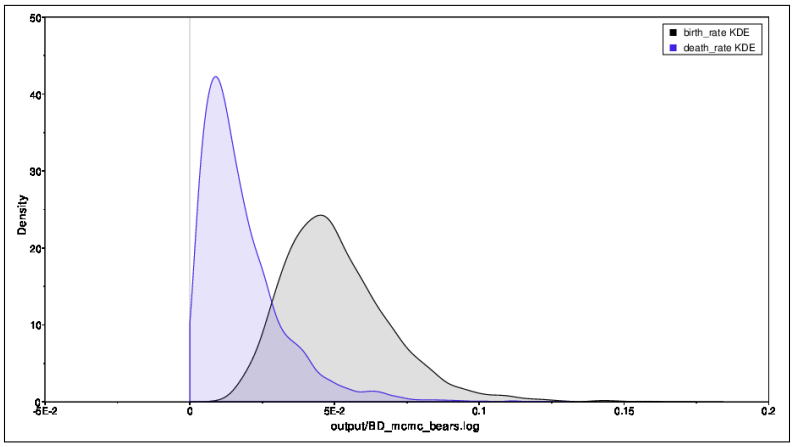


Figure 10.8: The marginal densities of **birth_rate** and **death_rate** estimated under the birth-death model in RevBayes.

Bibliography

Aldous, D. 2001. Stochastic models and descriptive statistics for phylogenetic trees, from Yule to today. Statistical Science 16:23–34.

dos Reis, M., J. Inoue, M. Hasegawa, R. Asher, P. Donoghue, and Z. Yang. 2012. Phylogenomic datasets provide both precision and accuracy in estimating the timescale of placental mammal phylogeny. Proceedings of the Royal Society B: Biological Sciences 279:3491–3500.

Hartmann, K., D. Wong, and T. Stadler. 2010. Sampling trees from evolutionary models. Systematic Biology 59:465–476.

Kendall, D. G. 1948a. On the generalized "birth-and-death" process. The Annals of Mathematical Statistics 19:1–15.

Kendall, D. G. 1948b. On the generalized “birth-and-death” process. Annals of Mathematical Statistics 19:1–15.

Nee, S., R. M. May, and P. H. Harvey. 1994. The reconstructed evolutionary process. Philosophical Transactions of the Royal Society B 344:305–311.

Rannala, B. and Z. Yang. 1996. Probability distribution of molecular evolutionary trees: A new method of phylogenetic inference. Journal of Molecular Evolution 43:304–311.

Thompson, E. A. 1975. Human Evolutionary Trees. Cambridge University Press, Cambridge, England.

Yang, Z. and B. Rannala. 1997. Bayesian phylogenetic inference using DNA sequences: A Markov chain Monte Carlo method. Molecular Biology and Evolution 14:717–724.

Yule, G. U. 1924. A mathematical theory of evolution, based on the conclusions of Dr. J. C. Wills, F. R. S. Philosophical Transactions of the Royal Society of London, Biology 213:21–87.

Part VI

Gene tree - Species tree estimation

Chapter 11

Multispecies Coalescent

11.1 Overview: Gene tree-species tree models

Ever since Zuckerkandl and Pauling (1965), researchers have acknowledged that phylogenies reconstructed from homologous gene sequences could differ from species phylogenies. As molecular sequences accumulated, the link between gene trees and species trees started to be modeled. The first models were based on parsimony, and aimed for instance at reconciling a gene tree with a species tree by minimizing the number of events of gene duplication and gene loss. In the past dozen years, probabilistic models have been proposed to reconstruct gene trees and species trees in a rigorous statistical framework. Models and algorithms have quickly grown in complexity, to model biological processes with increasing realism, to accommodate several processes at the same time, or to handle genome-scale data sets. In this overview we will not detail these models, and we invite the interested reader to take a look at recent reviews (*e.g.,* (SzÃűllÅŚsi et al. 2014)).

11.1.1 Processes of discord

There are several reasons why a gene tree may differ from a species tree. Of course, a gene tree may differ from the species tree just because a mistake was made during the analysis of the gene sequences, at any point in a pipeline going from the sequencing itself to the tree reconstruction. Such a mistake would produce an incorrect gene tree. Here we do not mean this kind of discord, but rather discord that comes from a real biological process that generates true gene histories that differ from true species histories. These processes include gene duplication, gene loss, gene transfer (used loosely here to also include reticulation, hybridization between species), and incomplete lineage sorting (Fig. 11.1). Incomplete lineage sorting will be discussed in more details in the following subsection.

Fig. 11.1 suggests that for all processes the gene tree can be seen as the product of a branching process operating inside the species tree. Indeed, all processes are modeled as some type of birth-death process running along the species tree. For duplication/loss models, birth correspond to gene duplication events, and death to gene loss events. Transfers can be added to the model by introducing another type of birth, with a child lineage appearing in another branch of the species tree. Incomplete lineage sorting is also modeled with a birth-death type of model, the coalescent. All these models can be made heterogeneous, for instance by allowing different sets of parameters for different branches of the species tree. This is useful to model differences in rates of duplication, loss or transfer among species, or to model different effective population sizes in a species tree. In `RevBayes` so far only models of incomplete lineage sorting have been implemented (models of duplication and loss and transfer will soon be added). Thanks to `RevBayes` modular design, there is quite a lot of flexibility in specifying the model, for instance by associating different parameters to different branches of the species tree, or by combining the gene tree-species tree model to other types of models, for instance models of trait evolution, or models of relaxed molecular clock.

11.1.2 Gene tree discordance is a problem for species tree reconstruction

There are several approaches to species tree reconstruction: concatenation and supertree approaches, which have been used for quite some time now, and more recently methods that rely on gene tree-species tree models.

1. Concatenation simply consists in taking all gene alignments, concatenating them into one super alignment, and then analyzing it as if it were a single gene sequence. Its main assumption is therefore that all sites of all genes have evolved according to the same species tree. This assumption is not correct because all the processes of discord presented above conspire to make gene trees different

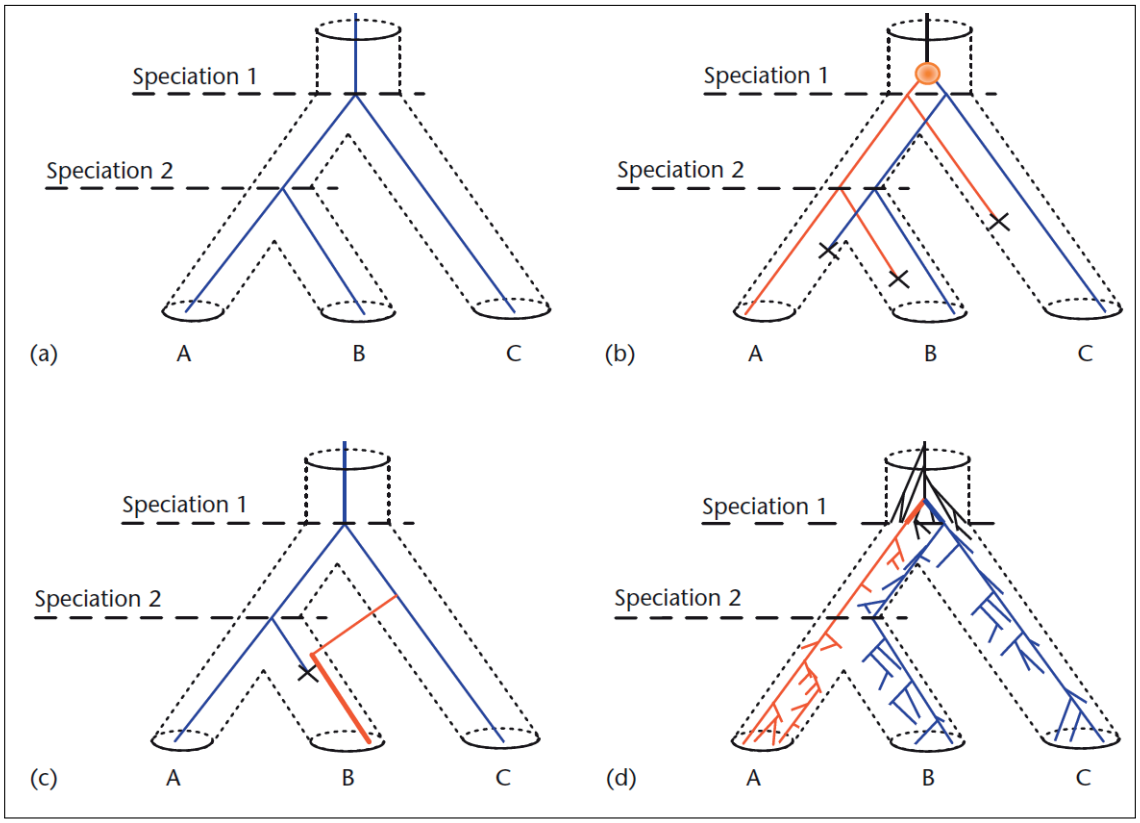


Figure 11.1: The processes of discord. The species tree is represented as a tubular structure. Gene trees are blue and red lines running along the species trees. a) A gene tree that perfectly matches the species tree. b) The gene tree and the species tree differ because of gene duplications and losses. c) The gene tree and the species tree differ because of gene transfer and gene loss. d) The gene tree and the species tree differ because of incomplete lineage sorting. [Replicated from Fig. 2 in Boussau (2009).]

from the species tree. In practice, this matters: simulation studies have found that in the presence of incomplete lineage sorting, in some particular areas of the parameter space, concatenation will often return an incorrect species tree (Leaché and Rannala 2011). Another example might be found in prokaryotic phylogenetics, where the quest for a tree of life has been difficult, to the point that many doubt that one could find a meaningful species tree representing vertical descent. Recent models of gene tree evolution incorporating lateral gene transfer allow tackling this question in a more principled way.

2. Supertree approaches differ from concatenation notably by discarding sequence information once individual gene trees have been built. Contrary to concatenation approaches that combine individual gene alignments, supertree approaches combine individual gene trees to obtain a species tree. Most supertree methods are not based on an explicit model of the processes causing discordance between gene trees and species tree (although there are exceptions, notably modelling incomplete lineage sorting, see below). Instead, they aim at finding a tree that would best describe the distribution of gene trees, according to some fairly arbitrary criterion. In practice, these methods have been found to provide reasonable results in many cases, but in simulations they are usually less accurate than concatenation.

3. Methods that rely on gene tree-species tree models appear very promising as they explicitly model the processes of discord, and can be combined with *e.g.,* models of sequence evolution, models of

co-evolution between gene trees, models of trait evolution...

11.1.3 Modelling incomplete lineage sorting: the multispecies coalescent

Incomplete lineage sorting is a population-level process. In a species, at a given time, there are several alleles for a given locus in the genome. These alleles have their own history, they diverged from each other at various times in the past. This history can differ from the species history, because several alleles can persist through a speciation event, and because, without selective effects, the sorting of alleles during a speciation event is random and can result in a tree that differs from the species tree (Fig. 11.1d). In all cases, incongruence between the gene tree and the species tree occurs when alleles persist over the course of several speciation events. When reconstructing a gene tree, one therefore gets the history of the alleles that have been sampled (at best), not the history of the species.

In 2003, Rannala and Yang proposed a powerful way to model the sorting of alleles along a phylogeny of several species (Rannala and Yang 2003), the multispecies coalescent (Fig. 11.2). This model is at the origin of most model-based approaches to reconstruct gene and species trees (Edwards, Liu and Pearl 2007; Heled and Drummond 2010). The multispecies coalescent appropriately models the evolution of a population of alleles along a species tree. Along the species tree, it allows different branch lengths, in units of time, and also allows different effective population sizes. Computing the probability of a gene tree given a species tree and other parameters is quite easy. Bascially it works by cutting the gene tree into independent species-specific subtrees, computing probabilities for each of those subtrees, and combining them all at the end to get the probability of the gene tree according to the multispecies coalescent, given the current parameter values. Cutting the gene tree into species-specific subtrees is quite easy, because we can use the dates of speciation events to identify parts of the gene trees that are before and after speciation events. The resulting subtrees are represented with the grey boxes in Fig. 11.2. In this figure, each subtree corresponds to one particular population, either extant or ancestral. Inside each subtree, given its length, the effective population size, and dates of coalescence (divergences of alleles), the coalescent model provides simple formulas for computing the probability of the gene subtree given other parameters. Because we consider that these subtree probabilities are all independent of one another, they are then multiplied to get the gene tree probability given current parameter values.

Two parameters associated to branches of the species tree have a direct impact on the expected amount of gene tree-species tree incongruence:

- **Time between speciations.** The more a branch length increases, the more the pool of alleles is expected to change. Alleles are therefore less likely to persist for several speciation events if the branches between these speciation events are long.
- **Effective population size between speciations.** In populations with small effective population sizes, chance events can cause large shifts in allele frequencies, and possibly disappearance of alleles. In large populations, because an allele is likely carried by a large number of individuals, its disappearance is less likely, the population of alleles is more stable. Alleles are therefore less likely to persist for several speciation events if the branches between these speciation events are characterised by small effective population sizes.

Overall, larger amounts of gene tree-species tree incongruence are expected in phylogenies characterised by short branches with large population sizes. A corollary of that is that larger amounts of gene tree-gene tree incongruence are expected as well. To measure the susceptibility of species phylogenies to generate

incomplete lineage sorting, the concept of *coalescent time units* has been introduced. Coalescent time units are obtained when branch length λ, in number of generations, is divided by effective population size N_e. As a consequence, in a species tree whose branches are expressed in coalescent time units, a branch length of 1 *coalescent time unit* means a branch length of N_e *generations*. Once branch lengths on the species tree are measured in coalescent time units, it becomes easy to spot species trees that generate a lot of incongruence: those are short trees.

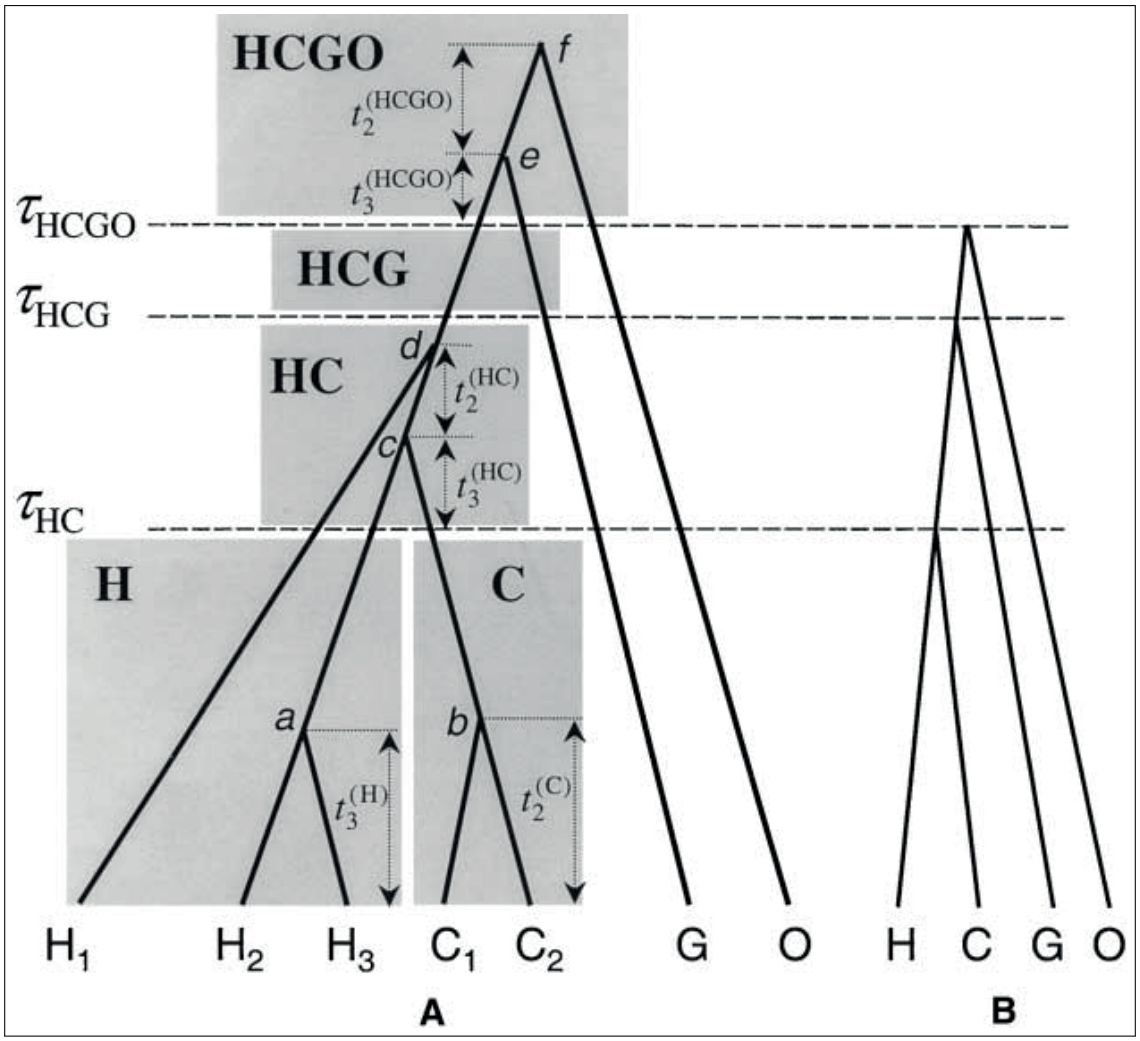


Figure 11.2: The multispecies coalescent. A) A gene tree, including 3 human alleles, 2 Chimp alleles, one Gorilla allele, and one Orang-outan allele. τ parameters are speciation times, t parameters are divergence time in the gene tree, the grey squares represent the ancestral populations, with their respeciive sizes. B) The corresponding species tree. In this model, the speciation times define minimal boundaries for allele divergence times. [Replicated from Fig. 1 in Rannala and Yang (2003).]

The exercises assume you have a working installation of RevBayes. In this introductory tutorial, we will apply the multispecies coalescent model to 10 gene alignments from 23 mammalian species. We will specify the multispecies coalescent, with different effective population sizes for each branch of the species tree. We will assume that:

- The species tree is drawn from a constant birth-death process.
- Along the branches of the species tree, a multispecies coalescent process generates gene trees. Different effective population sizes are assigned to each branch of the species tree.
- Along each gene tree, gene sequences are evolved according to an HKY model with a strict clock.
- Here, we run an MCMC on this model, using data from 10 genes in 23 mammalian species.

Scripts are all placed in
tutorials/RB_GeneTreeSpeciesTreeSimple_Tutorial/RB_GeneTreeSpeciesTreeSimple_tutorial_files/RevBayes_scripts/.

First RevBayes exercise: loading gene alignments into RevBayes

1. Open RevBayes
2. Let's load all 10 gene alignments.

```
locus_names = ["COIII", "FGA", "GHRmeredith", "lrpprc_169", "npas3", "sim1", "tex2
    ", "ttr", "zfy", "zic3"]

num_loci = locus_names.size()

# read in each data matrix separately
for ( i in 1:num_loci ) {
    data[i] <- readDiscreteCharacterData("data/" + locus_names[i] + ".fasta")
}

# Now we get some useful variables from the data. We need these later on.
primate_tree = readTrees("data/primates.tree")[1]
n_species <- primate_tree.ntips()
names <- primate_tree.names()
n_branches <- 2 * n_species - 1 # number of branches in a rooted tree

# We set our move index
mi = 0
```

3. We simulate a species tree topology according to a birth-death process with fixed parameter values:

```
# Specify a prior on the diversification and turnover rate
diversification ~ dnGamma(2,2)
relativeExtinction ~ dnBeta(1,1)

# Now transform the diversification and turnover rates into speciation and
    extinction rates
speciation := abs(diversification / (1.0 - relativeExtinction) )
extinction := speciation * relativeExtinction

# Specify a prior on the root age (our informed guess is about 75-80 mya)
root ~ dnUniform(0,1000)

# Now comes the most important trick!!!
#
# We need to set a low root age so that in the beginning of the MCMC all lineages
# evolve in the same "root" population.
# This enables that we can quickly get good estimates of the gene-trees and then
# pull up the species tree.
root.setValue(0.001)

sampling_fraction <- 23 / 450 # we sampled 23 out of the ~ 450 primate species

# create some moves that change the stochastic variables
# Moves are sliding and scaling proposals
moves[++mi] = mvSlide(diversification,delta=1,tune=true,weight=2)
moves[++mi] = mvSlide(relativeExtinction,delta=1,tune=true,weight=2)
moves[++mi] = mvScale(diversification,lambda=1,tune=true,weight=2)
moves[++mi] = mvScale(relativeExtinction,lambda=1,tune=true,weight=2)
moves[++mi] = mvSlide(root,delta=1,tune=true,weight=0.2)
moves[++mi] = mvScale(root,lambda=1,tune=true,weight=0.2)


# construct a variable for the tree drawn from a birth-death process
psi ~ dnBDP(lambda=speciation, mu=extinction, rootAge=abs(root), rho=
    sampling_fraction, nTaxa=n_species, names=names )

moves[++mi] = mvNarrow(psi, weight=5.0)
moves[++mi] = mvNNI(psi, weight=1.0)
moves[++mi] = mvFNPR(psi, weight=3.0)
moves[++mi] = mvGPR(psi, weight=3.0)
moves[++mi] = mvSubtreeScale(psi, weight=3.0)
moves[++mi] = mvNodeTimeSlideUniform(psi, weight=15.0)
```

4. Now that we have a species tree topology, we can have a multispecies coalescent process run along the species tree to generate gene trees.

```
# We assume independent effective population size parameters for each branch of
    the species tree.
for (i in 1:n_branches) {
  Ne[i] ~ dnExponential(0.1)
  moves[++mi] = mvScale(Ne[i],1,true,1.0)
}

# We could instead assume a single effective population size for the entire
    species tree with the following two lines:
#Ne ~ dnGamma(shape=1.0,rate=1.0)
#moves[++mi] = mvScale(Ne,1,true,1.0)

for (i in 1:num_loci) {

   # We need to read in files providing the link between gene names and species
       names
   taxa = readTaxonData("data/species_maps/primates_" + locus_names[i] + "
       _species_map.txt")

   # The gene tree from the multispecies coalescent process
   # Note that Ne is a vector of effective population sizes,
   # allowing 1 parameter per branch of the species tree.
   geneTree[i] ~ dnCoalMultiSpeciesConst(speciesTree=psi, Ne=Ne, taxa=taxa)

   # moves on the tree
   moves[++mi] = mvNNI(geneTree[i], 5.0)
   moves[++mi] = mvNarrow(geneTree[i], 5.0)
   moves[++mi] = mvFNPR(geneTree[i], 3.0)
   moves[++mi] = mvGPR(geneTree[i], 2.0)
   moves[++mi] = mvSubtreeScale(geneTree[i], 5.0)
   moves[++mi] = mvNodeTimeSlideUniform(geneTree[i], 20.0)
   moves[++mi] = mvRootTimeSlide(geneTree[i], 1.0, true, 3.0)

}
```

5. Now we have gene trees, complete with branch lengths. The aim is to simulate gene sequences along these gene trees. To do that, we need to decide how we transform time in number of generations, which are the units of the gene tree branch lengths, into time in effective number of substitutions per site. We choose to assume a strict clock, *i.e.,* a single coefficient that applies to all branches of the gene tree. However, we allow that different genes will have different values of this coefficient.

```
# We use a fixed estimate of the clock rate which is 1.00 substitution per site
    per million years
# Arbitrarily, we decide that the first gene has a clock rate of 1
clockRate[1] <- 1.0
```

```
# All other genes have clock rates defined relatively to the first gene
for ( i in 2:num_loci ) {
    relativeClockRate[i] ~ dnLognormal(0,1)
    clockRate[i] := clockRate[1] * relativeClockRate[i]
    moves[++mi] = mvScale(relativeClockRate[i],lambda=1,tune=true,weight=1)
}
```

6. Now the gene tree branch lengths are in units that are appropriate for simulating gene sequences. We just need to define the substitution matrix to decide how sequences will evolve. We use a single HKY matrix that will apply to all sites, and consider that within a gene all sites evolve at the same rate of evolution.

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

    #### specify the HKY substitution model applied uniformly to all sites of a
       gene
    kappa[i] ~ dnLognormal(0,1)
    moves[++mi] = mvScale(kappa[i],weight=1)

    pi_prior[i] <- v(1,1,1,1)
    pi[i] ~ dnDirichlet(pi_prior[i])
    moves[++mi] = mvSimplexElementScale(pi[i],weight=2)


    #### create a deterministic variable for the rate matrix
    Q[i] := fnHKY(kappa[i],pi[i])

}
```

7. Now we have defined all the bricks of the model, and we need to link the model to the data.

```
for ( i in 1:num_loci ) {
    # the sequence evolution model
    seq[i] ~ dnPhyloCTMC(tree=geneTree[i], Q=Q[i], branchRates=clockRate[i], type
       ="DNA")

    # attach the data
    seq[i].clamp(data[i])
}
```

8. Finally, we need to perform inference under the model, using the data.

```
# We get a handle on our model.
# We can use any node of our model as a handle, here we choose to use the topology
    .
mymodel = model(psi)

# Monitors to check the progression of the program
monitors[1] = mnScreen(printgen=100, root)
monitors[2] = mnModel(filename="output/primates_fixed_clock.log",printgen=10,
    separator = TAB)
monitors[3] = mnFile(filename="output/primates_fixed_clock.trees",printgen=10,
    separator = TAB, psi)
for ( i in 1:num_loci ) {
    # We add a monitor for each gene tree
    monitors[i+3] = mnFile(filename="output/primates_fixed_clock_" + locus_names[i
        ] + ".trees",printgen=10, separator = TAB, geneTree[i])
}

# Here we use a plain MCMC. You could also set nruns=2 for a replicated analysis
# or use mcmcmc with heated chains.
mymcmc = mcmc(mymodel, monitors, moves)

# This should be sufficient to obtain enough MCMC samples
mymcmc.burnin(generations=5000,tuningInterval=250)
mymcmc.run(generations=50000)
```

9. Now we can perform some post-run analyses.

```
# Now, we will analyze the tree output.
# Let us start by reading in the tree trace
treetrace = readTreeTrace("output/primates_fixed_clock.trees", treetype="clock")
# and get the summary of the tree trace
treetrace.summarize()

mapTree(treetrace,"output/primates_fixed_clock.tree")
```

11.2 Batch Mode

If you wish to run these exercises in batch mode, the files are provided for you.

You can carry out these batch commands by providing the file name when you execute the **rb** binary in your unix terminal (this will overwrite all of your existing run files).

- **$ rb NameOfTheFile.Rev**

11.3 Things to think about

How did the different methods perform? Did you expect to see these differences? It has been shown that the concatenation approach could be inconsistent under some conditions of population size and of divergence times (Degnan and Rosenberg 2006). Do you find that concatenation performs worse that its competitors? Which models seem to "mix" better? In particular, does the full multispecies coalescent mix well? Why can we expect this model in particular would have difficulties mixing?

11.4 Useful Links

- RevBayes: https://github.com/revbayes/code
- Tree Thinkers: http://treethinkers.org
- NJplot, a lightweight tree plotting program: http://doua.prabi.fr/software/njplot
- Seaview, a program to handle alignments and trees: http://doua.prabi.fr/software/seaview

Questions about this tutorial can be directed to:

- Bastien Boussau (email: boussau@gmail.com)
- Sebastian Höhna (email: sebastian.hoehna@gmail.com)

Bibliography

Boussau B. 2009. Phylogenetic Relationships Deduced from Whole Genome Comparisons. .

Degnan JH, Rosenberg NA. 2006. Discordance of species trees with their most likely gene trees. PLoS Genetics. 2:e68.

Edwards SV, Liu L, Pearl DK. 2007. High-resolution species trees without concatenation. Proceedings of the National Academy of Sciences of the United States of America. 104:5936–5941.

Heled J, Drummond AJ. 2010. Bayesian Inference of Species Trees from Multilocus Data. Molecular Biology and Evolution. 27:570.

Leaché AD, Rannala B. 2011. The accuracy of species tree estimation under simulation: a comparison of methods. Systematic Biology. 60:126–137.

Rannala B, Yang Z. 2003. Bayes estimation of species divergence times and ancestral population sizes using DNA sequences from multiple loci. Genetics. 164:1645–1656.

SzÃűllÅŚsi GJ, Tannier E, Daubin V, Boussau B. 2014. The inference of gene trees with species trees. Systematic Biology. .

Zuckerkandl E, Pauling L. 1965. Evolutionary divergence and convergence in proteins BT - Evolving genes and proteins. In: Bryson V, Vogel HJ, editors, Evolving genes and proteins, New York: Academic Press, pp. 97–166.

Part VII

Biogeography

Chapter 12

Dispersal, Extirpation and Cladogenesis (DEC)

12.1 Introduction

How did species come to live where they're found today? To answer this, we can leverage phylogenetic and geological information to model species distributions as the outcome of biogeographic processes. These natural processes require some additional considerations, such as how ranges are inherited following speciation events, how geological events might influence dispersal rates, and what factors affect rates of dispersal and extirpation. The major challenge of modeling range evolution is how to translate these natural processes into stochastic processes that remain tractable for inference. This tutorial provides a brief background in some of these models, then describes how to perform Bayesian inference of historical biogeography using `RevBayes`.

12.2 Dispersal-Extinction-Cladogenesis model

12.2.1 Range characters

Discrete biogeographical models typically rely on presence-absence data, where a species is observed or not observed across multiple discrete areas. For example, say there are three areas: A, B, and C. Say a species is present in areas A and C, then its range equals AC, which can also be encoded into the length-3 bit vector, 101. Bit vectors may also be transformed into (decimal) integers, *e.g.,* the binary number 101 equals the decimal number 5.

$$(\emptyset, A, B, AB, C, AC, BC, ABC) \Leftrightarrow (000, 100, 010, 110, 101, 011, 111) \Leftrightarrow (0, 1, 2, 3, 4, 5, 6, 7)$$

Decimal representation is rarely used in discussion, but it is useful to keep in mind when considering the total number of possible ranges for a species.

12.2.2 Modeling anagenic range evolution

How might we model the dynamics of species range evolution? In this section, we'll cover the Dispersal-Extinction-Cladogenesis model proposed by Ree et al. (2005). To begin, we'll focus on anagenesis: evolution that occurs between speciation events within lineages. Since we have discrete characters we'll use the continuous-time Markov chain, which allows us to compute transition probability of a character changing from i to j in time t through matrix exponentiation

$$\mathbf{P}_{i,j}(t) = \left[\exp\left\{\mathbf{Q}t\right\}\right]_{i,j},$$

where $\mathbf{Q}$ is the instantaneous rate matrix defining the rates of change between all pairs of characters, and $\mathbf{P}$ is the transition probability rate matrix. Remember, i and j represent different ranges, each of which is encoded as the set of areas occupied by the species. Exponentiation of the rate matrix is powerful because it integrates over all possible scenarios of character transitions that could occur during t so long as the chain begins in state i and ends in state j.

We can then encode $\mathbf{Q}$ to reflect the allowable classes of range evolution events with biologically meaningful parameters. We'll take a simple model of range expansion (e.g. $BC \rightarrow ABC$) and range contraction (e.g. $BC \rightarrow C$). (Range expansion may also be referred to as dispersal or area gain and range contraction as extirpation, (local) extinction, or area loss.) The rates in the transition matrix for three areas might appear as

$$\mathbf{Q} = \begin{array}{c|cccccccc} & \emptyset & A & B & AB & C & AC & BC & ABC \\ \hline \emptyset & - & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ A & e_A & - & 0 & d_{AB} & 0 & d_{AC} & 0 & 0 \\ B & e_B & 0 & - & d_{BA} & 0 & 0 & d_{BC} & 0 \\ AB & 0 & e_A & e_B & - & 0 & 0 & 0 & d_{AC} + d_{BC} \\ C & e_C & 0 & 0 & 0 & - & d_{CA} & d_{CB} & 0 \\ AC & 0 & e_C & 0 & 0 & e_A & - & 0 & d_{AB} + d_{CB} \\ BC & 0 & 0 & e_C & 0 & e_B & 0 & - & d_{BA} + d_{CA} \\ ABC & 0 & 0 & 0 & e_C & 0 & e_B & e_A & - \end{array},$$

where $e = (e_A, e_B, e_C)$ are the (local) extinction rates per area, and $d = (d_{AB}, d_{AC}, d_{BC}, d_{CB}, d_{CA}, d_{BA})$ are the dispersal rates between areas. Notice that the sum of rates leaving state $\emptyset$ is zero, meaning any species that loses all areas in its range remains permanently extinct.

? For the three-area DEC rate matrix above, what is the rate of leaving state AC in terms of dispersal and extinction parameters?

Note the rate of more than one event occurring simultaneously is zero, so a range must expand twice by one area in order to expand by two areas.

? What series of transition events might explain a lineage evolving from range ABC to range A? From range AB to range C?

Of course, this model can be specified for more than three areas.

? Imagine a DEC rate matrix with four areas, $ABCD$. What would be the dispersal rate for $Q_{BC,BCD}$? How many states does a DEC rate matrix with four areas have? What is the relationship between the number of areas and the number of states under the DEC model?

Let's consider what happens to the size of $\mathbf{Q}$ when the number of areas, N, becomes large. For three areas, $\mathbf{Q}$ is size 8×8. For ten areas, $\mathbf{Q}$ is size $2^{10} \times 2^{10} = 1024 \times 1024$, which approaches the largest size matrices that can be exponentiated in a practical amount of time. For twenty areas, $\mathbf{Q}$ is size $2^{20} \times 2^{20} \approx 10^6 \times 10^6$ and exponentiation is not viable.

12.2.3 Modeling cladogenic range evolution

Cladogenesis describes evolutionary change accompanying speciation. Daughter species are not expected to inherit their ancestral range identically in general. For each internal node in the reconstructed tree, one of two cladogenic events can occur: sympatry or allopatry. Say the range of a species is A the moment before speciation occurs at an internal phylogenetic node. Since the species range is size one, both daughter lineages necessarily inherit the ancestral species range (A). In DEC parlance, this is called a *narrow sympatry* event.

Now suppose the ancestral range is ABC. Under *subset sympatric cladogenesis*, one lineage identically inherits the ancestral species range, ABC, while the other lineage inherits only a single area, i.e. only A or B or C. For *widspread sympatric cladogenesis*, both lineages inherit the ancestral range, ABC. Under *allopatric cladogenesis*, the ancestral range is split evenly among daughter lineages, e.g. one lineage may inherit AB and the other inherits C.

For an excellent overview of described state transitions for cladogenic events, see Matzke (2013).

? Given the state is AB before cladogenesis, and allowing subset sympatry, widespread sympatry, and allopatry, what are the 7 possible states in the daughter lineages after cladogenesis?

The probabilities of anagenic change along lineages must account for all combinations of starting states and ending states. For 3 areas, there are 8 states, and thus $8 \times 8 = 64$ probability terms for pairs of states. For cladogenic change, we need transition probabilities for all combinations of states before cladogenesis, after cladogenesis for the left lineage, and after cladogenesis for the right lineage. Like above, for three areas, there are 8 states, and $8 \times 8 \times 8 = 512$ cladogenic probability terms.

? For three areas, there are three narrow, four widespread, 18 subset sympatric events and 12 allopatric cladogenesis events. What proportion of terms in the cladogenesis matrix are zero?

The DEC model ignores speciation events hidden by extinction or incomplete taxon sampling. The probability of cladogenesis and local extinction events would ideally be linked to a birth-death process, as it is in the GeoSSE model (Goldberg et al. 2011). Unfortunately, since the numerical method for SSE models scale poorly, and DEC models remain the only option when the geography has more than two or three areas. For more than ten areas, data augmentation may be used to infer ancestral ranges, as described in Section 13.1.

The rest of this section will describe how to run a simple DEC analysis using RevBayes.

Specifying a simple DEC model

We'll use the primate dataset with 23 taxa. To keep the model simple, we'll discretize their ranges into just three areas: the New World (A), Africa (B), and Eurasia (C). For simplicity, we'll assume their phylogeny is time-calibrated, errorless, and fixed.

Create some `String` variables for file handling,

```
data_fn = "data/primates_bg_n3.tsv"
tree_fn = "data/primates.tree"
out_fn = "output/bg_same"
```

then read in our character data, using `type="Bitset"` to indicate ranges are encoded with bits, e.g. 011

```
data = readCharacterDataDelimited(file=data_fn, type="Bitset")
```

and our tree

```
psi <- readTrees(tree_fn)[1]
```

Next, compute the number of states from the number of areas

```
n_areas = 3
n_states = floor(2^n_areas)
```

Declare index variables for our move vectors for future use

```
mi = 0
```

Now, we'll begin to construct the rate matrix for anagenic events. First create a matrix, 8-by-8 in size, initialized with all zeroes

```
for (i in 1:n_states) {
    for (j in 1:n_states) {
        r[i][j] <- 0.0
    }
}
```

Now we need to populate the non-zero rate matrix elements, which are in terms of dispersal and extinction rates. We'll use one dispersal rate and one extinction rate for this tutorial, and explore more complex models in later sections. For later reference, this will be called the "same rate" model.

First, create a extinction rate parameter and assign it a scale move

```
r_e ~ dnExponential(10.0)
mv[++mi] = mvScale(r_e, weight=5)
```

Before assigning the rates to the rate matrix, we'll create a vector to hold the per-area extinction rates

```
for (i in 1:n_areas) {
    e[i] := r_e
}
```

Now create the dispersal rate and scale move

```
r_d ~ dnExponential(10.0)
mv[++mi] = mvScale(r_d, weight=5)
```

then assign the between-area dispersal rates as determined by r_d

```
for (i in 1:n_areas) {
    for (j in 1:n_areas) {
        d[i][j] <- 0.
        if (i != j) {
            d[i][j] := r_d
        }
    }
}
```

Although the DEC rate matrix can be easily constructed by typing

```
q := fnDECRateMatrix(d,e)
```

manually encoding the structure of the DEC rate matrix illuminates the relationship between dispersal rates, extinction rates, area gain rates, and area loss rates. To start, we'll populate the non-zero rate matrix elements. Rates are indexed by the natural number value of the range (plus one), e.g. the range spanning Eurasia and Africa is coded as 011, which is state 4.

First assign the extinction (range loss) rates

```
r[2][1] := e[1]            # 100 -> 000 : Extirpate in area 1
r[3][1] := e[2]            # 010 -> 000 : Extirpate in area 2
r[4][2] := e[2]            # 011 -> 001 : Extirpate in area 2
r[4][3] := e[3]            # 011 -> 010 : Extirpate in area 3
r[5][1] := e[3]            # 001 -> 000 : Extirpate in area 3
r[6][2] := e[1]            # 101 -> 001 : Extirpate in area 1
r[6][5] := e[3]            # 101 -> 100 : Extirpate in area 3
r[7][3] := e[1]            # 110 -> 010 : Extirpate in area 1
r[7][5] := e[2]            # 110 -> 100 : Extirpate in area 2
r[8][4] := e[1]            # 111 -> 011 : Extirpate in area 1
r[8][6] := e[2]            # 111 -> 101 : Extirpate in area 2
r[8][7] := e[3]            # 111 -> 110 : Extirpate in area 3
```

then the dispersal (range gain) rates

```
r[2][4] := d[3][2]           # 001 -> 011 : Disperse from area 3 to 2
r[2][6] := d[3][1]           # 001 -> 101 : Disperse from area 3 to 1
r[3][4] := d[2][3]           # 010 -> 011 : Disperse from area 2 to 3
r[3][7] := d[2][1]           # 010 -> 110 : Disperse from area 2 to 1
r[5][6] := d[1][3]           # 100 -> 101 : Disperse from area 1 to 3
r[5][7] := d[1][2]           # 100 -> 110 : Disperse from area 1 to 2
r[4][8] := d[2][1] + d[3][1] # 011 -> 111 : Disperse from area 2 to 1 and from 3 to 1
r[6][8] := d[1][2] + d[3][2] # 101 -> 111 : Disperse from area 1 to 2 and from 3 to 2
r[7][8] := d[1][3] + d[2][3] # 110 -> 111 : Disperse from area 1 to 3 and from 2 to 3
```

Show the value of `r` and compare it to the matrix in Section 2.2.

Of course we did not need to declare `d` and `e` to assign `r`, but we'll see these intermediate variables act as a template expose the structure of `r` for modification.

So far, we only have the desired parameterization of the rate matrix, but we still haven't created a rate matrix function. Converting the vector-of-vectors, `r`, into a simplex allows us to use existing rate matrix functions.

First, we'll convert `r` into a one-dimensional vector, skipping the diagonal elements.

```
k = 1
for (i in 1:n_states) {
    for (j in 1:n_states) {
        if (i != j) {
            er_nat[k++] := r[i][j]
        }
    }
}
```

Finally, normalize `er_nat` using a simplex, then pass the resulting exchangeability rates as arguments into the rate matrix function, `q`.

```
er := simplex(er_nat)
bf <- simplex(rep(1,n_states))
q := fnFreeK(er, bf)
```

This yields the desired three-area DEC rate matrix modeling anagenic character change.

In contrast, cladogenic event probabilites are given by a transition probability matrix and do not require a rate matrix. First, we will create a vector of prior weights on cladogenesis events. Here, we assign a flat prior to all cladogenic events

```
widespread_sympatry_wt <- 1.0
subset_sympatry_wt   <- 1.0
allopatry_wt         <- 1.0
clado_prior          <- [ widespread_sympatry_wt, subset_sympatry_wt, allopatry_wt ]
```

then create the distribution over cladogenic event types and add its MCMC move

```
clado_type           ~ dnDirichlet(clado_prior)
mv[++mi]             = mvSimplexElementScale(clado_type, alpha=10, weight=5)
```

To give the simplex elements descriptive names when monitored, assign the values to deterministic nodes

```
widespread_sympatry  := clado_type[1]
subset_sympatry      := clado_type[2]
allopatry            := clado_type[3]
```

Then create the cladogenic transition probability matrix, which assigns probabilities to cladogenic event classes accoridng to `clado_type_prob`

```
clado_prob := fnCladoProbs(clado_type, n_areas, 2)
```

Add a parameter for a biogeographical clock, which scales the overall rate of range evolution. As a prior, an exponential distribution with rate 10 generates one dispersal or extinction event per 10 million years.

```
clock_bg ~ dnExponential(10)
mv[++mi] = mvScale(clock_bg, weight=5)
```

Finally, all our model components are encapsulated in the `dnPhyloCTMCClado` distribution, which is similar to `dnPhyloCTMC` except specialized to integrate over cladogenic events. Although this dataset has three areas, it is recognized single character with states valued from 1 to 2^3, hence `nSites=1`.

```
m ~ dnPhyloCTMCClado( tree=psi, Q=q, cladoProbs=clado_prob, branchRates=clock_bg, nSites
    =1, type="NaturalNumbers" )
```

The remaining tasks should be familiar by now, so we can proceed briskly. Attach the observed ranges to the model.

```
m.clamp(data)
```

Compose the model.

```
mdl = model(m)
```

Add the monitors. (The `mnJointConditionalAncestralState` monitor will be described in the next section.)

```
mn[1] = mnScreen(clock_bg, d[1][2], d[1][3], d[2][1], d[2][3], d[3][1], d[3][2], e[1], e
    [2], e[3], widespread_sympatry, subset_sympatry, allopatry, printgen=1000)
mn[2] = mnFile(clock_bg, d[1][2], d[1][3], d[2][1], d[2][3], d[3][1], d[3][2], e[1], e
    [2], e[3], widespread_sympatry, subset_sympatry, allopatry, file=out_fn+".params.txt
    ")
mn[3] = mnJointConditionalAncestralState(tree=psi, ctmc=m, filename=out_fn+".states.txt
    ", type="NaturalNumbers", printgen=10, withTips=true, withStartStates=true)
```

Create the MCMC object, and run the chain after burn-in.

```
ch = mcmc(mv,mn,mdl)
ch.burnin(1000, 10)
ch.run(10000)
```

Per-area rates

Biologically, local extinction events probably do not occur at equal rates across all areas, as done above. Ecological factors, geographical distances, etc. might cause these parameters to be weakly correlated or completely uncorrelated. Dispersal rates, also, might not be the same between pairs of areas, or even symmetric depending on the direction of dispersal. Rather than constraining all events of a type to share a common rate, instead you might give each area it's own extinction parameter

```
for (i in 1:3) {
    e[i] ~ dnExponential(10.0)
    mv[++mi] = mvScale(e[i], weight=5)
}
```

or give each ordered pair of areas it's own dispersal rate

```
for (i in 1:3) {
    for (j in 1:3) {
        d[i][j] <- 0.0
        if (i != j) {
            d[i][j] ~ dnExponential(10.0)
            mv[++mi] = mvScale(d[i][j], weight=5)
        }
    }
}
```

Note that you don't need the global dispersal rate **r_d** and extinction rate **r_e** anymore and you can remove the variable from your analysis. `RevBayes` might give you an error message if you have left them in.

12.2.4 Exercises

Exercises are independent of each other, except for Exercises 3a and 3b.

- 1) Widespread sympatric speciation is thought to be evolutionarily rare. Set the Dirichlet prior on cladogenic event types to heavily disfavor these events. Using Tracer, describe how changing the cladogenesis prior affects the extinction rate when compared with the "common rate" model.
- 2) Modifying the `RevBayes` script, parameterize the rate matrix so ranges may not grow beyond two areas in size. You may use `q_test := fnDECRateMatrix(e,d,2)` to confirm your results, which will give the same rate structure (and rates, up to a rescaling constant).
- 3a) Saving your commands to a file, create a script to produce the "per-area rate" model.
- 3b) Determine if the data support the "common rate" model over the "per-area rate" model. Use the stepping stone method to compute marginal likelihoods, which will let you compute Bayes factors for model selection.

12.2.5 (Advanced) Joint inference of phylogeny and historical biogeography

This is a slightly more advanced question that you may want to skip if you will always use known, fixed trees in your analyses. Using what you learned in this tutorial and the CTMC tutorial, perform a joint analysis of molecular and biogeographic evolution for primates.

Start by loading the sequence data matrix specified in **data/primates_cytb.nex**.

```
seqData <- readDiscreteCharacterData("data/primates_cytb.nex")
```

We need to get some useful variables from the data so that we will be able to specify the tree prior below. These variables are the number of tips, the number of nodes and the names of the species which we all can query from the continuous character data object.

```
numTips = seqData.ntaxa()
names = seqData.names()
numNodes = numTips * 2 - 1
```

Instead of having a fixed tree as in the previous, we should now define a *random* tree. We use a birth death prior with prior distributions on the **speciation** rate and **extinction** rate.

```
speciation ~ dnExponential(10.)
extinction ~ dnExponential(10.)
moves[++mi] = mvScale(speciation, lambda=1, tune=true, weight=3.0)
moves[++mi] = mvScale(extinction, lambda=1, tune=true, weight=3.0)
```

The phylogeny that we used are obviously not a complete sample of all the species and you should take the incomplete sampling into account. We will simply use an empirical estimate of the fraction of species which we included in this study. For more information about incomplete taxon sampling see **?** and **?**.

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

Now we are able to specify of tree variable **psi** which is drawn from a constant rate birth-death process. We will condition the age of the tree to be 75 million years old which is approximately the crown age of primates, although this estimate is still debated. We only condition here on the crown age for simplicity because we do not use any other fossil calibration.

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

Note that, here, we do not have included any fossil information: we are merely doing *relative* dating.

The first moves on the tree which we specify are moves that change the node ages. The first move randomly picks a subtree and rescales it, and the second move randomly pick a node and uniformly proposes a new node age between its parent age and oldest child's age.

```
moves[++mi] = mvSubtreeScale(psi, weight=5.0)
moves[++mi] = mvNodeTimeSlideUniform(psi, weight=10.0)
```

We also need moves on the tree topology to estimate the phylogeny. The two moves which you use are the nearest-neighbor interchange (NNI) and the fixed-nodeheight-prune-and-regraft (FNPR) (**?**).

```
moves[++mi] = mvNNI(psi, weight=5.0)
moves[++mi] = mvFNPR(psi, weight=5.0)
```

In the next step we set up the substitution model. First, we create a substitution model, just like what you probably did in previous tutorial (*e.g.,* RB_CTMC_Tutorial). In a first step, we will use a GTR+Gamma model. We can use a flat Dirichlet prior density on the exchangeability rates **er_mol** and the the base frequencies **pi_mol**.

```
er_mol_prior <- v(1,1,1,1,1,1)
er_mol ~ dnDirichlet(er_mol_prior)
pi_mol_prior <- v(1,1,1,1)
pi_mol ~ dnDirichlet(pi_mol_prior)
```

Now add the simplex scale move one each the exchangeability rates **er_mol** and the stationary frequencies **pi_mol** to the moves vector:

```
moves[++mi] = mvSimplexElementScale(er_mol,weight=15)
moves[++mi] = mvSimplexElementScale(pi_mol,weight=5)
```

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

```
Q_mol := fnGTR(er_mol,pi_mol)
```

The next part of the substitution process is the rate variation among sites. We will model this using the commonly applied 4 discrete gamma categories which only have a single parameter **alpha**. Let us specify the rate of **alpha** to 0.05 (thus the mean will be 20.0).

```
alpha_prior <- 0.05
```

Then create a stochastic node called **alpha** with an exponential prior:

```
alpha ~ dnExponential(alpha_prior)
```

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

```
gamma_rates := fnDiscretizeGamma( alpha, alpha, 4 )
```

The random variable that controls the rate variation is the stochastic node **alpha**. We will apply a simple scale move to this parameter.

```
moves[++mi] = mvScale(alpha, weight=2.0)
```

This finishes the substitution process part of the model.

Then next part of the model is the clock model. Here we need a clock model because we work on a time tree. We use an exponential distribution with expectation 0.1.

```
clock_mol ~ dnExponential(10)
moves[++mi] = mvScale(clock_mol, lambda=1, tune=true, weight=2.0)
```

Introduce a common prior for the molecular and biogeographical clocks.

```
clock_scale_bg ~ dnGamma(2.0,2.0)
clock_bg       := clock_mol * clock_scale_bg
```

Remember that you need to call the **PhyloCTMC** constructor to include the new site-rate parameter:

```
seq ~ dnPhyloCTMC(tree=psi, Q=Q_mol, siteRates=gamma_rates, branchRates=clockRate,
    type="DNA")
```

Finally we need to attach the molecular sequence data to our model.

```
seq.clamp(seqData)
```

We are essentially done now. We only need to add a new monitor for the tree so that we can monitor and build the maximum a posteriori tree later.

```
monitors[3] = mnFile(filename="output/biogeography_DEC_joint.trees", printgen=100, separator =
    TAB, psi)
```

Create the range evolution model as before, being sure to use the same `psi` and `clock_bg` from above.

The remaining challenge is to compose both submodels together using `model` and creating an MCMC object with `mcmc`. Good luck!

12.3 Epoch models and ancestral range reconstruction

12.3.1 Ancestral range reconstruction

From the previous section, we created an ancestral state monitor by

```
mn[3] = mnJointConditionalAncestralState(tree=psi, ctmc=m, filename=out_fn+".states.txt
    ", type="NaturalNumbers", printgen=10, withTips=true, withStartStates=true)
```

In this section, we'll use the output from running the DEC analysis assuming all each ordered pair of areas has its own dispersal parameter, and each area has its own extinction parameter. To generate these results, run

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

The joint-conditional ancestral state monitor samples the joint distribution of ancestral states every `printgen` iterations over the entire tree, conditioning on the observed tip states. Due to cladogenesis, the state before speciation and the states inherited after speciation may differ, which we monitor setting `withStartStates=true`. For convenience we record the tip states using `withTips=true` though these are known through the input data.

The resulting file, `bg_diff.states.txt`, appears as

```
Iteration     start_0 end_0 start_1 end_1 start_2 end_2 start_3 end_3 ...
0             2       2     2       4     2       2     2       6     ...
10            2       2     4       4     2       2     2       6     ...
20            2       2     2       4     2       2     6       6     ...
...
```

where columns give the integer-valued range corresponding to either the start or end of a branch leading to a particular node, and each row corresponds to a single sample drawn from the joint distribution of ancestral states. For example, the lineage leading to the node indexed 1 starts in state 2 (Africa) and ends in state 4 (Eurasia) at iteration 20. In Section 13.1, we will look at how stochastic mappings—the completely realized biogeographic history—may be analysed for data-augmented models. If you wish to write your own scripts to analyse the ancestral range reconstructions, know that each node's index is recorded when writing a tree's Newick string to file so states may be mapped on to the tree.

12.3.2 Data exploration with Phylowood

To interpret the biogeographical history of primates, we will generate a Phylowood (http://mlandis.github.io/phylowood) animation. First, we'll create variables for the relevant files

```
state_fn = "output/bg_diff.states.txt"
atlas_fn = "data/earth3.still.atlas.txt"
phw_fn = "output/bg_diff.phw.txt"
```

then create the animation file

```
convertToPhylowood(treefile=tree_fn, geofile=atlas_fn, statefile=state_fn, outfile=
    phw_fn, burnin=0., chartype="NaturalNumbers", bgtype="Range")
```

This file summarizes the MCMC output from a RevBayes biogeographical analysis as a Nexus-formatted file, which is used by Phylowood to generate interactive animations to explore biogeographic reconstructions.

→ Open `http://mlandis.github.io/phylowood`.

→ Drag and drop `./output/bg_diff.phw.txt` into the text field.

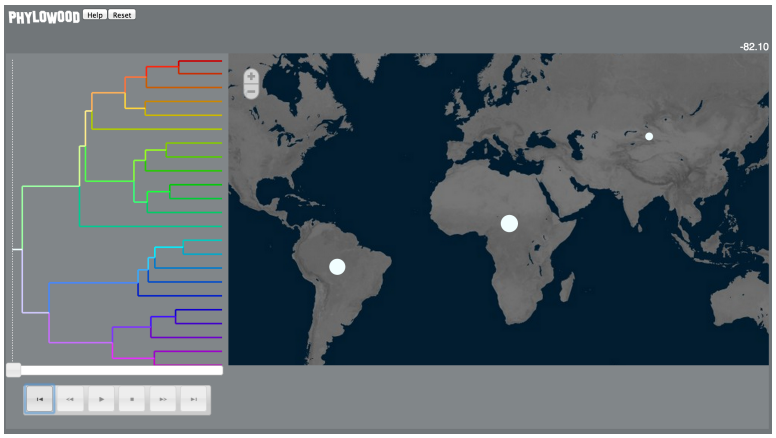


Figure 12.1: Phylowood frame showing posterior ancestral range of root node.

→ Click the Play button to view the animation.

There are three control panels to help you filter data: the media panel, the map panel, and the phylogeny panel. The media buttons correspond to Beginning, Slow/Rewind, Play, Stop, Fast Forward, Ending (from left to right). The animation will play the timeframe corresponding to the slider.

→ Drag the slider to the right (the present).

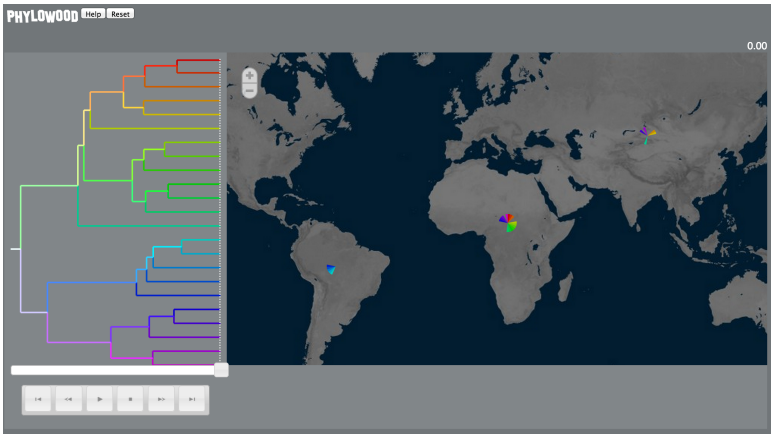


Figure 12.2: Phylowood frame showing distribution of extant taxon ranges.

→ Pan and zoom around the map.

Marker colors correspond to the phylogenetic lineages in the phylogeny panel. Markers are split into slices and (loosely) sorted phylogenetically, so nearby slices are generally closely related. At divergence events, a marker's radius is proportional to the marginal posterior probability the node was present in the area at that time. Between divergence events, marker's radius is simply an interpolation of the values at the two endpoints. Some information about geological constraints and cladogenic events is lost.

→ Mouseover an area to learn which lineage it belongs to and its presence probability.

Since it's difficult to see how specific clades evolve with so many taxa, Phylowood offers two ways to filter taxa from the animation. We call the set of a lineage, all its ancestral lineages towards the root, and all descendant lineages a phylogenetic heritage. The root's heritage is the entire clade. A leaf node's heritage is a path from the tip to the root.

→ Mouseover a lineage to temporarily highlight the lineage's heritage. Remove the mouseover to remove the highlight effect.

The highlight effect is temporary and quickly allows you to single out lineages of interest during animation. Phylowood also offers a masking effect that persists until an unmask command is issued.

→ Double-click the white root branch to mask the root node's heritage (all lineages). Single click a lineage to unmask that lineage's heritage.

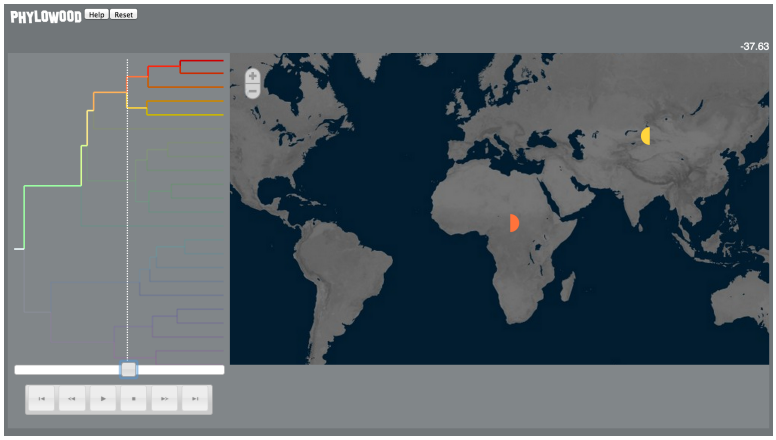


Figure 12.3: Phylowood frame highlighting the ancestral range for the MRCA extant lorises.

Now that the masking effects are in place, you're free to interact with other map components. In addition, the area of marker sizes is only distributed among unmasked lineages.

Visit `https://github.com/mlandis/phylowood/wiki` to learn more about Phylowood.

Phylowood is useful to understand whether your model generates sensible reconstructions. Keep in mind the animation and inference both use modern geographies, and that dispersal rates are not modeled to depend on geographical features. Notice the primate MRCA is widespread in two areas, the New World and Africa. The root age of the primate tree is about 80Mya, which is about 40Mya after East and West Gondwanaland split to separate modern Africa and South America, so vicariance did not give rise to New World Monkeys. This is corroborated by examining the fossil record, where the first appearance of primates in South America is about 35Mya. The standing theory is primates disperse from Africa to South America in a rare "sweepstakes." The naive primate biogeographic model described so far should produce more realistic reconstructions by taking continental drift into account.

12.3.3 Epoch models

To model the effects of geography, we will use *epoch* models. Rather than assuming the evolutionary process is constant with respect to time, it assumes the process is piecewise-constant. For example, Africa (along with all of Gondwanaland) split from Eurasia (and all of Laurasia) around 180 Mya. Africa merged with Eurasia only about 50 Mya. It is reasonable to expect that dispersal rates between Africa and Eurasia were lower before 50 Mya, and higher after 50 Mya until present.

To model this, we will specify one rate matrix that describes anagenic dispersal and extinction processes before for ages $t \geq 50$ and a second rate matrix for the process operating after $0 \leq t < 50$.

To proceed, first we will read in an `Atlas` file, which fully describes the geography per time interval. For more on the `Atlas` format, see Section XXX.

```
# read the atlas
atlas <- readAtlas("data/earth3.drift.atlas.txt")
n_epochs = atlas.nEpochs()
times <- atlas.epochTimes()
```

This atlas contains two epochs, each with three areas, and a single breakpoint at age 50 (50Mya). For example, the area for Africa in the second epoch from 50Mya – present reads

```
 {
    "latitude": 10.0,
    "longitude": 20.0,
        "dispersalValues": [ 0.1, 0.0, 1.0 ],
        "extinctionValues": [ 1.0 ],
    "name": "Africa"
}
```

The `dispersalValues` values share the ordering of the areas, and each area reports a value regarding it's relationship to other areas. Here, 0.1 in position 1 indicates the Atlantic Ocean obstructs dispersal from Africa to the Americas, while the 1.0 in position 3 indicates African primates may easily disperse into Eurasia by way of the Arabian Penninsula. (Note 0.0 in position 2 is the dispersal value from Africa to itself, so the value is arbitrary.)

To extract these values in matrix form

```
d_prior <- atlas.getValues("dispersal")
```

where, for example, the dispersal rate more than 50 Myr in the past (epoch 1) from Africa (area 2) to Eurasia (area 3) is accessed by

```
d_prior[1][2][3]
```

To use these empirical priors to rescale our naive priors

```
for (t in 1:n_epochs) {
       for (i in 1:n_areas) {
              for (j in 1:n_areas) {
                     if (i != j) {
                            d_raw[t][i][j] ~ dnExp( 10. )
                            mv[mvi++] = mvScale(d_raw[t][i][j], weight=2)
                            d[t][i][j] := d_raw[t][i][j] * abs(d_prior[t][i][j])
                     } else {
                            d[t][i][j] <- abs(0.)
                     }
              }
       }
}
```

We'll make no strong prior assumptions about epochs or areas differentially affecting extinction rates.

```
# set up extinction per epoch
e_prior <- atlas.getValues("extinction")
for (t in 1:n_epochs) {
       for (i in 1:n_areas) {
              e_raw[t][i] ~ dnExp( 10. )
              mv[mvi++] = mvScale(e_raw[t][i], weight=2)
              e[t][i] := e_raw[t][i] * abs(d_prior[t][i][1])
       }
}
```

Now we have dispersal rates and extinction rates per epoch in the matrices `d` and `e`.

Extant primate ranges do not appear capable of spanning all three areas simultaneously, so we might constrain the range evolution process to only allow ranges of size two

```
rangeSize <- simplex(0,1,1,0)
```

where the first and last elements are set to zero, so ranges cannot be size 0 or size 3. By calling `fnDECRoot` with the `rangeSize` parameter, we can also force the MRCA range size to be a single area.

```
rf := fnDECRoot( rep(1,n_states), rangeSize=simplex(0,1,0,0) )
```

Next, we'll create a rate matrix for each epoch

```
for (t in 1:n_epochs) {
       rates[t] ~ dnGamma(2,2)
       mv[mvi++] = mvScale(rates[t], weight=2)
       q[t] := fnDECRateMatrix(d[t], e[t], rangeSize)
}
```

with `rates` being an epochal rate multiplier, each with mean 1. Notice, we construct `q[t]` for each epoch `t` in `num_epochs`, using the epoch's dispersal values `d[t]` and extinction values `e[t]`. Finally, we wrap our vector of rate matrices, `q`, along with the epoch boundary times, `times`, and our epochal rates, `rates`

```
q_epoch := fnEpoch( Q=q, times=times, rates=rates )
```

Parameters like `clado_prob` and `clock_bg` still need to be created, as they were in the previous section.

Otherwise, the model is created as before, except passing `q_epoch` into `dnPhyloCTMCClado`.

```
m ~ dnPhyloCTMCClado( tree=psi, Q=q_epoch, rootFrequencies=rf, cladoProbs=clado_prob,
    branchRates=clock_bg, nSites=1, type="NaturalNumbers" )
m.clamp(data)
mdl = model(m)
```

This biologically and geographically informed model produces a more realistic range reconstruction

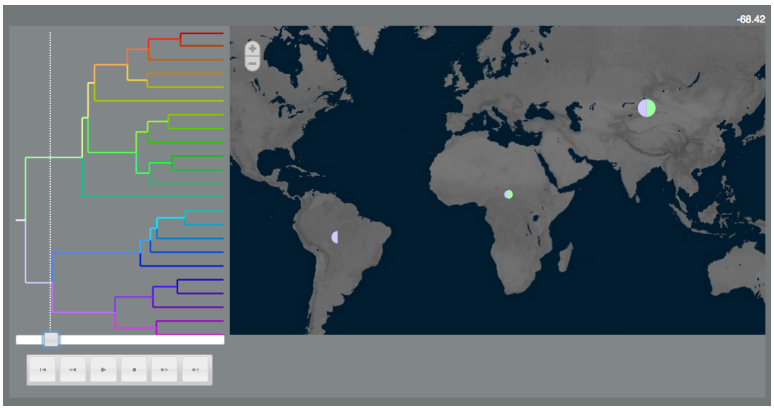


Figure 12.4: Phylowood frame showing Asian origin of primates, with subsequent dispersal into Africa and Americas.

This example model may be run by typing

```
RevBayes_scripts/biogeography_epoch.Rev}
```

which also produces the above Phylowood animation file.

Bibliography

Goldberg, E. E., L. T. Lancaster, and R. H. Ree. 2011. Phylogenetic inference of reciprocal effects between geographic range evolution and diversification. Systematic Biology 60:451–465.

Matzke, N. J. 2013. Probabilistic historical biogeography: new models for founder-event speciation, imperfect detection, and fossils allow improved accuracy and model-testing. Frontiers of Biogeography 5.

Ree, R. H., B. R. Moore, C. O. Webb, and M. J. Donoghue. 2005. A likelihood framework for inferring the evolution of geographic range on phylogenetic trees. Evolution 59:2299–2311.

Chapter 13

Many Area DEC

13.1 Large numbers of areas

13.1.1 Data augmentation

For small rate matrices, transition probabilities of beginning in state i and ending in state j equal the matrix exponential of the underlying rate matrix, scaled by the elapsed time of the process. This integrates over all unobserved transition events during the time interval t. Unfortunately, computing the matrix exponential scales poorly as the state space increases, *i.e.,* $O(n^4)$ for n states.

Alternatively, the probability of beginning in state i and ending in state j can be computed easily when the explicit series of event types and times are known. While we will never know the exact history of events, we can use stochastic mapping in conjunction with Markov chain Monte Carlo (MCMC) to repeatedly sample range evolution histories that are consistent with the ranges observed in the study taxa at the tips of the phylogeny. This technique is called data augmentation. It was first applied in phylogenetics to tertiary structure-dependent evolution of protein-coding nucleotide sequences (**?**), and later extended to range evolution in **?**.

Using data augmentation, we will approximate $\text{Prob}(\mathbf{X}_{aug}, \theta \mid \mathbf{X}_{obs}, T, M)$, where $\mathbf{X}_{obs}$ is the range data observed at the tips, $\mathbf{X}_{aug}$ is the distribution of ancestral range reconstructions over the phylogeny, T, where $\mathbf{X}_{aug}$ is inferred jointly with the parameters, θ, assuming the range evolution model, M, that describes $\mathbf{Q}$ above. From a Bayesian perspective, $\mathbf{X}_{aug}$ is effectively treated as part of the parameter space in the posterior distribution. Just as MCMC evaluates the posterior distribution for sampled parameter values, the same is done for sampled character histories. Thus, ancestral range reconstructions are a by-product of data augmentation, which are often of primary interest to biogeographers.

You may wonder why matrix exponentiation works fine for molecular substitution models and large multiple sequence alignments. It is the non-independence of character evolution that induces large state spaces, along with cladogenic processes that affect the entire set of characters (e.g. non-identical range inheritance following speciation). Molecular substitution models typically assume each character in an alignment evolves independently, which may be justified by the ability of recombination to degrade linkage disequilibrium over geological time scales. Conveniently, this keeps $\mathbf{Q}$ small even for datasets with many sites.

13.1.2 Large rate matrices and distance-dependent dispersal

A rate matrix may be represented compactly as the rate function

$$q^{(a)}_{\mathbf{y},\mathbf{z}} = \begin{cases} \lambda_0 & \text{if } z_a = 0 \\ \lambda_1 & \text{if } z_a = 1 \\ 0 & \mathbf{y} \text{ and } \mathbf{z} \text{ differ in more than one area} \end{cases} .$$

where $\mathbf{y}$ and $\mathbf{z}$ are the "from" and "to" ranges and a is the area that changes, λ_0 is the per-area rate of loss, and λ_1 is the per-area rate of gain. For example, $q^{(1)}_{011,111}$ is the rate of range expansion for $011 \rightarrow 111$ to gain area 1, which equals λ_1. Note the rate of more than one event occurring simultaneously is zero, so a range must expand twice by one area in order to expand by two areas. This model is analogous to the Jukes-Cantor model for three independent characters with binary states, except the all-zero "null range" is forbidden.

Extending this idea, we may reasonably expect that a range expansion event into an area depends on which

nearby areas are currently inhabited, which imposes non-independence between characters. The transition rate might then appear as

$$q^{(a)}_{\mathbf{y},\mathbf{z}} = \begin{cases} \lambda_0 & \text{if } z_a = 0 \\ \lambda_1 \eta(\mathbf{y}, \mathbf{z}, a, \beta) & \text{if } z_a = 1 \\ 0 & \mathbf{y} = 00...0 \\ 0 & \mathbf{y} \text{ and } \mathbf{z} \text{ differ in more than one area} \end{cases} .$$

For this tutorial, you can take $\eta(\cdot)$ to adjust the rate of range expansion into area a by considering how close it is to the current range, $\mathbf{y}$ relative to the closeness of all other areas unoccupied by the taxon. The β parameter rescales the importance of geographic distance between two areas by a power law. Importantly, $\eta(\cdot) = 1$ when $\beta = 0$, meaning geographic distance between areas is irrelevant. Moreover, when $\beta > 0$, $\eta(\cdot) < 1$ when area a is relatively distant and $\eta(\cdot) > 1$ when area a is relatively close. See **?** for a full description of the model.

? Write down a rate function where the per-area rates of gain and loss depend on the number of currently occupied areas.

13.1.3 Specifying a data augmented DEC model

Start to create helper variables for file handling,

```
area_fn = "data/earth25.still.atlas.txt"
data_fn = "data/primates_bg_n25.nex"
tree_fn = "data/primates.tree"
out_fn = "output/bg_large"
```

read in our tree,

```
psi <- readTrees( tree_fn )[1]
```

populate our range observations,

```
data   = readDiscreteCharacterData( data_fn )
```

and read in our geographical information (for details, see Section XX).

```
atlas  = readAtlas( area_fn)
n_areas = atlas.nAreas()
```

Lastly, create index variables to populate our move and monitor vectors,

```
mi = 0
```

and assign the number of generations to run the MCMC analysis

```
ngen = 500000
```

To later interpret ancestral state monitors phylogenetically, save a copy of `tree` annotated with internal node indexes.

```
write(psi,filename=tree_fn+".index.tre")
```

Proceeding with the model configuration, we'll first create our rate matrix that determines the rate of per-area gain and loss given the current geographical layout of the range. In `RevBayes`, data-augmented CTMC analyses require `RateMap` functions to determine event rates, which differ from the familiar `RateMatrix` functions that include `fnJC` and `fnGTR` as members. In their simplest form, `RateMap` functions generate the rates of change over the full set of characters, where each character evolves according to a provided `RateMatrix` function. Additionally, a `RateMap` accepts a rate modifier function that induces some correlation structure to character change evolution. In this section, we'll be creating a biogeographic `RateMap` function for the dispersal-extinction process given in Section 13.1.2.

First, create a biogeographical clock to scale the rate of range evolution.

```
clock_bg ~ dnExponential(10)
moves[++mi] = mvScale(clock_bg, lambda=0.5, weight=5.0)
```

Next, instantiate a simplex of gain and loss rates, distributed by a flat Dirichlet prior,

```
glr ~ dnDirichlet([1,1])
moves[++mi] = mvSimplexElementScale(glr, alpha=30.0, weight=5.0)
```

and use deterministic nodes to assign the rates nicknames.

```
r_gain := glr[1]
r_loss := glr[2]
```

Insert the simplex into the rate matrix q_{area}, which gives the average rate of area gain and loss per area.

```
q_area := fnFreeBinary(glr)
```

Next, we will create `dp` to represent the β parameter, which determines the importance of geographical distance to dispersal. Remember that values of β far from zero means distance is important. So, if we we assign a prior that pulls β towards zero, then posterior values of β far from zero indicate the range data are informative of the importance of distance to dispersal. We'll use an exponential distribution with mean 1.0 as a prior for `dp`.

```
dp ~ dnExponential(10.0)
moves[++mi] = mvScale(x=dp, lambda=0.5, tune=true, weight=5.0)
```

We will also create a deterministic node to modify the rate of dispersal between areas by evaluating `dp` and `atlas`. This node is determined by the function `fnBiogeoGRM`, where GRM stands for "geographical rate modifier", and plays the role of the $\eta(\cdot)$ rate-modifier function mentioned earlier. We will tell the `fnBiogeoGRM` function to modify dispersal rates based on distances and whether or not the area exists during an epoch.

```
grm := fnBiogeoGRM(atlas=atlas, distancePower=dp, useDistance=true)
```

Now we need a deterministic node to represent the rate matrix, $\mathbf{Q}$. To determine the value of this node, we'll use the function `fnBiogeoDE` to assign our model parameters to transition rates as described in the introduction. As input, we'll pass our gain and loss rates, `q_area`, our geographical rate modifier, `grm`, and the biogeographical clock `clock_bg`. In addition, we'll inform the function of the number of areas in our analysis and whether we will allow species to be absent in all areas (i.e. have the null range).

```
q_range := fnBiogeoDE(gainLossRates=q_area, branchRates=clock_bg, geoRateMod=grm,
   numAreas=n_areas, forbidExtinction=true)
```

As with the simple DEC model, we assign a flat Dirichlet prior over cladogenic event type probabilities

```
clado_prob ~ dnDirichlet( [1, 1, 1] )
widespread_sympatry := clado_prob[1]
subset_sympatry   := clado_prob[2]
allopatry         := clado_prob[3]
moves[++mi] = mvSimplexElementScale(clado_prob, alpha=20.0, weight=5.0)
```

Finally, the data evolve according to a phylogenetic CTMC process. Here, we decalre a stochastic node using `dnPhyloDACTMC` where `DA` indicates the distribution requires data augmentation to compute the

likelihood rather than Felsenstein's pruning algorithm. To create the distribution, we must pass it our `tree` and `q_range` objects, but additionally inform the distribution that it will be using a biogeographic model, that it will introduce the simple cladogenic range evolution events described in **?** (`useCladogenesis=true`), and that it will assign zero probability to a transition away from the null range state.

```
m ~ dnPhyloDACTMC(tree=psi, Q=q_range, cladoProbs=clado_prob, type="Biogeo",
    forbidExtinction=true, useCladogenesis=true)
```

So we may evaluate the graphical model's likelihood, we tell the CTMC to observe the `data` object, which then primes the model with data-augmented character histories.

Now `m` has a defined likelihood value.

```
m.clamp(data)
m.lnProbability()
   -156.0288
```

To integrate over the space of possible range histories, we still need to add moves to propose new data augmented histories. The major challenge to sampling character histories is ensuring the character histories are consistent with the observations at the tips of the tree. The proposals in this tutorial use **?**'s rejection sampling algorithm, with some modifications to account for cladogenic events and epoch-based rate maps.

The basic idea is simple. Each time a character history proposal is called, it selects a node at random from the tree. Branch history proposals propose a new character history to a single randomly-chosen branch. Node history proposals propose a new character history for the three branches connecting to a randomly chosen node, in addition to the cladogenic state of the node itself.

For each proposal, each character's history is resampled with probability `lambda` – i.e. `lambda=0.01` resamples 1% of characters, while `lambda=1.` resamples all characters. Once the new character history is proposed, the likelihood of the model is evaluated and the MCMC accepts or rejects the new state according to e.g. the Metropolis-Hastings algorithm.

Cladogenic events require special considerations during sampling, so we indicate `type="Biogeo"`. Currently, only rejection-sampling is available for cladogenic histories, hence `proposal="rejection"`. Finally, we apply high `weight` values to the proposals since the larger the state space is, the more character history proposals needed to effectively integrate over the space of sample paths.

Let's create the character history moves as follows: conservative character history updates for paths and nodes, with `lambda=0.05`

```
moves[++mi] = mvCharacterHistory(ctmc=m, qmap=q_range, tree=psi, lambda=0.05,
         type="Biogeo", graph="node", proposal="rejection", weight=n_nodes*5)
moves[++mi] = mvCharacterHistory(ctmc=m, qmap=q_range, tree=psi, lambda=0.05,
         type="Biogeo", graph="branch", proposal="rejection", weight=n_nodes)
```

and the same proposals for moderately-sized character history updates, with `lambda=0.2`

```
moves[++mi] = mvCharacterHistory(ctmc=m, qmap=q_range, tree=psi, lambda=0.2,
         type="Biogeo", graph="node", proposal="rejection", weight=n_nodes*5)
moves[++mi] = mvCharacterHistory(ctmc=m, qmap=q_range, tree=psi, lambda=0.2,
         type="Biogeo", graph="branch", proposal="rejection", weight=n_nodes)
```

and radical updates

```
moves[++mi] = mvCharacterHistory(ctmc=m, qmap=q_range, tree=psi, lambda=1.0,
         type="Biogeo", graph="node", proposal="rejection", weight=n_nodes*5)
moves[++mi] = mvCharacterHistory(ctmc=m, qmap=q_range, tree=psi, lambda=1.0,
         type="Biogeo", graph="branch", proposal="rejection", weight=n_nodes)
```

Next, create the model object,

```
my_model = model(m)
```

and monitors for our simple parameters.

```
monitors[1] = mnScreen(clock_bg, r_gain, r_loss, dp, subset_sympatry, allopatry,
    widespread_sympatry, printgen=100)
monitors[2] = mnFile(clock_bg, r_gain, r_loss, dp, subset_sympatry, allopatry,
    widespread_sympatry, filename=out_fn+".params.txt", printgen=10)
```

Like any parameter, we can sample the augmented range histories from the MCMC to approximate the posterior distribution of range histories. This is statistically equivalent to generating ancestral state reconstructions from a posterior distribution via stochastic mapping. We will extract these reconstructions using special monitors designed for the `dnPhyloDACTMC` distribution.

Next, we will create `mnCharHistoryNewick` monitors to record the sampled character history states for each node in the tree. This monitor has two `style` options: `counts` reports the number of gains and losses per branch in a tab-delimited Tracer-readable format; `events` reports richer information of what happens along a branch, anagenically and cladogenically, using an extended Newick format. How to read these file formats will be discussed in more detail in Section 13.1.4.

```
monitors[3] = mnCharHistoryNewick(filename=out_fn+".events.txt", ctmc=m, tree=psi,
    printgen=100, style="events")
monitors[4] = mnCharHistoryNewick(filename=out_fn+".counts.txt", ctmc=m, tree=psi,
    printgen=100, style="counts")
```

As our last monitor,`mnCharHistoryNhx` records character history values throughout the MCMC analysis, then stores some simple posterior summary statistics as a Nexus file. These summary statistics could be computed from the previously mentioned monitor output files, but `mnCharHistoryNhx` provides a simple way to produce Phylowood-compatible files. We will also discuss this file's format in more detail later in the tutorial.

```
monitors[5] = mnCharHistoryNhx(filename=out_fn+".phw.txt", ctmc=m, tree=psi, atlas=atlas
    , samplegen=100, maxgen=ngen, burnin=0.5)
```

Finally, create the MCMC object

```
my_mcmc = mcmc(my_model, monitors, moves)
```

and run

```
my_mcmc.run(generations=ngen)
```

The posterior surface over character histories is highly multimodal, so this analysis takes a long time to mix. If you're following this tutorial in a lab setting, you should proceed using the pre-analysed output files provided in `./example_output`.

13.1.4 Analysis output

We'll focus some attention on node 42, the most recent common ancestor of chimps and macaques, designated as the Old World most recent common ancestor (MRCA).

Biogeographic event counts from `mnCharHistoryNewick`

Recording stochastic mappings in a Tracer-compatible format requires some summarization. This monitor generates a tab-delimited file where the number of events of each type for each branch is recorded.

Open `./output/bg_3.counts.txt` in a text editor.

```
Iter    Posterior  Likelihood Prior  t_s0   t_s1   t_c0   t_c1   t_c2   t_c3   b0_s0
      b0_s1   b0_c    ...
0   -6518.54   -6519.25   0.706823   977 839 22 0   0   0   38  28  0   ...
10  -583.147   -585.333   2.186240   66  37  3   7   9   3   2   1   2   ...
20  -296.540   -297.340   0.799755   20  25  15  1   3   3   0   1   0   ...
30  -276.284   -277.257   0.972918   17  24  14  0   4   4   0   1   0   ...
40  -266.569   -266.948   0.379624   18  23  15  1   3   3   0   1   3   ...
...
```

For example, `b0_s1` gives the number of areas that are gained for the branch leading to the node indexed 0. Referencing FigTree, we see this corresponds to chimpanzees. `b0_c` gives the cladogenic event type that gives rise to the chimp lineage, where narrow sympatry, widespread sympatry, subset sympatry, and allopatry are recorded as `0`, `1`, `2`, and `3`, respectively. The columns `t_s0` and `t_s1` give the sum of events over all branches. `t_c0`, `t_c1`, `t_c2`, and `t_c3` give the total number of narrow sympatric, widespread sympatric, subset sympatric, and allopatric cladogenic events over the entire tree.

13.1.5 Biogeographic event histories from `mnCharHistoryNewick`

For more detailed data exploration, this analysis also provides annotated Newick strings with the complete character mappings for the tree. (This file format is a bit unwieldy and under revision.)

→ Open `./output/bg_3.events.txt` in a text editor.

Each iteration records the data-augmented character history (stochastic mapping) using metadata labels, which, for an internal node, looks like

```
[&index=42;nd=000000000000101000000000;pa=000000000000100000000000;ch0
   =000000000000101000000000;ch1=000000000000001000000000;cs=s;bn=41;ev={{t:0.138033,
   a:39.4458,s:1,i:15}}]
```

Having consulted FigTree earlier, we know node 42 is the Old World MRCA. The branch began with the range `pa=000000000000100000000000` and terminated in the state `nd=000000000000101000000000`. Since this node is not a tip node, it represents a speciation event, so the daughter ranges are also given, `ch0=000000000000101000000000` and `ch1=000000000000001000000000`. The cladogenic state for this speciation event was subset sympatric, `cs=s`, rather than sympatric (wide or narrow; `w` or `n`) or allopatric (`a`).

Anagenic dispersal and extinction events occurring along the lineage leading to node 42 are recorded in `ev`, where each event has a time (relative to the absolute branch length), absolute age, state (into), and character index (`t`, `a`, `s`, `i`, resp.). Potentially, `ev` contains multiple events. For this posterior sample, the MRCA of chimps and macaques dispersed into East Africa 39.4458 million years ago, which is consistent with the ranges recorded by `pa` and `nd`.

Although we have stochastic mappings under the posterior distribution, the remaining challenge is to summarize them into something useful. The Python script `bg_parse.py` is provided to manipulate this data format. Below are a few examples of interesting features of the posterior.

→ Open a Python console and read in the events.

```
> cd RevBayes_scripts
> python

...
```

```
>>> from bg_parse import *
>>> dd=get_events(fn="../output/bg_3.events.txt")
```

By default, `get_events()` extracts a dictionary-of-dictionaries from the posterior event samples. A dictionary is very much like a vector, except values are selected by keys, so this can be thought of as similar to a two-dimensional matrix. The first key corresponds to the node (branch) whose MCMC samples you'd like to retrieve, while the second dictionary's keys are the columns correspond to MCMC states and values specific to that node. For example, say we are interested in the last five cladogenic states sampled for node 42, Old World MRCA,

```
>>> dd[42].keys()
  ['ch1', 'iteration', 'bn', 'nd', 'ch0', 'prior', 'posterior', 'pa', 'cs', 'ev', '
    likelihood']
>>> dd[42]['cs'][-5:]
  ['widespread_sympatry', 'widespread_sympatry', 'widespread_sympatry', '
    widespread_sympatry', 'widespread_sympatry']
```

To get the `n=1` highest-valued sample for a branch by its posterior value

```
>>> get_best(dd[42],n=1,p='posterior')
{'prior': [-0.524716], 'iteration': [46700], 'bn': [41], 'nd': [[0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]], 'ch0': [[0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]], 'ch1': [[0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]], 'posterior': [-95.7676],
   'pa': [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]],
    'cs': ['widespread_sympatry'], 'ev': [[]], 'likelihood': [-95.2429]}
```

More data-exploration functions are found in `bg_parse.py`.

13.1.6 Exercises

- (Under construction.)

Part VIII

Phylogenetic Comparative Method

Chapter 14

Phylogenetic Comparative Analyses: Continuous Trait Evolution

14.1 Introduction

The subject of the comparative method is the analysis of trait evolution at the macroevolutionary scale. In a comparative context, many different questions can be addressed: tempo and mode of evolution, correlated evolution of multiple quantitative traits, trends and bursts, changes in evolutionary mode correlated with major key innovations in some groups, etc (for a good introduction see Harvey and Pagel 1991).

In order to correctly formalize comparative questions, the underlying phylogeny should always be explicitly accounted for. This point is clearly illustrated, in particular, by the independent contrasts method (Felsenstein 1985; Huelsenbeck and Rannala 2003). Practically speaking, the phylogeny and the divergence times are usually first estimated using a separate phylogenetic reconstruction software. In a second step, this time-calibrated phylogeny is used as an input to the comparative method. Doing this, however, raises a certain number of methodological problems:

- the uncertainty about the phylogeny (and about divergence times) is ignored
- the traits themselves may have something to say about the phylogeny
- the rate of substitution, and more generally the parameters of the substitution process, can also be seen as quantitative traits, amenable to a comparative analysis.

All these points are not easily formalized in the context of the step-wise approach mentioned above. Instead, what all this suggests is that phylogenetic reconstruction, molecular dating and the comparative method should all be considered jointly, in the context of one single overarching probabilistic model.

Thanks to its modular structure, `RevBayes` represents a natural framework for attempting this integration. The aim of the present tutorial is to guide you through a series of examples where this integration is achieved, step by step. It can also be considered as an example of the more general perspective of *integrative modeling*, which can be recruited in many other contexts.

14.2 Data and files

We provide several data files which we will use in this tutorial. You may want to use your own data instead. 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_lhtlog.nex`**: 2 life-history traits (endocranial volume (ECV), body mass; each for males and females separately) for 23 primate species (taken from the Anage database, De Magalhaes and Costa 2009). The traits have been log-transformed.
- **`primates.tree`**: A time calibrated phylogeny of the same 23 primates.

14.3 Univariate Brownian evolution of quantitative traits

As a first preliminary exercise, we wish to reconstruct the evolution of body mass in primates and, in particular, estimate the body mass of their last common ancestor. For this, we will assume that the

logarithm of body mass follows a simple univariate Brownian motion along the phylogeny. In a first step, we will ignore phylogenetic uncertainty: thus, we will assume that the Brownian process describing body mass evolution runs along a fixed time-calibrated phylogeny (with fixed divergence times), such as specified in the file **primates.tree**.

→ You may want to take the time to visualize the tree given in **primates.tree** as well as the matrix of quantitative traits specified by the **primates_lhtlog.nex** file, before going into the modeling work described below.

14.3.1 The model and the priors

A univariate Brownian motion $x(t)$ is parameterized by its starting value at the root of the phylogeny $x(0)$ and a rate parameter σ. This rate parameter tunes the amplitude of the variation per unit of time. Specifically, along a given time interval $(0, T)$, the value of X at time T is normally distributed, with mean $x(0)$ and variance $\sigma^2 T$:

$$x(T) \sim \text{Normal}\left(x(0), \sigma^2 T\right).$$

Concerning σ, we can formalize the idea that we are ignorant about the *scale* (the order of magnitude) of this parameter by using a log-uniform prior:

$$\sigma \sim \frac{1}{\sigma}.$$

Concerning the initial value $x(0)$ of the Brownian process at the root of the phylogeny. Alternatively, you may want to specify a normal distribution as the prior distribution on the root value if you have some prior information.

Finally, the tree topology ψ is, as mentioned above, fixed to some externally given phylogeny. The entire model is now specified: tree ψ, variance σ and Brownian process $x(t)$:

$$\begin{aligned} \sigma &\sim \frac{1}{\sigma}, \\ x(0) &\sim \text{Uniform}, \\ x(t) \mid \Psi, \sigma &\sim \text{Brownian}\left(x(0),\, \psi,\, \sigma\right). \end{aligned}$$

Conditioning the model on empirical data by clamping $x(t)$ at the tips of the phylogeny, we can then run a MCMC to sample from the joint posterior distribution on σ and x. Once this is done, we can obtain posterior means, medians or credible intervals for the value of body mass or other life-history traits for specific ancestors.

14.3.2 Programming the model in RevBayes

The problem of continuous trait evolution —just as for discrete trait evolution— along a phylogeny is that we do not know the values of the traits at the internal nodes. That means, that we need to treat the states at the internal nodes as additional parameters of the model. For discrete characters we use the sum-product (a.k.a. pruning) algorithm (Felsenstein 1981) to analytically integrate over all possible states at the internal nodes. For continuous characters (traits) similar methods have been proposed. In RevBayes you

have three main ways of specifying this model and running an analysis on it. The three approaches are: (1) phylogenetic independent contrasts using the reduced likelihood (REML), (2) Brownian motion using a phylogenetic covariance matrix, and (3) a full Brownian motion model using data augmentation. Each of these approaches has there advantages and disadvantages as will be explained below. Nevertheless, all approaches give the same results in terms of rate estimation.

14.3.3 Phylogenetic Independent Contrasts using the reduced likelihood (REML)

The reduced or restricted maximum likelihood (REML) method computes the probability of observing the continuous character at the tips by an analytical solution to integrate over the internal states (Felsenstein 1985). This analytical solution is very fast to compute and thus can be applied to large phylogenies and/or many independent characters. However, the REML method looses the information about the location of the root state and thus you cannot infer which state the root or other internal nodes have.

You do not need to understand the algorithm but we provide a sketch of the idea behind REML to give you some insights. REML compute the values at the internal nodes as the phylogenetic contrasts $x_k = x_i - x_j$ where x_i and x_j are the values of the child nodes in the phylogeny. Then, we can compute the probability of observing the contrast x_k using the probability density of a normal distribution with mean $\mu = 0$ and standard deviation $\sigma = \sqrt{\nu_i + \delta_i + \nu_j + \delta_j}$ where ν_i and ν_j are the (scaled) branch lengths leading to node i and j respectively. δ is the additional uncertainty that is propagated through the phylogeny and is compute by $\delta_k = ((\nu_i + \delta_i) * (\nu_j + \delta_j))/(\nu_i + \delta_i + \nu_j + \delta_j)$. These computations are done for you in the `RevBayes` distribution called **`dnPhyloBrownianREML`**.

In the directory **`RevBayes_scripts/`** you will find a script called **`primatesMass_BM_REML.Rev`**. This script implements the univariate Brownian model described above. Instead of re-typing the content of script entirely in the context of an interactive `RevBayes` session, you can instead run the script directly:

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

This script essentially reformulates what has been explained in the last subsection and serves as an example solution for you. For the later section you need to adjust the script.

Let us go through the script step by step in the `Rev` language. First, load the trait data:

```
contData <- readContinuousCharacterData("data/primates_lhtlog.nex")
```

If you type you will see that the continuous character data matrix contains several characters (columns).

```
contData

   Continuous character matrix with 23 taxa and 11 characters
   ==========================================================
   Origination:                primates_lhtlog.nex
   Number of taxa:             23
   Number of included taxa:    23
   Number of characters:       11
   Number of included characters: 11
   Datatype:                   Continuous
```

Since we only want the body mass (of females) we exclude all but the third character

```
contData.excludeAll()
contData.includeCharacter(3)
```

Next, load the time-tree from file. Remember that we use in this first simple example a fixed tree that we assume is known without uncertainty.

```
treeArray <- readTrees("data/primates.tree")
psi <- treeArray[1]
```

→ You may want to look at this tree before by loading the **primates.tree** in FigTree or any other tree visualization software.

As usual, we start be initializing some useful helper variables. For example, we set up a counter variable for the number of moves that we already added to our analysis. This will make it much easier if we extend the model or analysis to include additional moves or to remove some moves.

```
mi = 0
```

Then, we define the overall rate parameter σ which we assign a (truncated) log-uniform prior. Note that it is more efficient in Bayesian inference to specify a uniform prior and then to transform the parameter which we will use here:

```
logSigma ~ dnUniform(-5,5)
sigma := 10^logSigma
```

Using this approach we have specified a prior probability distribution on **sigma** between 10^{-5} to 10^{5} which should be broad enough to include all reasonable values.

Since the rate of trait evolution **logSigma** is a stochastic variable and we want to estimate it, we need to add a sliding move on it. Remember that the sliding move proposes new values drawn from a window with width **delta** and is centered around the current values; thus it slides through the parameter space together with the current parameter value.

```
moves[++mi] = mvSlide(logSigma, delta=1.0, tune=true, weight=2.0)
```

Next, define a random variable from the univariate Brownian-Phylo-REML process, which we will call **logmass**. We need to provide the tree variable **psi**, some branch-specific rate multiplier parameter which we simply set to 1, the shared rate for this site **sigma** and the number of sites (number of continuous traits) that we use.

```
logmass ~ dnPhyloBrownianREML(psi, branchRates=1.0, siteRates=sigma, nSites=1)
```

Now, condition the Brownian model on empirically observed values for body mass in the extant taxa.

```
logmass.clamp( contData )
```

The model is now entirely specified and we can create a model object containing the entire model graph by providing it with only one of our model variables, *e.g.*,**sigma**.

```
mymodel = model(sigma)
```

To see what it happing during the MCMC let us make a screen monitor that tracks the rate **sigma**.

```
monitors[1] = mnScreen(printgen=10, sigma)
```

Additionally, we'll use a file monitor that does the same thing, but directly stores the values into a file.

```
monitors[2] = mnFile(filename="output/primates_mass_REML.log", printgen=10, separator = TAB,
    sigma)
```

We can finally create a mcmc, and run it for a good 100 000 cycles after we did a burnin phase of 10 000 iterations:

```
mymcmc = mcmc(mymodel, monitors, moves)
mymcmc.burnin(generations=10000,tuningInterval=500)
mymcmc.run(100000)
```

Exercises

- Run the model.
- using **Tracer**, visualize the posterior distribution on the rate parameter **sigma**
- calculate the 95% credible interval for the rate of evolution of the log of body mass (σ)

14.3.4 Phylogenetic covariance matrix

The second method that we will use creates a phylogenetic covariance matrix. The phylogenetic covariance matrix method integrates over the states at the internal nodes as well but uses instead a multivariate normal distribution. The key advantage is that this method provides information about the root state since it models the root state as an additional parameter of the model. The disadvantage is that it is very computationally intensive. That means, that the phylogenetic covariance matrix approach may take long for very large data sets (at least in its current implementation).

→ Copy the file **primatesMass_BM_REML.Rev**, name it for example **primatesMass_BM_Cov.Rev** and start editing it.

In the previous example, the REML approach, we did not specify a parameter for the state at the root. In this exercise, we need this additional parameter. Let us use a uniform prior distribution on the logarithm of the root mass. A uniform prior between -100 and 100 should be diffuse enough. Just image how big or small an individual needs to be if it has body mass smaller than exp(-100) or larger than exp(100).

```
rootlogmass ~ dnUniform(-100,100)
```

Next, we'll specify a sliding move that proposes new values for the **rootlogmass** randomly drawn from a window centered around the current value.

```
moves[++mi] = mvSlide(rootlogmass,delta=10,tune=true,weight=2)
```

Finally, we need to substitute the **dnPhyloBrownianREML** by **dnPhyloBrownianMVN** to use the phylogenetic covariance matrix approach. Again, we provide the tree variable **psi**, some branch-specific rate multiplier parameter which we simply set to 1, the shared rate for this site **sigma** and the number of sites (number of continuous traits) that we use.

```
logmass ~ dnPhyloBrownianMVN(psi, branchRates=1.0, siteRates=sigma, rootStates=rootlogmass,
    nSites=1)
```

This will automatically connect the parameters of the model together.

Additionally, we now have the extra parameter **rootlogmass** which we want to monitor. Thus we need to replace the file monitor and use instead.

```
monitors[2] = mnFile(filename="output/primates_mass_Cov.log", printgen=10, separator = TAB,
    sigma, rootlogmass)
```

→ Don't forget to change the output file names in the monitors, otherwise your old analyses files will be overwritten.

Exercises

- Run the analysis.
- Using **Tracer**, visualize the posterior distribution on the rate parameter **sigma** and the **rootlogmass**
- How does the posterior distribution of **sigma** looks compared with the first analysis?
- Calculate the 95% credible interval for the rate of evolution of the log of body mass (σ) and the **rootlogmass**

14.3.5 Data augmentation

The third method to we will use is a data augmentation method. The data augmentation method uses explicitly the states at the internal nodes. We will specify the full model in Rev. That means that we will create a random variable for each node of the tree using a for loop. Each trait is then simply assign a normal distribution, as we described above in the model description. The advantage of this method is that you estimate the values at the internal nodes directly and that you have full control about modifying any part of the model. The disadvantage is that the MCMC algorithm will be harder if you want to jointly estimate the phylogeny, although the likelihood computation is very fast. The problem is the mixing of the MCMC because proposing new trees involves proposing new good values for the states at the internal nodes.

→ Copy the file **primatesMass_BM_REML.Rev**, name it for example **primatesMass_BM_DA.Rev** and start editing it.

]As in the previous example we will use a uniform prior distribution on the logarithm of the root mass.

```
rootlogmass ~ dnUniform(-100,100)
```

Again, we'll specify a sliding move that proposes new values for the **rootlogmass** randomly drawn from a window centered around the current value.

```
moves[++mi] = mvSlide(rootlogmass,delta=10,tune=true,weight=2)
```

In order to create the random variables for the internal states we need to know the number of nodes and the number of tips. We will store these as some helper variables.

```
numNodes = psi.nnodes()
numTips = psi.ntips()
```

Now we are ready to specify the Brownian motion model for each branch. That is, we simply specify a new normal distributed random variable for each node with mean being equal to the value of the parent variable and the standard deviation being equal to the product of the square root of the branch length and our rate parameter **sigma**. We store all the variables in the vector **logmass**. Then we are able to access the value at the parent node using the index of the parent node, which we can obtain from the tree using the function **psi.parent(i)**. Similarly, since the variance depends on the branch length we retrieve the branch length of node with index **i** using the function **psi.branchLength(i)**.

First we need to copy (create a reference to) the **rootlogmass**

```
logmass[numNodes] := rootlogmass
```

Let us start by creating the random variables for the internal nodes. Remember that the variance is equal to **sigma**-squared times the branch length, and we need to compute the square root of it to obtain the standard deviation.

```
# univariate Brownian process along the tree
# parameterized by sigma
for (i in (numNodes-1):(numTips+1) ) {
  logmass[i] ~ dnNormal( logmass[psi.parent(i)], sd=sigma*sqrt(psi.branchLength(i)) )
  # moves on the Brownian process
  moves[++mi] = mvSlide( logmass[i], delta=10, tune=true ,weight=2)
}
```

You may have noticed that we specified in the loop a move for each internal **logmass**. This is because we want to use the MCMC algorithm to integrate over the uncertainty in the states.

Next, we repeat the same loop but now for the tip nodes. Instead of applying a move to each tip node we will clamp the nodes. The nodes will be clamped with the data that we read in before.

```
for (i in numTips:1 ) {
  logmass[i] ~ dnNormal( logmass[psi.parent(i)], sd=sigma*sqrt(psi.branchLength(i)) )

  # condition Brownian model on quantitative trait data (second column of the dataset)
  logmass[i].clamp(contData.getTaxon(psi.nodeName(i))[1])
}
```

Here we do not need a **dnPhyloBrownianREML** or **dnPhyloBrownianMVN** distribution anymore because we explicitly instantiated the model. Note that the approach we have taken using the loop is the tree-plate approach described in (Höhna et al. 2014)

Since we have several additional parameters —the states at the internal nodes— we will use a model monitor to write to file instead.

```
monitors[2] = mnModel(filename="output/primates_mass_DA.log", printgen=10, separator = TAB)
monitors[3] = mnExtNewick(filename="output/primates_mass_DA_ext.trees", isNodeParameter=TRUE,
    printgen=10, separator = TAB, tree=psi, logmass)
```

To get the annotate tree we use the map tree function.

```
treetrace = readTreeTrace("output/primates_mass_DA_ext.trees", treetype="clock")
map_tree = mapTree(treetrace,"output/primates_mass_DA_ext_MAP.tree")
```

→ Don't forget to change the output file names in the monitors, otherwise your old analyses files will be overwritten.

Exercises

- Run the analysis.
- Using **Tracer**, visualize the posterior distribution on the rate parameter **sigma** and the **rootlogmass** and the internal states.
- How does the posterior distribution of **sigma** looks compared with the first and second analysis?
- Calculate the 95% credible interval for the rate of evolution of the log of body mass (σ) and the **rootlogmass**. Have they changed?

14.3.6 Brief intermediate summary

In the above exercises you have analyzed the log-transformed body mass using a Brownian motion (BM) along a phylogeny. We assumed the phylogeny to be known without error and were primarily interested in the rate how fast the body mass evolves. The exercises showed you three different ways how to approach the BM model, each having its advantages and disadvantages. You will need to decide for your analysis which approach is most appropriate. Our intention was mainly to show you some of the flexibility in RevBayes how to specify the same model in many different ways, exposing sometimes more and sometimes

less of the internal model graph structure. As an extra exercise you could start thinking about how to extend the basic BM.

14.4 Multiple independently evolving traits

The Brownian motion can easily be applied to multiple traits. The exactly same procedure and model will be applied as above and thus we will skip the repetitive description. The interesting questions that you can ask about multiple traits is —amongst others— if rates between traits vary.

As an example we use now the first four characters of the data matrix, which are: 1) female log endocranial volume (ECV), 2) male log ECV, 3) female log body mass and 4) male log body mass.

```
contData <- readContinuousCharacterData("data/primates_lhtlog.nex")
contData.excludeCharacter(5:11)
```

In the first simple example we assume that all site rates are equal, thus we use a rate multiplier we we call **perSiteRates**.

```
perSiteRates <- [1,1,1,1]
```

Furthermore, we have now four characters and hence we will estimate a root state for each one of them. We use the same prior probability for every character, just as above for the single character example.

```
for (i in 1:4) {
   rootlogmass[i] ~ dnUniform(-100,100)
   moves[++mi] = mvSlide(rootlogmass[i],delta=10,tune=true,weight=2)
}
```

Finally, we bring together all the parameter to specify our Brownian motion model along the tree. Note that we specify here the site rates as the product of the global rate **sigma** time the sites specific rates (which are, however all equal in this first exercie).

```
logmass ~ dnPhyloBrownianMVN(psi, branchRates=1.0, siteRates=sigma*perSiteRates, rootStates=
    rootlogmass, nSites=4)
```

After this you need to create the model using the **model** function, create your monitors, create the MCMC and finaly run the MCMC.

Exercises

- Run the analysis.
- Using **Tracer**, visualize the posterior distribution on the rate parameter **sigma** and the four **rootlogmass** values.
- Compare the results to the previous single trait analyses?

14.4.1 Independent site rates

```
perSiteRates ~ dnDirichlet([1,1,1,1])
moves[++mi] = mvSimplexElementScale(perSiteRates,alpha=10,tune=true,weight=4)
```

Exercises

- Run the analysis.
- Using **Tracer**, visualize the posterior distribution on the per site rate parameter **perSiteRates**.
- How are the estimate per site rates compared to each other?

14.4.2 Partially shared site rates

We hope that your results in the previous analysis showed that the rates for females and males for both the ECV and the body mass are very similar. This motivates us to specify an explicit model where the rates between ECV and body mass evolution are different but shared between females and males. We will model the difference using the **siteRateDiff** parameter which is drawn from a Beta(1,1) distribution, which is essentially a uniform distribution between 0 and 1. We still choose the Beta distribution because it is easier to specify more informative priors.

```
siteRateDiff ~ dnBeta(1,1)
```

Next, we specify a sliding move on the **siteRateDiff**.

```
moves[++mi] = mvSlide(siteRateDiff,delta=10,tune=true,weight=2)
```

To obtain the rates we use **siteRateDiff** as the rate for the ECV evolution in females and males and 1-**siteRateDiff** as the rate for the body mass evolution in females and males. Note, however, that we need a small workaround here. Rev is a strictly typed language, which you may have noticed. The per site rates need to be positive real number (negative rates obviously don't make sense). However, the difference between two number is not guaranteed to be positive, only if we know that the first term is larger than the second. This is exactly the case for 1-**siteRateDiff** but RevBayes doesn't know this so we need to tell it by using the absolute value **abs** function.

```
perSiteRates[1] := siteRateDiff
perSiteRates[2] := siteRateDiff
perSiteRates[3] := abs(1.0 - siteRateDiff) # specify the type here
perSiteRates[4] := abs(1.0 - siteRateDiff)
```

You could have specified many other prior distribution on the relative rate (or ratio) such as a uniform prior distribution and then use logit or hyperbolic tangent transformation. The only important factor is that the rates are somehow normalized because we use the global, shared rate **sigma**. Using the beta distribution makes the rates automatically normalized because the two different rates sum to one. Note that in the previous all four rates summed to 1.0 and here the two different rates sum to 1.0. This has the effect that we should estimate the global rate **sigma** to be half of what you estimated previously. You may want to check this once you ran the analysis and loaded the output into Tracer.

The remaining functions are the same as in the previous example.

Exercises

- Run the analysis.
- Using **Tracer**, visualize the posterior distribution on the per site rate parameter **perSiteRates**.
- Does the value of equal rate (**siteRateDiff**=0.5) fall into the 95% credible interval?
- If the rate does not fall into the 95% credible interval, should we reject the hypothesis that the underlying rates are equal?

14.4.3 Model selection

Our previous analysis explored the parameters of the rate of evolution. You will have seen that the rate of evolution was smaller for the ECV compared with the rate of evolution of body size. However, we did not perform a statistical test to reject the hypothesis if the underlying rates between the different characters are significantly different. Therefore, we will perform a marginal likelihood estimation for all three model: 1) the equal rates, 2) the independent rates, and 3) the shared rates.

→ You need to adopt the three scripts of your previous analysis.

Previously you created an **mymcmc** object using the **mcmc** function. Now, you need to replace this object with the power posterior MCMC object.

```
pow_p = powerPosterior(mymodel, moves, monitors, "output/pow_p_BM_equal.out", cats=100,
    sampleFreq=10)
```

Next, you need to run the burnin and afterwards run the power posterior sampler. This will create 101 output files (100 stepping stones and one for the posterior) logging the values into the monitors that you have specified before (we recommend you not to use a screen monitor). You could look into each of these files to check that you obtain sufficiently many samples for each stone.

```
pow_p.burnin(generations=10000,tuningInterval=250)
pow_p.run(generations=10000)
```

Use stepping-stone sampling to calculate marginal likelihoods.

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

`RevBayes` will print to the screen the estimated marginal likelihood. Write down this value and repeat the exercise for the other two models.

Exercises

- Run the three different marginal likelihood estimations and write down the marginal likelihood.
- Compute the log-Bayes factors by taking the difference of the alternative model (independent rates & shared rates) and the null hypothesis (equal rates).
- Is any of the log-Bayes factors larger than 1.16 for substantial support of the alternative hypothesis (or 2.3 for strong support)?

14.5 Estimating the phylogeny from continuous character data

In the motivation we argued that the phylogeny should be estimated together with the parameters of the evolutionary process. Only for simplicity we used fixed trees in the previous exercises. Instead of using a tree fixed to a pre-defined value, the tree should now be moved during the MCMC. Implementing this joint model in `RevBayes` is just a matter of adding the following features to the model defined in the previous section (after duplicating the script). You may want to use the REML approach for computational efficient (*i.e.,* good MCMC mixing) although the phylogenetic covariance matrix and the data augmentation are generally possible too.

We need to get some useful variables from the data so that we will be able to specify the tree prior below. These variables are the number of tips, the number of nodes and the names of the species which we all can query from the continuous character data object.

```
numTips = contData.ntaxa()
names = contData.names()
numNodes = numTips * 2 - 1
```

Instead of having a fixed tree as in the previous, we should now define a *random* tree. We use a birth death prior with prior distributions on the **`diversification`** rate and **`turnover`** rate. The **`diversification`** rate corresponds to the rate of growth of the tree and the **`turnover`** rate corresponds to the rate how quickly a species is replaced by a new one. This parametrization has the advantage that we can use meaningful prior distributions from the fossil record.

```
diversification ~ dnLognormal(0,1)
turnover ~ dnGamma(4,4)
```

Then, we compute deterministically the **`speciation`** and **`extinction`** rate from the **`diversification`** rate and **`turnover`** rate.

```
speciation := diversification + turnover
extinction := turnover
```

Instead, you could also use directly the speciation and extinction rates, *e.g.,* endowed with some diffuse exponential prior.

```
# rescaling moves on speciation and extinction rates
moves[++mi] = mvScale(diversification, lambda=1, tune=true, weight=3.0)
moves[++mi] = mvScale(turnover, lambda=1, tune=true, weight=3.0)
```

The phylogeny that we used are obviously not a complete sample of all the species and you should take the incomplete sampling into account. We will simply use an empirical estimate of the fraction of species which we included in this study. For more information about incomplete taxon sampling see Höhna et al. (2011) and Höhna (2014).

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

Now we are able to specify of tree variable **psi** which is drawn from a constant rate birth-death process. We will condition the age of the tree to be 75 million years old which is approximately the crown age of primates, although this estimate is still debated. We only condition here on the crown age for simplicity because we do not use any other fossil calibration.

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

Note that, here, we do not have included any fossil information: we are merely doing *relative* dating.

The first moves on the tree which we specify are moves that change the node ages. The first move randomly picks a subtree and rescales it, and the second move randomly pick a node and uniformly proposes a new node age between its parent age and oldest child's age.

```
moves[++mi] = mvSubtreeScale(psi, weight=5.0)
moves[++mi] = mvNodeTimeSlideUniform(psi, weight=10.0)
```

We also need moves on the tree topology to estimate the phylogeny. The two moves which you use are the nearest-neighbor interchange (NNI) and the fixed-nodeheight-prune-and-regraft (FNPR) (Höhna and Drummond 2012).

```
moves[++mi] = mvNNI(psi, weight=5.0)
moves[++mi] = mvFNPR(psi, weight=5.0)
```

We are essentially done now. We only need to add a new monitor for the tree so that we can monitor and build the maximum a posteriori tree later.

```
monitors[3] = mnFile(filename="output/primates_mass_multiBM_tree.trees", printgen=100,
    separator = TAB, psi)
```

→ A short analysis of 50 000 iterations will be sufficient.

Exercises

- Run the analysis.
- Create the maximum a posteriori (MAP) tree **readTreeTrace** and **mapTree**.
- Look at the estimated tree and compare it to the tree you used before.

- How does the posterior distribution of **sigma** looks compared with the first and second analysis?
- Compute the posterior probabilities of each tree using the **treetrace.summarize()** command.
- What is the posterior probability of the best tree?

14.6 Including information about the tree from molecular data

Starting from the model implemented in the last section, we now want to account for phylogenetic uncertainty using information contained in molecular data. As first pointed out by Huelsenbeck and Rannala (2003), this can easily be done in a Bayesian framework, through the use of a joint model combining sequence data and quantitative traits. Specifically:

- two data sets are loaded: one for sequence data and one for quantitative traits
- a tree is defined under birth-death process
- a Brownian model is defined over the tree (just as described in the previous section)
- the Brownian model is conditioned on the quantitative trait data
- a substitution model is defined over the same tree (GTR+Gamma)
- the substitution model is conditioned on the molecular sequence data.

14.6.1 Programming the model in RevBayes

First, we create a substitution model, just like what you probably did in previous tutorial (*e.g.,* RB_CTMC_Tutorial). In a first step, we will use a GTR+Gamma model. Start by loading the sequence data matrix specified in **data/primates_cytb.nex**.

```
seqData <- readDiscreteCharacterData("data/primates_cytb.nex")
```

We can use a flat Dirichlet prior density on the exchangeability rates **er** and the the base frequencies **pi**.

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

Now add the simplex scale move one each the exchangeability rates **er** and the stationary frequencies **pi** to the moves vector:

```
moves[++mi] = mvSimplexElementScale(er)
moves[++mi] = mvSimplexElementScale(pi)
```

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

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

The next part of the substitution process is the rate variation among sites. We will model this using the commonly applied 4 discrete gamma categories which only have a single parameter **alpha**. Let us specify the rate of **alpha** to 0.05 (thus the mean will be 20.0).

```
alpha_prior <- 0.05
```

Then create a stochastic node called **alpha** with an exponential prior:

```
alpha ~ dnExponential(alpha_prior)
```

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

```
gamma_rates := fnDiscretizeGamma( alpha, alpha, 4 )
```

The random variable that controls the rate variation is the stochastic node **alpha**. We will apply a simple scale move to this parameter.

```
moves[++mi] = mvScale(alpha, weight=2.0)
```

This finishes the substitution process part of the model.

Then next part of the model is the clock model. Here we need a clock model because we work on a time tree. We use our exponentiated uniform distribution to specify a flat prior distribution which is scale invariant.

```
logClockRate ~ dnUniform(-5,5)
clockRate := 10^logClockRate
```

We will apply a sliding window move to the **logClockRate**.

```
moves[++mi] = mvSlide(logClockRate, delta=0.1, tune=true, weight=2.0)
```

Remember that you need to call the **PhyloCTMC** constructor to include the new site-rate parameter:

```
seq ~ dnPhyloCTMC(tree=psi, Q=Q, siteRates=gamma_rates, branchRates=clockRate, type="
    DNA")
```

Finally we need to attach the molecular sequence data to our model.

```
seq.clamp(seqData)
```

You may have wondered how the continuous trait model and the molecular sequence model are combined. This happened automatically when you used the same tree parameter **psi** for both models. Since both models share now the same parameter, they will be used for a joint analyses. The model function thus will construct the joint model graph for the continuous trait model as well as the molecular sequence model.

After you ran the MCMC analyses, read in the tree trace. We will assume that you used a tree monitor and called the file *output/primates_joint.trees*. Change the name if you used a different one.

```
treetrace = readTreeTrace("output/primates_joint.trees", "clock")
```

Then build the maximum a posteriori tree.

```
map = mapTree( file="primates_joint.tree", treetrace )
```

Write this model and make sure that it runs when you give it to `RevBayes`.

Exercises

- Run the analysis.
- Using **Tracer**, visualize the posterior distribution on the rate parameter **sigma** and compare it to the previous analyses.
- Look at the MAP tree which you estimated now.

Bibliography

De Magalhaes, J. and J. Costa. 2009. A database of vertebrate longevity records and their relation to other life-history traits. Journal of evolutionary biology 22:1770–1774.

Felsenstein, J. 1981. Evolutionary trees from DNA sequences: A maximum likelihood approach. Journal of Molecular Evolution 17:368–376.

Felsenstein, J. 1985. Phylogenies and the comparative method. American Naturalist Pages 1–15.

Harvey, P. H. and M. D. Pagel. 1991. The comparative method in evolutionary biology vol. 239. Oxford university press Oxford.

Höhna, S. 2014. Likelihood Inference of Non-Constant Diversification Rates with Incomplete Taxon Sampling. PLoS One 9:e84184.

Höhna, S. and A. J. Drummond. 2012. Guided Tree Topology Proposals for Bayesian Phylogenetic Inference. Systematic Biology 61:1–11.

Höhna, S., T. A. Heath, B. Boussau, M. J. Landis, F. Ronquist, and J. P. Huelsenbeck. 2014. Probabilistic Graphical Model Representation in Phylogenetics. Systematic Biology 63:753–771.

Höhna, S., T. Stadler, F. Ronquist, and T. Britton. 2011. Inferring speciation and extinction rates under different species sampling schemes. Molecular Biology and Evolution 28:2577–2589.

Huelsenbeck, J. P. and B. Rannala. 2003. Detecting correlation between characters in a comparative analysis with uncertain phylogeny. Evolution 57:1237–1247.