Statistical Phylogenetic Inference using RevBayes

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

December 16, 2014

Contents

Chapter 1

Basic 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).

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.

1.1.1 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 frameworkallows 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).

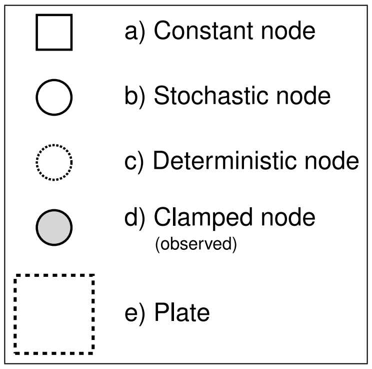


Figure 1.1: The symbols for a visual representation of a graphical model. a) Solid squares represent constant nodes, which specify fixed-valued variables. b) Stochastic nodes are represented by solid circles. These variables correspond to random variables and may depend on other variables. c) Deterministic 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:

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

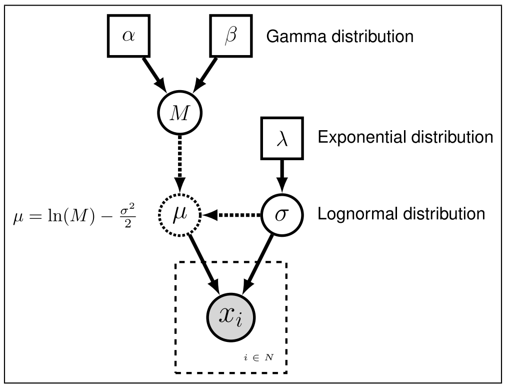


Figure 1.2: Graphical model representation of a simple lognormal model. A total of N observations of variable x are observed and occupy a clamped node. This parameter is log-normally distributed with parameters μ and σ (log mean and standard deviation, respectively). The parameter μ is a deterministic 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.

`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.

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
`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)
```

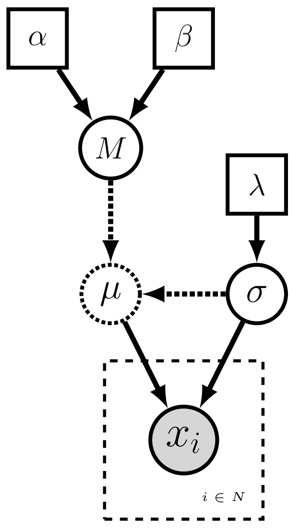
The flexibility gained from the graphical model framework and the interpreted language allows you to easily change a model by swapping components. For example, if you decide that a bimodal lognormal distribution is a better representation of your uncertainty in **`M`**, then you can simply change the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

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)
}
```

Getting help in `RevBayes`

`RevBayes` provides an elaborate help system. Most of the help is found online on our website http://www.RevBayes.
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:

```
RevBayes > ?dnNorm
RevBayes > ?mcmc
RevBayes > ?mcmc.run
```

Additionally, `RevBayes` will print the correct usage of a function if it is executed without any arguments:

```
RevBayes > mcmc()
   Error:   Argument mismatch for call to function 'mcmc'( ). Correct usage is:
   MCMC function (Model model, Monitor[] monitors, Move[] moves, String
      moveschedule = "sequential" | "random" | "single"
```

Continue on to the next page to start the exercise...

1.2 Exercise: Basic Rev Commands

1.2.1 Introduction

The first section of this exercise involves

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

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:

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

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

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

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

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

Here you can see that comments always start with the 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:

```
RevBayes > # Math-Functions
RevBayes > exp(1)                                    # exponential function
RevBayes > ln(1)                                     # logarithmic function
   with natural base
RevBayes > sqrt(16)                                  # square root function
RevBayes > 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

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

Moreover, we provide functions for the common statistical distributions.

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

```
RevBayes > 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.

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

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

```
RevBayes > 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.

```
RevBayes > dnorm
```

1.2.2 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.

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

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

```
RevBayes > 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.

```
RevBayes > 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.

```
    RevBayes > 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 (**`:=`**).

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

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

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

```
    RevBayes > d <- ln(b)                       # assignment of constant
        node 'd' with the value if the logarithmic function with
       parameter 'b'
```

```
RevBayes > d                                # printing the value of
   'd'
RevBayes > 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 (**==**).

```
RevBayes > e := (c == d)
RevBayes > 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.

```
RevBayes > a <- 2                           # reassignment of
   variable a; every deterministic node which has 'a' as a parameter
    changes its value
RevBayes > a                                # printing the value of
   'a'
RevBayes > b                                # printing the value of
   'b'
RevBayes > c                                # printing the value of
   'c'
RevBayes > d                                # printing the value of
   'd'
RevBayes > 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.

```
RevBayes > # Variable assignment: stochastic
RevBayes > lambda <- 1                      # assign constant node '
   lambda' with value '1'
RevBayes > 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

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

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

```
RevBayes > mu <- 0
RevBayes > sigma <- 1
RevBayes > y ~ dnNorm(mu,sigma)
RevBayes > y.probability()                          # print the probability
   of 'y'
RevBayes > y.lnProbability()                        # print the log-
   probability if 'y'
RevBayes > 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.

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

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

```
    RevBayes > index += 10                          # add 10 to the current
       value
    RevBayes > 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.

```
    RevBayes > 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.

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

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

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

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

```
    RevBayes > 1:10                                 # range function
    RevBayes > rep(10,1)                            # replicate an element n
        times
    RevBayes > 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

```
RevBayes > 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

```
RevBayes > 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:

```
RevBayes > 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

```
RevBayes > # loops
RevBayes > for (<variable> in <set of value>) <single statement>
RevBayes >
RevBayes > for (<variable> in <set of value>)
RevBayes > <single statement>
RevBayes >
RevBayes > for (<variable> in <set of value>) {
RevBayes > <multiple statements>
RevBayes > <multiple statements>
RevBayes > <multiple statements>
RevBayes > }
```

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

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

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

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

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

User Defined Functions

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

```
RevBayes > 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`**

```
RevBayes > # simple square function
RevBayes > 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`.

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

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

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

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

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

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

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

1.3 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.

Model and Data

We will use the data example from **?** (Table 1.2). A summary is given in table 1.2.

Table 1.2: 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]))
```

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 $P(\alpha, \beta|y)$. 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 $\sigma = \delta_\alpha$:

$$\alpha' \sim \text{norm}(alpha[i-1], delta\_alpha) \tag{1.1}$$

```
> alpha_prime <- rdnNorm(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 parameters value.

```
>    alpha_prime <- rdnNorm(1,alpha[i-1],delta_alpha)[1]
>    beta_prime <- rdnNorm(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 op 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.

1.4 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.

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 $t = 1851, \dots, 1962$ 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

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

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)
```

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`

Useful Links

- RevBayes: https://github.com/revbayes/code

Questions about this tutorial can be directed to:

- Sebastian Höhna (email: sebastian.hoehna@gmail.com)
- Tracy Heath (email: tracyh@berkeley.edu)
- Michael Landis (email: mlandis@berkeley.edu)

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

Continuous Time Markov Model for Discrete Character Evolution

Overview

This tutorial demonstrates how to set up and perform an analysis for different substitution models. 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_Basics_Tutorial

2.1 Exercise: Character Evolution under various Substitution Models

2.1.1 Getting Started

For the exercises outlined in this tutorial, we will use `RevBayes` interactively by typing commands in the command-line console. The format of this exercise uses `lavender blush shaded boxes` to delineate important steps. The various `RevBayes` commands and syntax are specified using **`typewriter text`**. And the specific commands that you should type (or copy/paste) into `RevBayes` are indicated by shaded box and prompt. For example, after opening the `RevBayes` program, you can load your data file:

```
RevBayes > data_ITS <- readDiscreteCharacterData("data/fagus_ITS.nex")
```

For this command, type in the command and its options:
`data_ITS <- readDiscreteCharacterData("data/fagus_ITS.nex")`. **DO NOT** type in "**`RevBayes >`**", the prompt is simply included to replicate what you see on your screen.

Multi-line entries, particularly loops, will often be displayed in boxes without the **`RevBayes >`** prompt so that they can be copied and pasted wholly.

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

This tutorial 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 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 data and output files from: https://molevol.mbl.edu/index.php/RevBayes

- Open the file **`data/fagus_ITS.nex`** in your text editor. This file contains the sequences for the ITS gene sampled from 13 species (Box 1). The elements of the **`DATA`** block indicate the type of data, number of taxa, and length of the sequences.

Box 1: A fragment of the NEXUS file containing the ITS 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;
```

- Also note that “pre-cooked” output files are provided in the download. Throughout this tutorial, you can use those files to summarize output if you do not have time to run the full analyses yourself.

2.1.2 Launch RevBayes

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 the program information, including the current version number and functions that will provide information about the program — **`contributors()`** and **`license()`**.

2.1.3 An Unpartitioned Analysis

Black Box Analysis

The first section of this exercise involves (1) setting up a uniform GTR+Γ model for an alignment comprising two genes, (2) approximating the posterior probability of the tree topology and branch lengths (and all other parameters) using MCMC, (3) summarizing the MCMC output by computing the maximum a posteriori tree, and (4) estimating the marginal likelihood of the model using stepping-stone and path sampling.

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:

```
RevBayes > source("RevBayes_scripts/quick_uniform.Rev")
```

If everything loaded properly, then you should see the program begin running the power posterior analysis needed for estimating the marginal likelihood. If you continue to let this run, then you will see it output the states of the Markov chain once the MCMC analysis begins. (It is worth noting, however, that the file **`quick_uniform.Rev`** performs shorter runs with fewer generations for a faster run time.)

Ultimately, this is how you will execute most analyses in RevBayes and the full specification of the model and analyses are contained in the sourced files. You could easily run this entire analysis on your own data if you changed the name of the files containing the tutorial's sequences in the model specification file: **`RB_tutorial_files/uniform_partition_model.Rev`**. However, it is important to understand the components of the model to be able to take advantage of the flexibility and richness of RevBayes. Furthermore, without inspecting the Rev scripts sourced in **`quick_uniform.Rev`**, you may have inadvertently conducted an inappropriate analysis 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.

Full Model Specification

Load Data

First load in the sequences using the **`readDiscreteCharacterData()`** function. This function returns a *vector* of data matrices and, even though there is only one element in the vector, we must index that element using the **`[1]`** notation. (You will also note that list indexing in Rev starts with **`1`** like in the R language.)

```
RevBayes > data_atpB <- readDiscreteCharacterData("data/conifer_atpB.nex
   ")[1]
RevBayes > data_rbcL <- readDiscreteCharacterData("data/conifer_rbcL.nex
   ")[1]
```

Executing these lines initializes each data matrix as their respective Rev variables. Since the first step in this exercise is to assume a single model for both genes, we need to combine the two datasets. Concatenate the two data matrices using the **`+`** operator. This returns a single data matrix with both genes.

```
RevBayes > data <- data_atpB + data_rbcL
```

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:

```
RevBayes > data
   Origination:                    conifer_atpB.nex
   Number of taxa:                9
   Number of characters:          2659
   Number of included characters: 2659
   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`**), the tip labels (**`names`**), and the number of internal branches (**`n_branches`**). Each of these variables will be necessary for setting up different parts of our model.

```
RevBayes > n_species <- data.ntaxa()
RevBayes > names <- data.names()
RevBayes > n_branches <- 2 * n_species - 3
```

Now we can proceed with building our GTR+Γ model.

The GTR Parameters

The first step in this exercise involves estimating the marginal likelihood of our model assuming an unpartitioned alignment. This corresponds to the assumption that the process that gave rise to our data was homogeneous across all sites. Specifically, we will assume that both genes evolved under the same GTR+Γ model (Fig. 4.1).

First, we will define and specify a prior on the exchangeability rates of the GTR model. We will use a flat Dirichlet prior distribution on these six rates. To do this, we must begin by defining a constant node that specifies the vector of concentration values of the Dirichlet prior using the **`v()`** function:

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

The constant node **`er_prior`** corresponds to the node labeled e in the graphical model depicted in Figure **??**. The vector function, **`v()`**, creates a vector of six values. Display the current value of **`er_prior`** by simply typing the variable name:

```
RevBayes > er_prior
       [ 1, 1, 1, 1, 1, 1 ]
```

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

```
RevBayes > er ~ dnDirichlet(er_prior)
```

The Dirichlet distribution assigns probability densities to grouped parameters: *e.g.*, those that measure proportions and must sum to 1. Above, we specified a 6-parameter Dirichlet prior on the relative rates of the GTR model, where the placement of each value specified represents one of the 6 relative rates: (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 parameters or concentration parameters and a value is specified for each of the 6 GTR rates. The expectation and variance for each variable are related to the sum of the shape parameters. The prior above is a 'flat' or symmetric Dirichlet since all of the shape parameters are equal (1,1,1,1,1,1), thus we are specifying a model that allows for equal rates of change between nucleotides, such that the expected rate for each is equal to $\frac{1}{6}$ (Zwickl and Holder 2004). Figure 4.2a shows the probability density of each rate under this model. If we parameterized the Dirichlet distribution such that all of the parameters were equal to 100, this would also specify a prior with an expectation of equal exchangeability rates (Figure 4.2b). However, by increasing the shape parameters of the Dirichlet distribution, `er_prior <- v(100,100,100,100,100,100)`, would heavily restrict the MCMC from sampling sets of GTR rates in which the values were not equal or very nearly equal (*i.e.*, this is a very *informative* prior). We can consider a different Dirichlet parameterization if we had strong prior belief that transitions and transversions occurred at different rates. In this case, we could specify a more informative 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 the expected rate for transitions would equal $\frac{8}{32}$, and there would be greater prior probability on sets of GTR rates that matched this configuration (Figure 4.2c). An alternative informative prior would be one where we assumed that each of the 6 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 4.2d). Without strong prior knowledge about the pattern of relative rates, however, we can better capture our statistical uncertainty with 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 sample that value. The Dirichlet prior on our parameter `er` creates a *simplex* of values that sum to 1. In RevBayes, there are many different proposal mechanisms – called *moves* – and each move operates on a specific data type (called RevType). Check the RevType of the variable `er` using the `structure()` function:

```
RevBayes > structure(er)

   _variable     = er <0x7ffed8449370>
   _RevType      = Simplex
   _RevTypeSpec  = [ Simplex, RealPos[], ModelContainer, Container, RevObject ]
   _value        = [ 0.00308506, 0.491487, 0.186317, 0.0275106, 0.1982...
   _size         = 6
   _dagNode      = er <0x7ffed8448bb0>
   _dagType      = Stochastic DAG node
   _refCount     = 1
   _distribution = <0x7ffed8448cf0>
   _touched      = TRUE
   _clamped      = FALSE
   _lnProb       = -inf
   _storedLnProb = 6.95325e-310
   _parents      = [ er_prior <0x7ffed8449250> ]
```

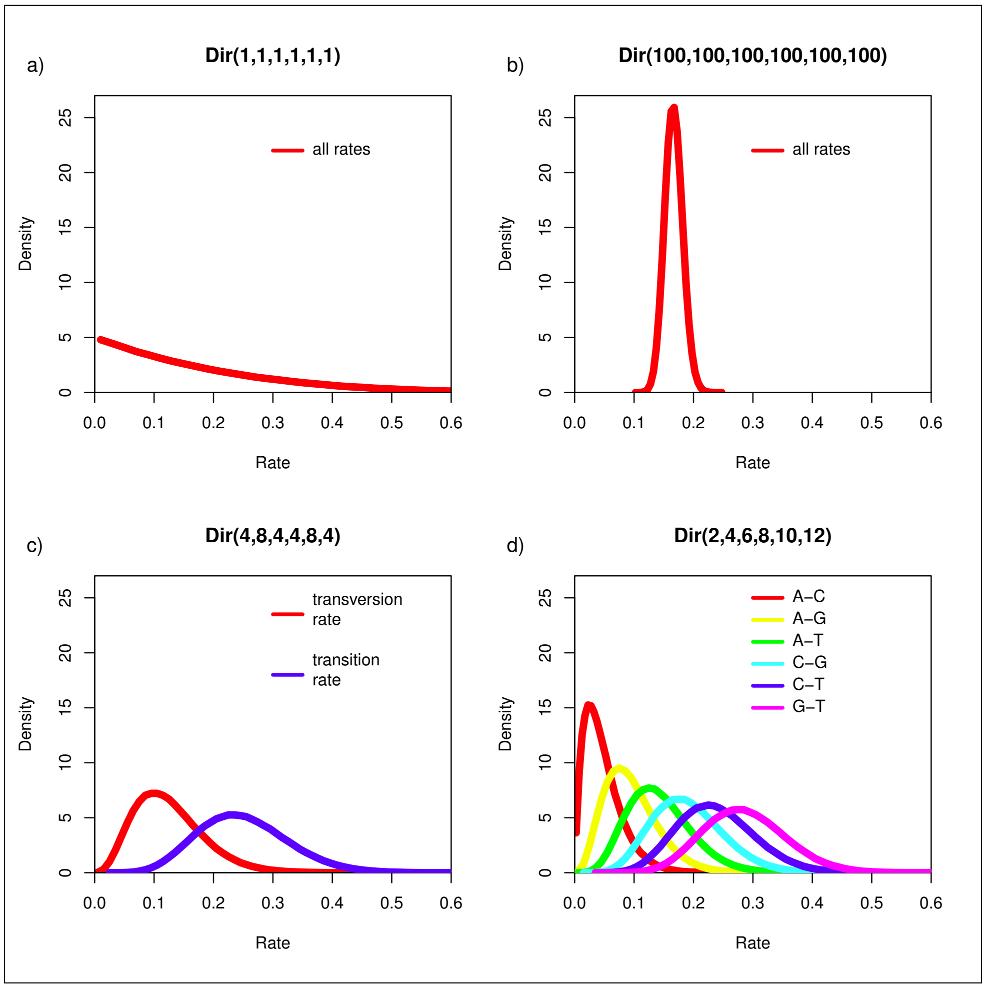


Figure 2.1: Four different examples of Dirichlet priors on exchangeability rates.

```
 _children      = [  ]
```

The **structure()** function – which has an accepted abbreviation of **str()** – is verbose and provides a lot of information that may at first appear confusing. In particular, this function provides the *memory addresses* of the node, its parent node(s), and the distribution. These strings indicate the location of the variable in computer memory. Thus, if you view the structure of your **er** node, these sequences of numbers will be different from the ones in the box above. Much of this information is helpful primarily for troubleshooting and debugging purposes, however, the components that you may want to look at are: **RevType**, **value**, **dagType**, **clamped**, **lnProb**, and the names of the **variable**, **parents**, and **children**.

We must create a vector containing all of the moves for each of our stochastic nodes. This vector will be passed in to the function constructing our MCMC or power posterior runs. All moves in the Rev language are called **mv***, where ***** is a wild card for the move name. Initialize the first element of our vector of moves by setting the proposal on the exchangeability rates:

```
RevBayes > moves[1] <- mvSimplexElementScale(er, alpha=10, tune=true, weight=3)
```

The various proposal mechanisms available in RevBayes each require specific input arguments. The **mvSimplexElementScale** move can only operate on a simplex and the first argument is the stochastic node that you wish to update. The variable node is followed by three more arguments: (1) **alpha** is the *tuning parameter* of the move and controls the size of the proposal; (2) by setting **tune=true** we are telling the program to adjust the tuning parameter if the acceptance rate of the proposal is too high or too low – the target for this move is an acceptance rate of 0.44; (3) the **weight** specifies how frequently this move is performed at each step in the Markov chain.

It is important to note that by default, a single generation in RevBayes updates all stochastic nodes in proportion to the **weight** argument specified. This approach is different from many phylogenetic MCMC programs – such as MrBayes – which only perform one move per MCMC iteration. Instead, for each generation in RevBayes, a *move list* is carried out in random order. By setting **weight=3** above, we are ensuring that the new values are proposed for **er** three times per iteration. Thus, if you set **weight=0.5** the move will only have a 50% chance of occurring at each generation. Essentially, the **weight** argument for each move indicates the number of times it will be performed. If you keep all of the move weights set to the values specified in this tutorial for the uniform model, you will have set 20 different moves and the MCMC simulator will execute 38 moves per generation. This is a practical approach for MCMC analysis using complex models and is used by several other programs (Phylobayes, Phycas, Bali-Phy). However, because RevBayes is updating many parameters each generation, it is not straightforward to compare run-times with a program like MrBayes that only performs approximately one update per generation. Furthermore, using this approach to MCMC simulation, you can sufficiently sample the chain in fewer generations while sampling more frequently than you would in programs that only perform one update per step.

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:

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

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

```
RevBayes > moves[2] <- mvSimplexElementScale(sf, 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.

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

Gamma-Distributed Site Rates

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 (Yang 1994). 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 4.3). 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 4.3).

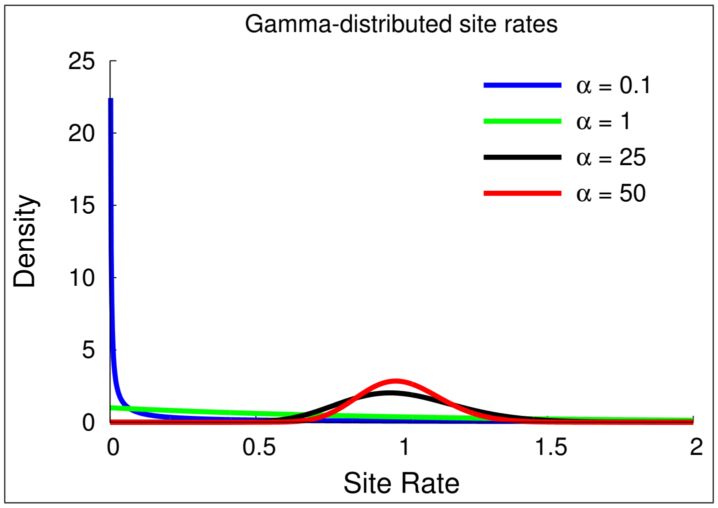


Figure 2.2: The probability density of mean-one gamma-distributed rates under different shape parameters.

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

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

```
RevBayes > shape_prior <- 0.05
```

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

```
RevBayes > shape ~ dnExponential(shape_prior)
```

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

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

```
RevBayes > gamma_rates := fnDiscretizeGamma( shape, shape, 4 )
```

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

```
moves[3] <- mvScale(shape, lambda=1.0, tune=true, weight=2.0)
```

Tree Topology and Branch Lengths

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

We will assume that all possible labeled, unrooted tree topologies have equal probability. This is the **`dnUniformTopology()`** distribution in RevBayes. Specify the **`topology`** stochastic node by passing in the number of species **`n_species`** and tip labels **`names`** to the **`dnUniformTopology()`** distribution:

```
RevBayes > topology ~ dnUniformTopology(n_species, names)
```

For some types of stochastic nodes there are several available moves. Often the different moves explore parameter space in a different way and nothing prevents one from using multiple different moves to improve mixing. For the unrooted tree topology, we can use both a nearest-neighbor interchange move (**`mvNNI`**) and a subtree-prune and regrafting move (**`mvSPR`**). These moves do not have tuning parameters associated with them, thus you only need to pass in the **`topology`** node and **`weight`**

```
RevBayes > moves[4] <- mvNNI(topology, weight=10.0)
RevBayes > moves[5] <- mvSPR(topology, weight=5.0)
```

Next we have to create a stochastic node for each of the $2N-3$ branches in our tree (where $N =$ **n_species**). We can do this using a **for** loop — this is a plate in our graphical model. In this loop, we can create each branch-length node and assign each move. Copy this entire block of Rev code into the console:

```
mi <- 5
for (i in 1:n_branches) {
   br_lens[i] ~ dnExponential(10.0)
   moves[mi++] <- mvScale(br_lens[i],lambda=1,tune=true,weight=1)
}
```

It is convenient to monitor a deterministic variable of the branch lengths. In MrBayes, *tree length* was reported to the log file instead of the length of each branch. The tree length is the sum of all branch lengths and this can be computed using the **sum()** function which calculates the sum of any vector of values.

```
RevBayes > tree_length := sum(br_lens)
```

Finally, we can create a branch-length phylogeny by combining the tree topology and branch lengths using the **treeAssembly()** function, which applies the value of the i^{th} member of the **br_lens** vector to the branch leading to the i^{th} node in **topology**. Thus, the **phylogeny** variable is a deterministic node:

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

Putting it All Together

Now that we have initialized virtually all of our model parameters and we can link all of the parts in the stochastic node that will be clamped by the data. The sequence substitution model is a distribution called the *phylogenetic continuous-time Markov chain* and we use the **dnPhyloCTMC** constructor function to create this node. This distribution requires several input arguments: (1) the **tree** with branch lengths, (2) the instantaneous rate matrix **Q**, the node characterizing the rate variation across sites (though **siteRates** can be omitted if you do not assume rate variation across sites), (3) **nSites** is the number of sites in the alignment, and (4) the **type** of character data.

```
RevBayes > phyloSeq ~ dnPhyloCTMC(tree=phylogeny, Q=Q, siteRates=
   gamma_rates, nSites=data.nchar(1), type="DNA")
```

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

```
RevBayes > phyloSeq.clamp(data)
```

When this function is called, RevBayes sets each of the stochastic nodes representing the tip nodes of the tree to the sequence corresponding to that species in the alignment. This essentially tells the program that this is where the DAG ends and the states of the tip nodes are fixed.

Now we can wrap up the whole model to conveniently access 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:

```
RevBayes > mymodel <- model(sf)
```

Now we have specified a simple, single-partition 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:

```
RevBayes > mymodel
```

Perform MCMC Analysis Under the Uniform Model

This section will cover setting up the MCMC sampler and summarizing the posterior distribution of trees.

Specify Monitors

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.

```
RevBayes > monitors[1] <- mnModel(filename="output/conifer_uniform.log",
    printgen=100)
```

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.

```
RevBayes > monitors[2] <- mnFile(filename="output/conifer_uniform.trees
   ",printgen=100, phylogeny)
```

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

```
RevBayes > monitors[3] <- mnScreen(printgen=10, separator = " | ",
   tree_length)
```

Initialize and Run MCMC

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:

```
RevBayes > 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.

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

Now, run the MCMC:

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

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

Summarize the MCMC Output

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 2009) 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:

```
RevBayes > treetrace <- readTreeTrace("output/conifer_uniform.trees")
RevBayes > treetrace.summarize()
```

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

```
RevBayes > mapTree(treetrace,"output/conifer_uniform_map.tre")
```

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 full_analysis.Rev`**

Useful Links

- RevBayes: https://github.com/revbayes/code
- MrBayes: http://mrbayes.sourceforge.net
- PhyloBayes: http://www.phylobayes.org
- Bali-Phy: http://www.bali-phy.org
- Tree Thinkers: http://treethinkers.org

Questions about this tutorial can be directed to:

- Tracy Heath (email: tracyh@berkeley.edu)
- Michael Landis (email: mlandis@berkeley.edu)
- Sebastian Höhna (email: sebastian.hoehna@gmail.com)
- Brian R. Moore (email: brianmoore@ucdavis.edu)

Bibliography

Rambaut A, Drummond AJ. 2009. Tracer v1.5. Edinburgh (United Kingdom): Institute of Evolutionary Biology, University of Edinburgh. Available from: http://beast.bio.ed.ac.uk/Tracer.

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

Zwickl DJ, Holder MT. 2004. Model parameterization, prior distributions, and the general time-reversible model in Bayesian phylogenetics. Systematic Biology. 53:877–888.

Chapter 3

Markov chain Monte Carlo Algorithms

Overview

This tutorial demonstrates how to set up and perform an analysis for different substitution models. 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_Basics_Tutorial

3.1 Exercise: Character Evolution under various Substitution Models

3.1.1 Getting Started

Chapter 4

Model Selection and Bayes Factors

Overview

RevBayes has as a central idea that phylogenetic models, like any statistical model, are 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. RevBayes implements 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, transformations, models, and constants of different sorts).

This tutorial demonstrates how to set up and perform an analysis that calculates Bayes factors to select among different partition configurations of aligned DNA sequences. After selecting the model that is best supported by the data, the exercise continues with basic inference of an unrooted tree topology and branch lengths using Markov chain Monte Carlo (MCMC).

4.1 Exercise: Model Selection & Partitioning using Bayes Factors

4.1.1 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). To accommodate process heterogeneity within and/or between various gene(omic) regions, we will evaluate the support for various partition schemes using Bayes factors to compare the marginal likelihoods of the candidate partition schemes.

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.

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

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

For most sequence alignments, several (possibly many) partition schemes of varying complexity are plausible *a priori*, which therefore requires a way to objectively identify the partition scheme that balances estimation bias and error variance associated with under- and over-parameterized mixed models, respectively. Increasingly, mixed-model selection is based on *Bayes factors* (*e.g.*, Suchard, Weiss and Sinsheimer 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, Schmitz and Reeder 2005; Nylander et al. 2004; McGuire et al. 2007). The analysis pipeline that we will use in this tutorial is depicted in Figure 4.1.

Given two models, M_0 and M_1, the Bayes factor comparison assessing the relative plausibility of each

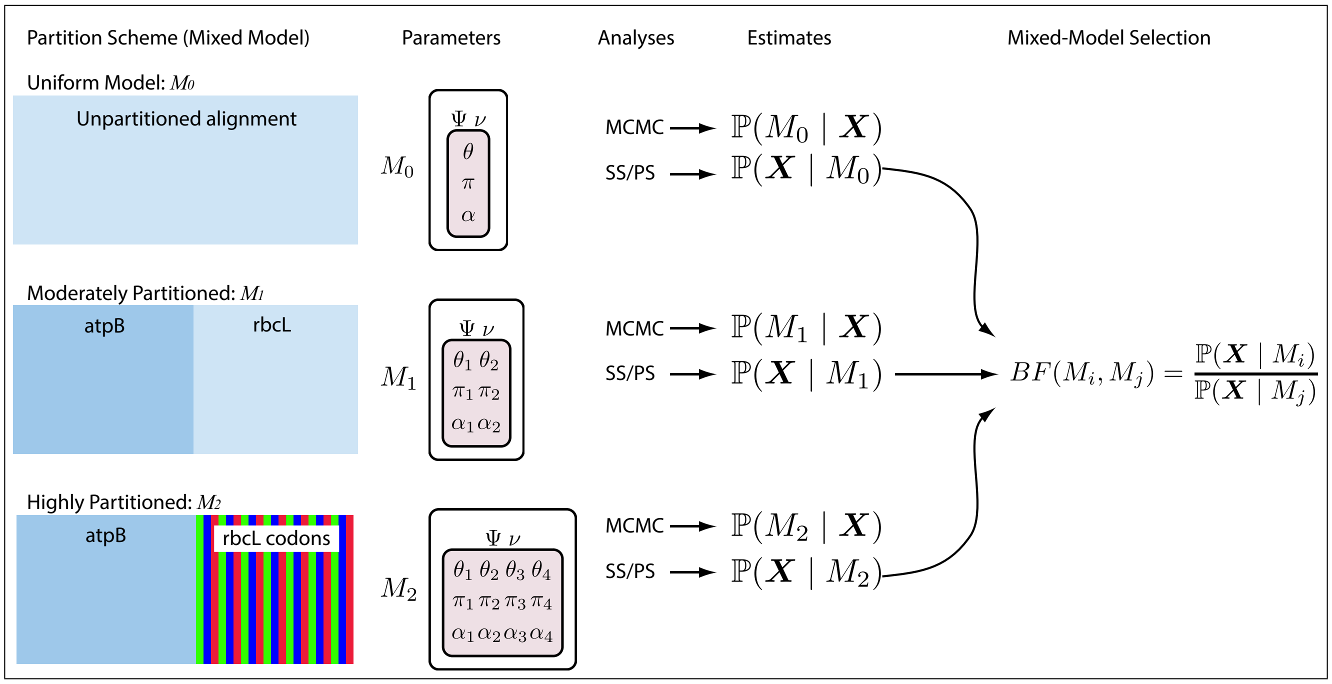

Partition Scheme (Mixed Model)
Parameters
Analyses
Estimates
Mixed-Model Selection
Uniform Model: M_0
Unpartitioned alignment
M_0
$\Psi\ \nu$
θ π α
MCMC
SS/PS
$\mathbb{P}(M_0 \mid \boldsymbol{X})$
$\mathbb{P}(\boldsymbol{X} \mid M_0)$
Moderately Partitioned: M_1
atpB
rbcL
M_1
$\theta_1\ \theta_2$ $\pi_1\ \pi_2$ $\alpha_1\ \alpha_2$
$\mathbb{P}(M_1 \mid \boldsymbol{X})$
$\mathbb{P}(\boldsymbol{X} \mid M_1)$
$BF(M_i, M_j) = \frac{\mathbb{P}(\boldsymbol{X} \mid M_i)}{\mathbb{P}(\boldsymbol{X} \mid M_j)}$
Highly Partitioned: M_2
atpB
rbcL codons
M_2
$\theta_1\ \theta_2\ \theta_3\ \theta_4$ $\pi_1\ \pi_2\ \pi_3\ \pi_4$ $\alpha_1\ \alpha_2\ \alpha_3\ \alpha_4$
$\mathbb{P}(M_2 \mid \boldsymbol{X})$
$\mathbb{P}(\boldsymbol{X} \mid M_2)$

Figure 4.1: The analysis pipeline for Exercise 1. We will explore three partition schemes for the conifer 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 (atpB and rbcL), 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 partition corresponds to the atpB gene region, and the remaining partitions correspond to the three codon positons of the rbcL gene region—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.

model as an explanation of 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):

$$BF(M_0, M_1) = \frac{\mathbb{P}(M_0 \mid \mathbf{X}, \theta_0)}{\mathbb{P}(M_1 \mid \mathbf{X}, \theta_1)} \div \frac{\mathbb{P}(M_0)}{\mathbb{P}(M_1)}. \tag{4.1}$$

This, somewhat vague, definition does not lead to clear-cut identification of the "best" model. Instead, *you* must decide the degree of your belief in M_0 relative to M_1. Despite the absence of any strict "rule-of-thumb", you can refer to the scale (outlined by Jeffreys 1961) for interpreting these measures (Table 4.1).

Table 4.1: The scale for interpreting Bayes factors by Harold Jeffreys (1961).

$BF(M_0, M_1)$	Strength of evidence
$< 1:1$	Negative (supports M_1)
$1:1$ to $3:1$	Barely worth mentioning
$3:1$ to $10:1$	Substantial
$10:1$ to $30:1$	Strong
$30:1$ to $100:1$	Very strong
$> 100:1$	Decisive

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

Unfortunately, direct calculation of the posterior odds to prior odds ratio is unfeasible for most phylogenetic models. 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 marginalized over all parameters for M_i; it is also referred to as the *model evidence* or *integrated likelihood*. More explicitly, the marginal likelihood is the probability of the set of observed data ($\mathbf{X}$) under a given model (M_i), while averaging over all possible values of the parameters of the model (θ_i) with respect to the prior density on θ_i

$$\mathbb{P}(\mathbf{X} \mid M_i) = \int \mathbb{P}(\mathbf{X} \mid \theta_i)\mathbb{P}(\theta_i)dt. \tag{4.2}$$

If you refer back to equation 4.1, you can see that, with very little algebra, the ratio of marginal likelihoods is equal to the Bayes factor:

$$BF(M_0, M_1) = \frac{\mathbb{P}(\mathbf{X} \mid M_0)}{\mathbb{P}(\mathbf{X} \mid M_1)} = \frac{\mathbb{P}(M_0 \mid \mathbf{X}, \theta_0)}{\mathbb{P}(M_1 \mid \mathbf{X}, \theta_1)} \div \frac{\mathbb{P}(M_0)}{\mathbb{P}(M_1)}. \tag{4.3}$$

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 4.2), 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) and path sampling estimators (Gelman and Meng 1998; Lartillot and Philippe 2006; Friel and Pettitt 2008).

4.1.2 Getting Started

This tutorial assumes that you have already downloaded, compiled, and installed RevBayes. We also recommend that—if you are working on a Unix machine—you put the `rb` binary in your path.

For the exercises outlined in this tutorial, we will use RevBayes interactively by typing commands in the command-line console. The format of this exercise uses `lavender blush shaded boxes` to delineate important steps. The various RevBayes commands and syntax are specified using **`typewriter text`**. And the specific commands that you should type (or copy/paste) into RevBayes are indicated by shaded box and prompt. For example, after opening the RevBayes program, you can load your data file:

```
RevBayes > data_atpB <- readDiscreteCharacterData("data/conifer_atpB.nex
   ")[1]
```

For this command, type in the command and its options:
`data_atpB <- readDiscreteCharacterData("data/conifer_atpB.nex")[1]`. **DO NOT** type in "**`RevBayes >`**", the prompt is simply included to replicate what you see on your screen.

Multi-line entries, particularly loops, will often be displayed in boxes without the **`RevBayes >`** prompt so that they can be copied and pasted wholly.

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

This tutorial 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 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 data and output files from: https://molevol.mbl.edu/index.php/RevBayes

- Open the file **`data/conifer_atpB.nex`** in your text editor. This file contains the sequences for the atpB gene sampled from 9 species (Box 1). The elements of the **`DATA`** block indicate the type of data, number of taxa, and length of the sequences.

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

```
#NEXUS

Begin data;
        Dimensions ntax=9 nchar=1394;
        Format datatype=dna gap=-;
        Matrix
Ginkgo_biloba           TTATTGGTCCAGTACTGGATGTAGCTTTTCCCCCGGGCAATATGCCTAATATTTACAATTCTTTG...
Araucaria_araucana      -----GGTCCGGTACTGGATGTATCTTTTCCTCCAGATGAAATGCCCTATATTTACAATTCTTTG...
Cedrus_deodara          TCATTGGCCCAGTACTGGA?GTCTCTTTTCCTCCAGGTAATATGCCTAATATTTACAATTCATTG...
Cupressus_arizonica     -----------------GATGTATCTTTCCCTCCAGGTAGTATGCCTAGAATTTACAATTCTTTG...
Juniperus_communis      -----------------------------------------------------------------...
Pinus_densiflora        TCATTGGCCCAGTACTGGATGTCTCTTTTCCTCCAGGTAATATGCCTAATATTTACAATTCATTG...
Podocarpus_chinensis    TCATCGGCCCTGTACTGGATGTATCTTTTCCTCCAGATGGTATGCCTTTTATTTACAATTCTTTA...
```

```
Sciadopitys_verticillata TCATTGGTCCAGTACTAGATGTATCTTTCCCTCCAGGCAATATGCCTAGAATTTACAATTCTTTG...
Taxus_baccata            TTATCGGCCCAGTACTAGATGTCTCTTTTCCTCCAGGTAATATGCCTAAAATTTACAATTCCTTA...
        ;
End;
```

- Also note that "pre-cooked" output files are provided in the download. Throughout this tutorial, you can use those files to summarize output if you do not have time to run the full analyses yourself.

4.1.3 Launch RevBayes

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 the program information, including the current version number and functions that will provide information about the program — **`contributors()`** and **`license()`**.

Currently, the help-system of RevBayes is virtually nonexistent. This will not always be the case, but is par for the course when using new/experimental software. The most complete help file available is for the **`mcmc()`** function. Display the help for this function using the **`?`** symbol:

```
RevBayes > ?mcmc
```

Additionally, RevBayes will print the correct usage of a function if it is executed without any arguments:

```
RevBayes > mcmc()
   Error:   Argument mismatch for call to function 'mcmc'( ). Correct usage is:
   MCMC function (Model model, VectorRbPointer<Monitor> monitors,
   VectorRbPointer<Move> moves, String moveschedule = sequential|random|single)
```

4.1.4 An Unpartitioned Analysis

Black Box Analysis

The first section of this exercise involves (1) setting up a uniform GTR+Γ model for an alignment comprising two genes, (2) approximating the posterior probability of the tree topology and branch lengths (and all other parameters) using MCMC, (3) summarizing the MCMC output by computing the maximum a posteriori tree, and (4) estimating the marginal likelihood of the model using stepping-stone and path sampling.

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:

```
RevBayes > source("RevBayes_scripts/quick_uniform.Rev")
```

If everything loaded properly, then you should see the program begin running the power posterior analysis needed for estimating the marginal likelihood. If you continue to let this run, then you will see it output the states of the Markov chain once the MCMC analysis begins. (It is worth noting, however, that the file **`quick_uniform.Rev`** performs shorter runs with fewer generations for a faster run time.)

Ultimately, this is how you will execute most analyses in RevBayes and the full specification of the model and analyses are contained in the sourced files. You could easily run this entire analysis on your own data if you changed the name of the files containing the tutorial's sequences in the model specification file: **`RB_tutorial_files/uniform_partition_model.Rev`**. However, it is important to understand the components of the model to be able to take advantage of the flexibility and richness of RevBayes. Furthermore, without inspecting the Rev scripts sourced in **`quick_uniform.Rev`**, you may have inadvertently conducted an inappropriate analysis 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.

Full Model Specification

Load Data

First load in the sequences using the **`readDiscreteCharacterData()`** function. This function returns a *vector* of data matrices and, even though there is only one element in the vector, we must index that element using the **`[1]`** notation. (You will also note that list indexing in Rev starts with **`1`** like in the R language.)

```
RevBayes > data_atpB <- readDiscreteCharacterData("data/conifer_atpB.nex
   ")[1]
RevBayes > data_rbcL <- readDiscreteCharacterData("data/conifer_rbcL.nex
   ")[1]
```

Executing these lines initializes each data matrix as their respective Rev variables. Since the first step in this exercise is to assume a single model for both genes, we need to combine the two datasets. Concatenate the two data matrices using the **`+`** operator. This returns a single data matrix with both genes.

```
RevBayes > data <- data_atpB + data_rbcL
```

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:

```
RevBayes > data
   Origination:                      conifer_atpB.nex
   Number of taxa:               9
   Number of characters:         2659
   Number of included characters: 2659
   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`**), the tip labels (**`names`**), and the number of internal branches (**`n_branches`**). Each of these variables will be necessary for setting up different parts of our model.

```
RevBayes > n_species <- data.ntaxa()
RevBayes > names <- data.names()
RevBayes > n_branches <- 2 * n_species - 3
```

Now we can proceed with building our GTR+Γ model.

The GTR Parameters

The first step in this exercise involves estimating the marginal likelihood of our model assuming an unpartitioned alignment. This corresponds to the assumption that the process that gave rise to our data was homogeneous across all sites. Specifically, we will assume that both genes evolved under the same GTR+Γ model (Fig. 4.1).

First, we will define and specify a prior on the exchangeability rates of the GTR model. We will use a flat Dirichlet prior distribution on these six rates. To do this, we must begin by defining a constant node that specifies the vector of concentration values of the Dirichlet prior using the **`v()`** function:

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

The constant node **`er_prior`** corresponds to the node labeled e in the graphical model depicted in Figure **??**. The vector function, **`v()`**, creates a vector of six values. Display the current value of **`er_prior`** by simply typing the variable name:

```
RevBayes > er_prior
       [ 1, 1, 1, 1, 1, 1 ]
```

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

```
RevBayes > er ~ dnDirichlet(er_prior)
```

The Dirichlet distribution assigns probability densities to grouped parameters: *e.g.*, those that measure proportions and must sum to 1. Above, we specified a 6-parameter Dirichlet prior on the relative rates of the GTR model, where the placement of each value specified represents one of the 6 relative rates: (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 parameters or concentration parameters and a value is specified for each of the 6 GTR rates. The expectation and variance for each variable are related to the sum of the shape parameters. The prior above is a 'flat' or symmetric Dirichlet since all of the shape parameters are equal (1,1,1,1,1,1), thus we are specifying a model that allows for equal rates of change between nucleotides, such that the expected rate for each is equal to $\frac{1}{6}$ (Zwickl and Holder 2004). Figure 4.2a shows the probability density of each rate under this model. If we parameterized the Dirichlet distribution such that all of the parameters were equal to 100, this would also specify a prior with an expectation of equal exchangeability rates (Figure 4.2b). However, by increasing the shape parameters of the Dirichlet distribution, **`er_prior <- v(100,100,100,100,100,100)`**, would heavily restrict the MCMC from sampling sets of GTR rates in which the values were not equal or very nearly equal (*i.e.*, this is a very *informative* prior). We can consider a different Dirichlet parameterization if we had strong prior belief that transitions and transversions occurred at different rates. In this case, we could specify a more informative 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 the expected rate for transitions would equal $\frac{8}{32}$, and there would be greater prior probability on sets of GTR rates that matched this configuration (Figure 4.2c). An alternative informative prior would be one where we assumed that each of the 6 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 4.2d). Without strong prior knowledge about the pattern of relative rates, however, we can better capture our statistical uncertainty with 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 sample that value. The Dirichlet prior on our parameter **`er`** creates a *simplex* of values that sum to 1. In RevBayes, there are many different proposal mechanisms – called *moves* – and each move operates on a specific data type (called RevType). Check the RevType of the variable **`er`** using the **`structure()`** function:

```
RevBayes > structure(er)

  _variable     = er <0x7ffed8449370>
  _RevType      = Simplex
  _RevTypeSpec  = [ Simplex, RealPos[], ModelContainer, Container, RevObject ]
  _value        = [ 0.00308506, 0.491487, 0.186317, 0.0275106, 0.1982...
  _size         = 6
  _dagNode      = er <0x7ffed8448bb0>
  _dagType      = Stochastic DAG node
  _refCount     = 1
  _distribution = <0x7ffed8448cf0>
  _touched      = TRUE
  _clamped      = FALSE
  _lnProb       = -inf
  _storedLnProb = 6.95325e-310
  _parents      = [ er_prior <0x7ffed8449250> ]
```

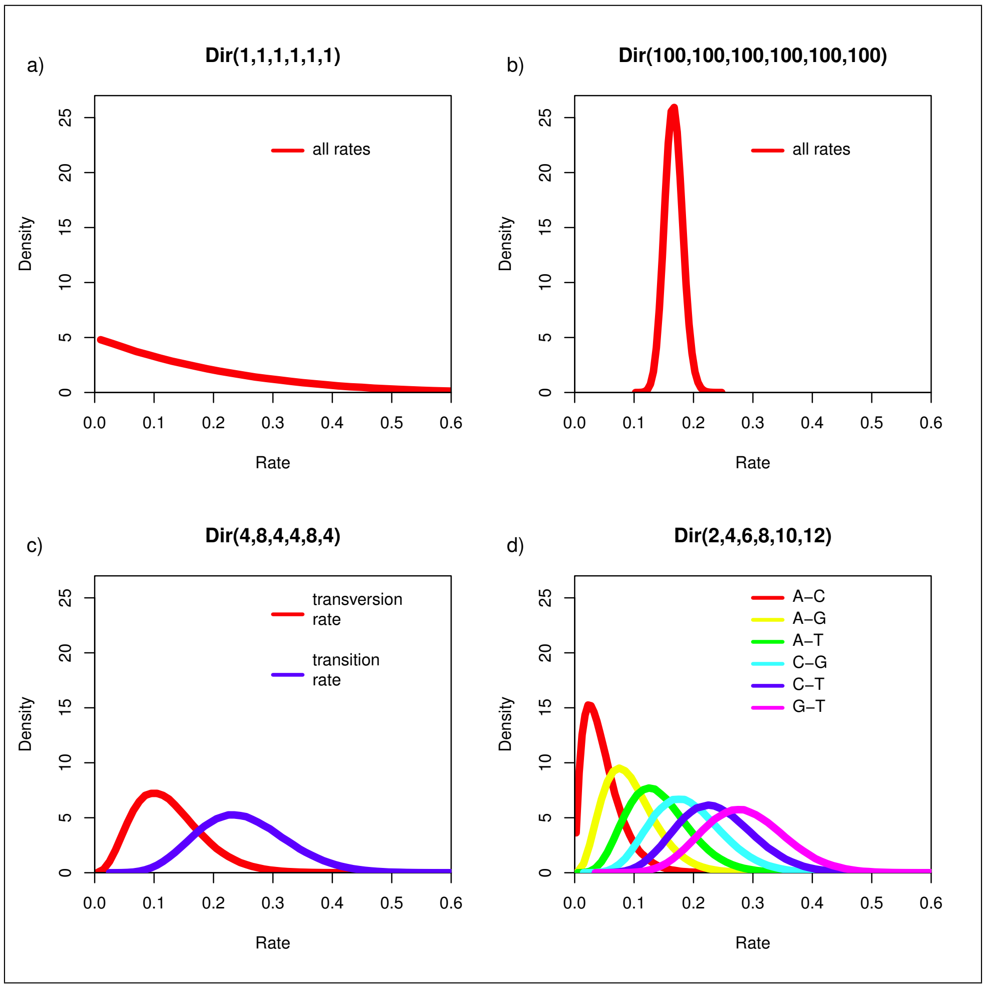


Figure 4.2: Four different examples of Dirichlet priors on exchangeability rates.

```
  _children      = [  ]
```

The **structure()** function – which has an accepted abbreviation of **str()** – is verbose and provides a lot of information that may at first appear confusing. In particular, this function provides the *memory addresses* of the node, its parent node(s), and the distribution. These strings indicate the location of the variable in computer memory. Thus, if you view the structure of your **er** node, these sequences of numbers will be different from the ones in the box above. Much of this information is helpful primarily for troubleshooting and debugging purposes, however, the components that you may want to look at are: **RevType**, **value**, **dagType**, **clamped**, **lnProb**, and the names of the **variable**, **parents**, and **children**.

We must create a vector containing all of the moves for each of our stochastic nodes. This vector will be passed in to the function constructing our MCMC or power posterior runs. All moves in the Rev language are called **mv***, where * is a wild card for the move name. Initialize the first element of our vector of moves by setting the proposal on the exchangeability rates:

```
RevBayes > moves[1] <- mvSimplexElementScale(er, alpha=10, tune=true, weight=3)
```

The various proposal mechanisms available in RevBayes each require specific input arguments. The **mvSimplexElementScale** move can only operate on a simplex and the first argument is the stochastic node that you wish to update. The variable node is followed by three more arguments: (1) **alpha** is the *tuning parameter* of the move and controls the size of the proposal; (2) by setting **tune=true** we are telling the program to adjust the tuning parameter if the acceptance rate of the proposal is too high or too low – the target for this move is an acceptance rate of 0.44; (3) the **weight** specifies how frequently this move is performed at each step in the Markov chain.

It is important to note that by default, a single generation in RevBayes updates all stochastic nodes in proportion to the **weight** argument specified. This approach is different from many phylogenetic MCMC programs – such as MrBayes – which only perform one move per MCMC iteration. Instead, for each generation in RevBayes, a *move list* is carried out in random order. By setting **weight=3** above, we are ensuring that the new values are proposed for **er** three times per iteration. Thus, if you set **weight=0.5** the move will only have a 50% chance of occurring at each generation. Essentially, the **weight** argument for each move indicates the number of times it will be performed. If you keep all of the move weights set to the values specified in this tutorial for the uniform model, you will have set 20 different moves and the MCMC simulator will execute 38 moves per generation. This is a practical approach for MCMC analysis using complex models and is used by several other programs (Phylobayes, Phycas, Bali-Phy). However, because RevBayes is updating many parameters each generation, it is not straightforward to compare run-times with a program like MrBayes that only performs approximately one update per generation. Furthermore, using this approach to MCMC simulation, you can sufficiently sample the chain in fewer generations while sampling more frequently than you would in programs that only perform one update per step.

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:

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

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

```
RevBayes > moves[2] <- mvSimplexElementScale(sf, 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.

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

Gamma-Distributed Site Rates

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 (Yang 1994). 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 4.3). 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 4.3).

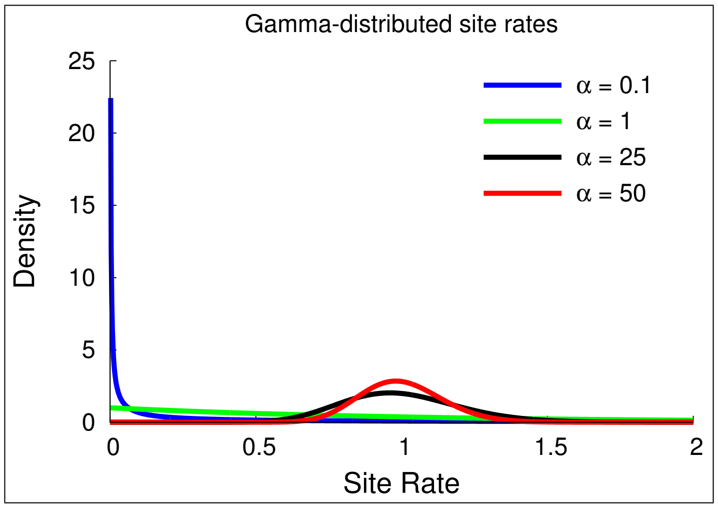


Figure 4.3: The probability density of mean-one gamma-distributed rates under different shape parameters.

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

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

```
RevBayes > shape_prior <- 0.05
```

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

```
RevBayes > shape ~ dnExponential(shape_prior)
```

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

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

```
RevBayes > gamma_rates := fnDiscretizeGamma( shape, shape, 4 )
```

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

```
moves[3] <- mvScale(shape, lambda=1.0, tune=true, weight=2.0)
```

Tree Topology and Branch Lengths

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

We will assume that all possible labeled, unrooted tree topologies have equal probability. This is the **`dnUniformTopology()`** distribution in RevBayes. Specify the **`topology`** stochastic node by passing in the number of species **`n_species`** and tip labels **`names`** to the **`dnUniformTopology()`** distribution:

```
RevBayes > topology ~ dnUniformTopology(n_species, names)
```

For some types of stochastic nodes there are several available moves. Often the different moves explore parameter space in a different way and nothing prevents one from using multiple different moves to improve mixing. For the unrooted tree topology, we can use both a nearest-neighbor interchange move (**`mvNNI`**) and a subtree-prune and regrafting move (**`mvSPR`**). These moves do not have tuning parameters associated with them, thus you only need to pass in the **`topology`** node and **`weight`**

```
RevBayes > moves[4] <- mvNNI(topology, weight=10.0)
RevBayes > moves[5] <- mvSPR(topology, weight=5.0)
```

Next we have to create a stochastic node for each of the $2N-3$ branches in our tree (where $N =$ **`n_species`**). We can do this using a **`for`** loop — this is a plate in our graphical model. In this loop, we can create each branch-length node and assign each move. Copy this entire block of Rev code into the console:

```
mi <- 5
for (i in 1:n_branches) {
   br_lens[i] ~ dnExponential(10.0)
   moves[mi++] <- mvScale(br_lens[i],lambda=1,tune=true,weight=1)
}
```

It is convenient to monitor a deterministic variable of the branch lengths. In MrBayes, *tree length* was reported to the log file instead of the length of each branch. The tree length is the sum of all branch lengths and this can be computed using the **`sum()`** function which calculates the sum of any vector of values.

```
RevBayes > tree_length := sum(br_lens)
```

Finally, we can create a branch-length phylogeny by combining the tree topology and branch lengths using the **`treeAssembly()`** function, which applies the value of the i^{th} member of the **`br_lens`** vector to the branch leading to the i^{th} node in **`topology`**. Thus, the **`phylogeny`** variable is a deterministic node:

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

Putting it All Together

Now that we have initialized virtually all of our model parameters and we can link all of the parts in the stochastic node that will be clamped by the data. The sequence substitution model is a distribution called the *phylogenetic continuous-time Markov chain* and we use the **`dnPhyloCTMC`** constructor function to create this node. This distribution requires several input arguments: (1) the **`tree`** with branch lengths, (2) the instantaneous rate matrix **`Q`**, the node characterizing the rate variation across sites (though **`siteRates`** can be omitted if you do not assume rate variation across sites), (3) **`nSites`** is the number of sites in the alignment, and (4) the **`type`** of character data.

```
RevBayes > phyloSeq ~ dnPhyloCTMC(tree=phylogeny, Q=Q, siteRates=
   gamma_rates, nSites=data.nchar(1), type="DNA")
```

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

```
RevBayes > phyloSeq.clamp(data)
```

When this function is called, RevBayes sets each of the stochastic nodes representing the tip nodes of the tree to the sequence corresponding to that species in the alignment. This essentially tells the program that this is where the DAG ends and the states of the tip nodes are fixed.

Now we can wrap up the whole model to conveniently access 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:

```
RevBayes > mymodel <- model(sf)
```

Now we have specified a simple, single-partition 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:

```
RevBayes > mymodel
```

Perform MCMC Analysis Under the Uniform Model

This section will cover setting up the MCMC sampler and summarizing the posterior distribution of trees.

Specify Monitors

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.

```
RevBayes > monitors[1] <- mnModel(filename="output/conifer_uniform.log",
    printgen=100)
```

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.

```
RevBayes > monitors[2] <- mnFile(filename="output/conifer_uniform.trees
   ",printgen=100, phylogeny)
```

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

```
RevBayes > monitors[3] <- mnScreen(printgen=10, separator = " | ",
   tree_length)
```

Initialize and Run MCMC

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:

```
RevBayes > 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.

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

Now, run the MCMC:

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

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

Summarize the MCMC Output

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 2009) 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:

```
RevBayes > treetrace <- readTreeTrace("output/conifer_uniform.trees")
RevBayes > treetrace.summarize()
```

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

```
RevBayes > mapTree(treetrace,"output/conifer_uniform_map.tre")
```

Estimating the Marginal Likelihood

Typically, model comparison is performed prior to running the full MCMC analysis under a model. If you calculated the Bayes factors to determine the relative support for the uniform model and found that there was strong evidence supporting this model over others (hint: this is not true if you proceed with this tutorial), then it would be worth your time to proceed with the MCMC steps outlined above. The following steps will describe using stepping-stone and path sampling methods on a set of power posteriors to estimate marginal likelihoods under the uniform 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. 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 and re-load the data and model:

```
RevBayes > clear()
RevBayes > source("RevBayes_scripts/uniform_partition_model.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.

```
RevBayes > pow_p <- powerPosterior(mymodel, moves, "pow_p_uniform.out",
   cats=50)
```

We can start the power posterior by first burning in the chain and and discarding the first 10000 states.

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

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

```
RevBayes > 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.

```
RevBayes > ss <- steppingStoneSampler(file="pow_p_uniform.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 6.1.

```
RevBayes > ss.marginal()
```

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

```
RevBayes > ps <- pathSampler(file="pow_p_uniform.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 6.1.

```
RevBayes > ps.marginal()
```

Stop here or continue on to evaluate partitioned models...

4.1.5 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—atpB and rbcL—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.

Clear Workspace and Reload Data

```
RevBayes > clear()
```

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

```
RevBayes > filenames <- v("data/conifer_atpB.nex", "data/conifer_rbcL.
   nex")
```

Set a variable for the number of partitions:

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

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

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

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.

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

Specify the Parameters by Looping Over Partitions

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

```
mi <- 0 # an iterator for the move vector
for (i in 1:n_parts){
  ## index i=1 : atpB gene ##
  ## index i=2 : rbcL gene ##

  # Exchangeability rates #
  er_prior[i] <- v(1,1,1,1,1,1)
  er[i] ~ dnDirichlet(er_prior[i])
  moves[mi++] <- mvSimplexElementScale(er[i], alpha=10, tune=true, weight=3)

  # Stationary base frequencies #
  sf_prior[i] <- v(1,1,1,1)
  sf[i] ~ dnDirichlet(sf_prior[i])
  moves[mi++] <- mvSimplexElementScale(sf[i], alpha=10, tune=true, weight=2)

  # Instantaneous rate matrix (deterministic) #
  Q[i] := fnGTR(er[i],sf[i])

  # Gamma-dist site rates #
  shape_prior[i] <- 0.05
  shape[i] ~ dnExponential( shape_prior[i] )
  gamma_rates[i] := fnDiscretizeGamma( shape[i], shape[i], 4 )
  moves[mi++] <- mvScale(shape[i], lambda=0.8, tune=true, weight=3.0)
}
```

Uniform Topology and Branch Lengths

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.

```
# Unrooted tree topology distribution #
topology ~ dnUniformTopology(n_species, names)

# Tree topology moves #
moves[mi++] <- mvNNI(topology, weight=10.0)
moves[mi++] <- mvSPR(topology, weight=5.0)

#### Specify a prior and moves on the branch lengths ####
# Create a vector of branch-length variables using a for loop #
for (i in 1:n_branches) {
  br_lens[i] ~ dnExponential(10.0)
  moves[mi++] <- mvScale(br_lens[i],lambda=1,tune=true,weight=1)
}

# A deterministic node for the tree length #
tree_length := sum(br_lens)

# Build the tree by combining the topology with br_lens #
phylogeny := treeAssembly(topology, br_lens)
```

Putting it All Together

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 **`phylogeny`** 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=phylogeny, Q=Q[i], siteRates=
     gamma_rates[i], nSites=data[i].nchar(1), type="DNA")
  phyloSeq[i].clamp(data[i])
}
```

And we can pass in a single, shared node to wrap up our model DAG:

```
RevBayes > mymodel <- model(topology)
```

Estimating the Marginal Likelihood

Now run the power posterior analysis on the two-gene model.

```
RevBayes > pow_p <- powerPosterior(mymodel, moves, file="pow_p_twogene.
   out", cats=50)
RevBayes > pow_p.burnin(generations=1000,tuningInterval=100)
RevBayes > pow_p.run(generations=1000)
```

Calculate the marginal likelihoods under stepping-stone sampling:

```
RevBayes > ss <- steppingStoneSampler(file="pow_p_twogene.out",
   powerColumnName="power", likelihoodColumnName="likelihood")
RevBayes > ss.marginal()
```

And under path sampling:

```
RevBayes > ps <- pathSampler(file="pow_p_twogene.out", powerColumnName="
   power", likelihoodColumnName="likelihood")
RevBayes > ps.marginal()
```

Record the marginal likelihoods in Table 6.1.

4.1.6 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 rbcL gene by codon position.

Clear Workspace and Reload Data

```
RevBayes > clear()
RevBayes > data[1] <- readDiscreteCharacterData("data/conifer_atpB.nex")
   [1]
RevBayes > data_rbcL <- readDiscreteCharacterData("data/conifer_rbcL.nex
   ")[1]
```

Specify Data Matrices for Each Codon Position

We must now add our codon-partitions to the **data** vector that already contains the matrix for atpB in the first index. Thus, the second index will be the rbcL codon position 1. 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[2]** to exclude all but the 1^{st} positions.

```
RevBayes > data[2] <- data_rbcL
RevBayes > data[2].setCodonPartition(1)
```

Assign the 2^{nd} codon positions to **data[3]**:

```
RevBayes > data[3] <- data_rbcL
RevBayes > data[3].setCodonPartition(2)
```

Assign the 3^{rd} codon positions to **data[4]**:

```
RevBayes > data[4] <- data_rbcL
RevBayes > data[4].setCodonPartition(3)
```

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

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

And set the special variables from the data:

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

Specify the Parameters by Looping Over Partitions

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 each of the partitions.

```
mi <- 0 # an iterator for the move vector
for (i in 1:n_parts){
  ## index i=1 : atpB gene
  ## index i=2 : rbcL gene position 1
  ## index i=3 : rbcL gene position 2
  ## index i=4 : rbcL gene position 3

  # Exchangeability rates #
  er_prior[i] <- v(1,1,1,1,1,1)
  er[i] ~ dnDirichlet(er_prior[i])
  moves[mi++] <- mvSimplexElementScale(er[i], alpha=10, tune=true, weight=3)

  # Stationary base frequencies #
  sf_prior[i] <- v(1,1,1,1)
  sf[i] ~ dnDirichlet(sf_prior[i])
  moves[mi++] <- mvSimplexElementScale(sf[i], alpha=10, tune=true, weight=2)

  # Instantaneous rate matrix (deterministic) #
  Q[i] := fnGTR(er[i],sf[i])

  # Gamma-dist site rates #
  shape_prior[i] <- 0.05
  shape[i] ~ dnExponential( shape_prior[i] )
  gamma_rates[i] := fnDiscretizeGamma( shape[i], shape[i], 4 )
  moves[mi++] <- mvScale(shape[i], lambda=0.8, tune=true, weight=3.0)
}
```

Uniform Topology and Branch Lengths

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

```
# Unrooted tree topology distribution #
topology ~ dnUniformTopology(n_species, names)

# Tree topology moves #
moves[mi++] <- mvNNI(topology, weight=10.0)
moves[mi++] <- mvSPR(topology, weight=5.0)

#### Specify a prior and moves on the branch lengths ####
# Create a vector of branch-length variables using a for loop #
for (i in 1:n_branches) {
  br_lens[i] ~ dnExponential(10.0)
  moves[mi++] <- mvScale(br_lens[i],lambda=1,tune=true,weight=1)
}

# A deterministic node for the tree length #
tree_length := sum(br_lens)

# Build the tree by combining the topology with br_lens #
```

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

Putting it All Together

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

```
for (i in 1:n_parts){
  phyloSeq[i] ~ dnPhyloCTMC(tree=phylogeny, Q=Q[i], siteRates=
     gamma_rates[i], nSites=data[i].nchar(1), type="DNA")
  phyloSeq[i].clamp(data[i])
}
```

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

```
RevBayes > mymodel <- model(topology)
```

Estimating the Marginal Likelihood

Sample likelihoods from the set of power posteriors:

```
RevBayes > pow_p <- powerPosterior(mymodel, moves, file="
   pow_posterior_genecodon.out", cats=50)
RevBayes > pow_p.burnin(generations=1000,tuningInterval=100)
RevBayes > pow_p.run(generations=1000)
```

Compute the stepping-stone estimate of the marginal likelihood:

```
RevBayes > ss <- steppingStoneSampler(file="pow_posterior_genecodon.out
   ", powerColumnName="power", likelihoodColumnName="likelihood")
RevBayes > ss.marginal()
```

Compute the path-sampling estimate of the marginal likelihood:

```
RevBayes > ps <- pathSampler(file="pow_posterior_genecodon.out",
   powerColumnName="power", likelihoodColumnName="likelihood")
RevBayes > ps.marginal()
```

Now record the marginal likelihoods in Table 6.1.

4.1.7 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 6.1 to summarize the marginal log-likelihoods estimated using the stepping-stone and path-sampling methods.

Table 4.2: Estimated marginal likelihoods for different partition configurations*.

	Marginal lnL estimates	
Partition	*Stepping-stone*	*Path sampling*
4.1.4 uniform (M_1)		
4.1.5 moderate (M_2)		
4.1.6 extreme (M_3)		

*you can edit this table

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 use a different form of equation 4.3 to calculate the ln-Bayes factor (we will denote this value $\mathcal{K}$):

$$\mathcal{K} = \ln[BF(M_0, M_1)] = \ln[\mathbb{P}(\mathbf{X} \mid M_0)] - \ln[\mathbb{P}(\mathbf{X} \mid M_1)], \tag{4.4}$$

where $\ln[\mathbb{P}(\mathbf{X} \mid M_0)]$ is the *marginal lnL* estimate for model M_0. The value resulting from equation 6.3 can be converted to a raw Bayes factor by simply taking the exponent of $\mathcal{K}$

$$BF(M_0, M_1) = e^{\mathcal{K}}. \tag{4.5}$$

Alternatively, you can interpret the strength of evidence in favor of M_0 using the $\mathcal{K}$ and skip equation 6.4. 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 6.1 and equation 6.3, calculate the ln-Bayes factors (using $\mathcal{K}$) for the different model comparisons. Enter your answers in Table 5.3 using the stepping-stone and the path-sampling estimates of the marginal log likelihoods.

Once you complete Table 5.3, you will notice that the Bayes factor comparison indicates strong evidence in support of the highly partitioned model using both the stepping-stone and path sampling estimates of the marginal likelihoods. However, this does not mean that model M_3 is the *true* partition model. We only considered three out of the many, many possible partitions for 2,659 sites (the number of possible partitions can be viewed if you compute the 2659th Bell number). Given the strength of support for the highly partitioned model, it is possible that further partitioning is warranted for these data. In particular, partitioning the dataset by codon position for both atpB *and* rbcL is an important next step for this exercise (consider taking some time on your own to test this model).

Table 4.3: Bayes factor calculation*.

Model comparison	**ln-Bayes Factor** ($\mathcal{K}$) *Stepping-stone*	*Path sampling*
M_1, M_2		
M_2, M_3		
M_1, M_3		
Supported model?		

*you can edit this table

Because of the computational costs of computing marginal likelihoods and the vast number of possible partitioning strategies, it is not feasible to evaluate all of them. New methods based on nonparametric Bayesian models have recently been applied to address this problem (Lartillot and Philippe 2004; Huelsenbeck and Suchard 2007; Wu, Suchard and Drummond 2013). These approaches use an infinite mixture model (the Dirichlet process; Ferguson 1973; Antoniak 1974) that places non-zero probability on *all* of the countably-infinite possible partitions for a set of sequences. Bayesian phylogenetic inference under these models is implemented in the program PhyloBayes (Lartillot, Lepage and Blanquart 2009) and the subst-bma plug-in for BEAST2 (Wu, Suchard and Drummond 2013).

Note that Bayes factors based on comparison of HM-based marginal likelihoods often *strongly* favor the most extremely partitioned mixed model. In fact, the harmonic mean estimator has been shown to provide unreliable estimates of marginal likelihoods, compared to more robust approaches (Lartillot and Philippe 2006; Xie et al. 2011; Fan et al. 2011). Based on these studies, it is recommended that you avoid using HM-derived marginal likelihoods for Bayes factor comparisons. (The Canadian Bayesian Radford Neal says the harmonic mean is the "worst Monte Carlo method ever".)

4.1.8 Perform MCMC Analysis Under Preferred Model

Clear Workspace and Load the Data and Model

```
RevBayes > clear()
RevBayes > source("RevBayes_scripts/<preferred>_partition_model.Rev")
```

Specify Monitors

```
RevBayes > monitors[1] <- mnModel(filename="conifer_prefmodel_mcmc.log",
    printgen=100)
```

```
RevBayes > monitors[2] <- mnFile(filename="conifer_prefmodel_mcmc.trees
   ",printgen=100, phylogeny)
```

```
RevBayes > monitors[3] <- mnScreen(printgen=10, separator = " | ",
   tree_length)
```

Initialize and Run MCMC

```
RevBayes > mymcmc <- mcmc(mymodel, monitors, moves)
```

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

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

4.1.9 Summarize and Analyze MCMC Output

```
RevBayes > treetrace <- readTreeTrace("conifer_prefmodel_mcmc.trees")
RevBayes > treetrace.summarize()
```

```
RevBayes > mapTree(treetrace,"conifer_prefmodel_MAP.tre")
```

The trees in these files are also annotated with various branch- or node-specific parameters or statistics in an extended Newick format called NHX. We can use FigTree to visualize these summary trees.

- Open the summary tree in FigTree: **`conifer_prefmodel_MAP.tre`**.

- Use the tools on the side panel to display the posterior probabilities as node labels.

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 full_analysis.Rev`**

Useful Links

- RevBayes: https://github.com/revbayes/code
- MrBayes: http://mrbayes.sourceforge.net
- PhyloBayes: http://www.phylobayes.org
- Bali-Phy: http://www.bali-phy.org
- Tree Thinkers: http://treethinkers.org

Questions about this tutorial can be directed to:

- Tracy Heath (email: tracyh@berkeley.edu)
- Michael Landis (email: mlandis@berkeley.edu)
- Sebastian Höhna (email: sebastian.hoehna@gmail.com)

Bibliography

Antoniak CE. 1974. Mixtures of Dirichlet processes with applications to non-parametric problems. Annals of Statistics. 2:1152–1174.

Brandley MC, Schmitz A, Reeder TW. 2005. Partitioned Bayesian analyses, partition choice, and the phylogenetic relationships of scincid lizards. Systematic Biology. 54:373–390.

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

Fan Y, Wu R, Chen MH, Kuo L, Lewis PO. 2011. Choosing among partition models in Bayesian phylogenetics. Molecular Biology and Evolution. 28:523–532.

Ferguson TS. 1973. A Bayesian analysis of some nonparametric problems. Annals of Statistics. 1:209–230.

Friel N, Pettitt AN. 2008. Marginal likelihood estimation via power posteriors. Journal of the Royal Statistical Society: Series B (Statistical Methodology). 70:589–607.

Gelman A, Meng XL. 1998. Simulating normalizing constants: From importance sampling to bridge sampling to path sampling. Statistical science. pp. 163–185.

Huelsenbeck JP, Suchard M. 2007. A nonparametric method for accommodating and testing across-site rate variation. Systematic Biology. 56:975–987.

Jeffreys H. 1961. Theory of Probability. Oxford: Oxford University Press.

Kass RE, Raftery AE. 1995. Bayes factors. Journal of the American Statistical Association. 90:773–795.

Lartillot N, Lepage T, Blanquart S. 2009. Phylobayes 3: a Bayesian software package for phylogenetic reconstruction and molecular dating. Bioinformatics. 25:2286.

Lartillot N, Philippe H. 2004. A Bayesian mixture model for across-site heterogeneities in the amino-acid replacement process. Molecular Biology and Evolution. 21:1095–1109.

Lartillot N, Philippe H. 2006. Computing Bayes factors using theromodynamic integration. Systematic Biology. 55:195–207.

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

McGuire JA, Witt CC, Altshuler DL, Remsen JV. 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 JAA, Ronquist F, Huelsenbeck JP, Aldrey JLN. 2004. Bayesian phylogenetic analysis of combined data. Systematic Biology. 53:47–67.

Rambaut A, Drummond AJ. 2009. Tracer v1.5. Edinburgh (United Kingdom): Institute of Evolutionary Biology, University of Edinburgh. Available from: http://beast.bio.ed.ac.uk/Tracer.

Ronquist F, Huelsenbeck JP. 2003. Mrbayes 3: Bayesian phylogenetic inference under mixed models. Bioinformatics. 19:1572–1574.

Suchard MA, Weiss RE, Sinsheimer JS. 2001. Bayesian selection of continuous-time Markov chain evolutionary models. Molecular Biology and Evolution. 18:1001–1013.

Wu CH, Suchard MA, Drummond AJ. 2013. Bayesian selection of nucleotide substitution models and their site assignments. Molecular Biology and Evolution. 30:669–688.

Xie W, Lewis PO, Fan Y, Kuo L, Chen MH. 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.

Zwickl DJ, Holder MT. 2004. Model parameterization, prior distributions, and the general time-reversible model in Bayesian phylogenetics. Systematic Biology. 53:877–888.

Chapter 5

Dating and Relaxed Clocks

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

5.1.1 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`.

5.1.2 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/1tFOXXX

Download the alignment file: http://bit.ly/1xs6pEd

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 5.1).

Table 5.1: 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, Tedford and Taylor (1999)
Parictis montanus	33.9–37.2	Clark and Guensburg (1972); Krause et al. (2008)
Kretzoiarctos beatrix	11.2–11.8	Abella, Montoya and Morales (2011); Abella et al. (2012)

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

5.1.3 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_RelaxedClock_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 5.1.2).

Create a directory called **`data`** in your tutorial directory.

Save the tree and alignment files downloaded above (Section 5.1.2) 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` GitHub repository at this URL: http://bit.ly/1zj5u9n. You can refer to these examples to verify your own work.

5.1.4 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 5.1. 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, Yang and Donoghue 2012; Heled and Drummond 2012; Heath, Huelsenbeck and Stadler 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 5.1 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. 5.1).

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. 5.1).

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

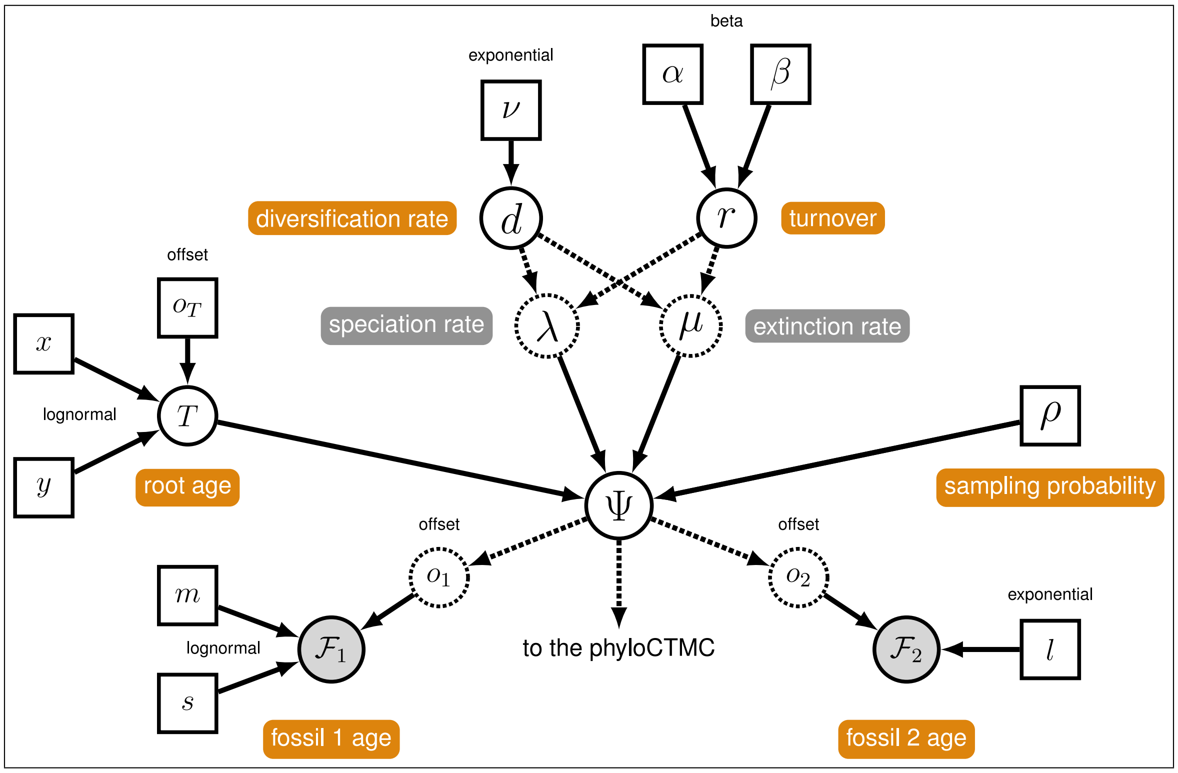


Figure 5.1: The graphical model representation of the node-calibrated birth-death process in `RevBayes`.

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.

```
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.

5.1.5 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. When estimating trees on an absolute time-scale, it is often necessary to parameterize relaxed clock models with two rates, a base rate which effectively scales the tree and a clock rate. Then, the absolute rate applied to the tree is a deterministic node (Fig. 5.2).

Create the Rev File

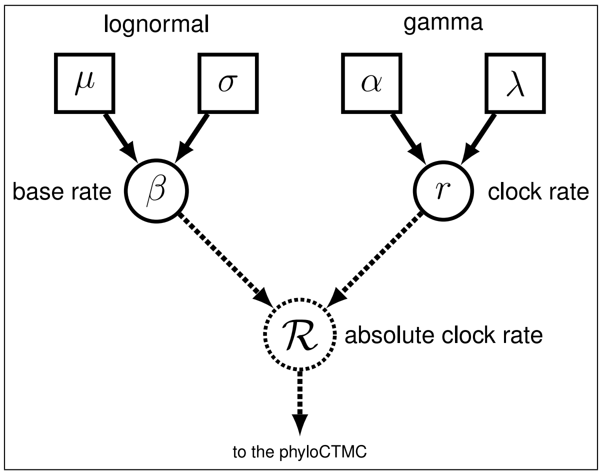


Figure 5.2: The graphical model representation of the global molecular clock model used in this exercise.

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

We specify the absolute clock rate by first creating a node for the base rate. This value is set to be drawn from a lognormal prior.

```
br_M <- 5.4E-3
br_s <- 0.05
br_mu <- ln(br_M) - ((br_s * br_s) * 0.5)
base_rate ~ dnLnorm(br_mu, br_s)
moves[mi++] = mvScale(base_rate,lambda=0.25,tune=true,weight=5.0)
```

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 absolute clock rate is the value on which the phylogenetic CTMC model depends. This is a deterministic node and equal to the product of the base rate and clock rate.

```
abs_clock_rt := clock_rate * base_rate
```

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 phyoCTMC.

```
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 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, "output/GMC_bears_powp.out", cats
   =50)
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 6.1.

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. 5.3).

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 5.3, we can represent the vector of N branch rates using the plate notation. Additionally, each branch rate is rescaled by the base rate.

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.

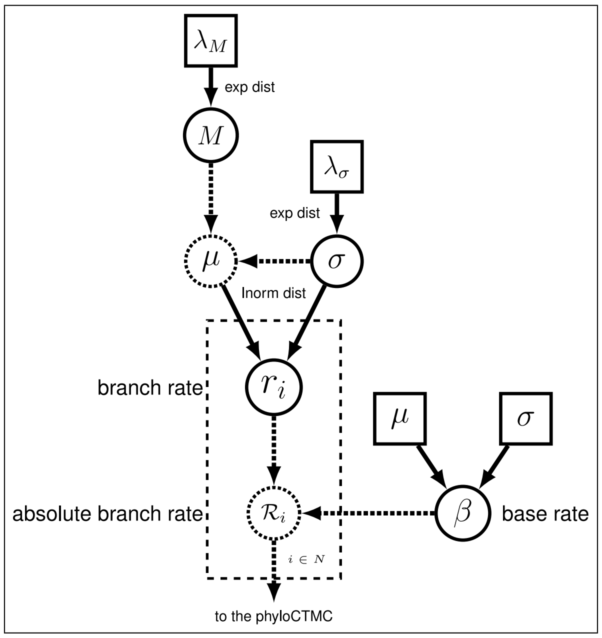


Figure 5.3: The graphical model representation of the UCLN model used in this exercise.

The Base Clock Rate

As in the strict clock model above, we create a lognormally distributed stochastic node, representing the base rate.

```
br_M <- 5.4E-3
br_s <- 0.05
br_mu <- ln(br_M) - ((br_s * br_s) * 0.5)
base_rate ~ dnLnorm(br_mu, br_s)
moves[mi++] = mvScale(base_rate,lambda=0.25,tune=true,weight=5.0)
```

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 5.3.

```
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.)
}
```

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)
```

We can combine the base rate and branch rates in a vector of deterministic nodes.

```
branch_subrates := branch_rates * base_rate
```

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 6.1.

The Autocorrelated Lognormal Rates Model

A model assuming that the rate at each node is lognormally distributed with a mean centered on its parent rate and a variance proportional to the time-duration since the parent node is an autocorrelated model (ACLN; Thorne, Kishino and Painter 1998; Kishino, Thorne and Bruno 2001; Thorne and Kishino 2002). This corresponds to a geometric Brownian motion model. The ACLN model relies on the topology and branch-durations of the time-tree and is thus more complex to represent graphically. Thus, we use the convenience of the tree plate to show the conditional dependence structure among node rates and ages in Figure 5.4.

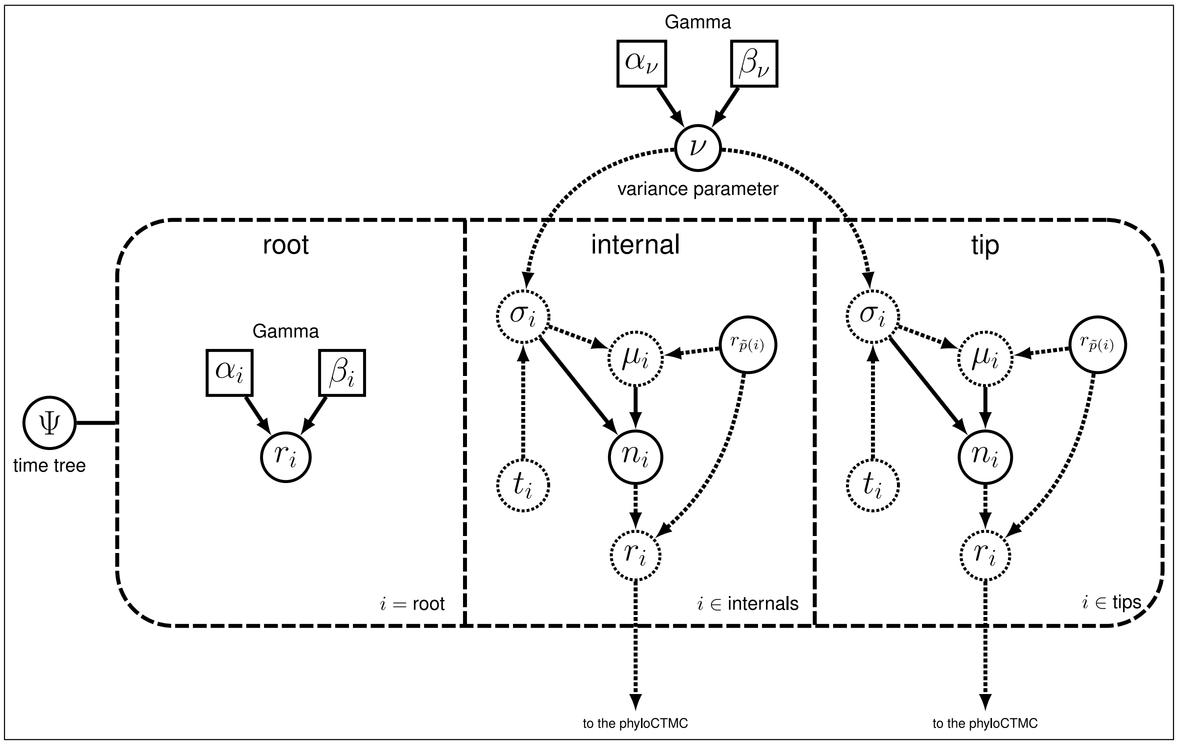


Figure 5.4: The graphical model representation of the ACLN model used in this exercise.

In this model, for any node (internal or tip) that is not the root, the rate at that node r_i is drawn from a lognormal distribution with an expected value equal to the rate of the parent node $r_{\tilde{p}(i)}$ and a variance that is the product of the time difference t_i and the variance parameter ν. Note that the graphical model represented in Figure 5.4 is simplified to to make it easier to understand, thus some deterministic nodes are obfuscated. Importantly, it is worth recognizing that the ACLN model describes the rate values at the *nodes* of the tree and not the branches. Because of this, additional deterministic nodes are used to compute the rate along branch i, b_i, which is the average of the two nodes subtending that branch: $b_i = (r_i + r_{\tilde{p}(i)})/2$.

Create the Rev File

Open your text editor and create the autocorrelated-lognormal relaxed-clock model file called `m_ACLN_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 Base Clock Rate

As in the strict clock and UCLN models above, we create a lognormally distributed stochastic node, representing the base rate.

```
br_M <- 5.4E-3
br_s <- 0.05
br_mu <- ln(br_M) - ((br_s * br_s) * 0.5)
base_rate ~ dnLnorm(br_mu, br_s)
moves[mi++] = mvScale(base_rate,lambda=0.25,tune=true,weight=5.0)
```

Autocorrelated Node Rates

Begin by declaring the parameters of the ACLN model. The first is the parameter **`nu`** which determines the degree of autocorrelation among node rates. If **`nu`** is very large, then the variance of the lognormal distribution on node rates is also very large, resulting in low autocorrelation. Conversely, if **`nu`** is very small, then closely related nodes will have very similar rates. And if **`nu = 0`** the model collapses to a strict clock, where all nodes have the same substitution rate. For this dataset we will assign an exponential prior to **`nu`** with an expected value of 1.0 and use a scale-type move to propose changes.

```
nu ~ dnExponential(1.0)
moves[mi++] = mvScale(nu, lambda=0.5, tune=true, weight=4.0)
```

The next parameter of the ACLN model is the rate value at the root of the tree. We will assume that this rate is drawn from a gamma distribution with an expected value of 0.5.

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

Now we can declare our stochastic node containing the node rates. This is conditioned on the **`timetree`** node we defined by our birth-death model, the variance parameter **`nu`**, the **`root_rate`**, and the **`base_rate`**.

```
node_rates ~ dnACLN(timetree, nu, root_rate, base_rate)
```

Because the ACLN model describes the distribution of rates at the nodes of the tree, we must compute the rate for each *branch* as a vector of deterministic nodes. Where the rate for a given branch is the average of the rates a the nodes subtending that branch. For this, we can use an explicit function written at the `Rev` language level that takes the tree and all other parameters of the ACLN model and looks up the relevant parameters for a given index. We can declare a vector of deterministic branch rates using a **`for`** loop.

```
n_branches <- 2 * n_taxa - 2
for(i in 1:n_branches){
   branch_rates[i] := aveRateOnBranch(node_rates, timetree, root_rate,
      base_rate, index=i)
}
```

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)
```

Defining moves for a model like the ACLN is a bit tricky. Because of strong parameter interactions among all of the node rates and ages, the model doesn't mix well with standard moves. The autocorrelation may also make it difficult to efficiently sample the tree topology jointly under this model. Thus, it is worth putting some thought into the MCMC proposals. In `RevBayes` there are two moves especially for the ACLN-distributed node rates. The first, **`mvScaleSingleACLNRates`**, selects a single non-root node at random and proposes a new value for the rate using a scale-type move. Because of the variance in rates across nodes, it is difficult to specify the move so that the tuning parameter is optimal for all nodes. Therefore, it may be necessary to declare multiple instances of this move with different tuning values that are not changed during burn-in. Additionally, we can also add a move that is retuned so that the proposal size is optimal for the average node rate.

```
moves[mi++] = mvScaleSingleACLNRates(node_rates, 4.0, false, n_branches)
moves[mi++] = mvScaleSingleACLNRates(node_rates, 2.0, false, n_branches)
moves[mi++] = mvScaleSingleACLNRates(node_rates, 3.0, true, 3*
   n_branches)
```

The second move for ACLN node rates is a "mixing step" (Thorne, Kishino and Painter 1998; Kishino, Thorne and Bruno 2001; Thorne and Kishino 2002; Rannala and Yang 2003; Yang and Rannala 2006). This move rescales the node ages and proportionally changes the node rates so that the absolute branch lengths remain unchanged and the likelihood is unaffected. This type of proposal can help your chain more efficiently sample parameter space without requiring costly recalculations of the model likelihood.

```
moves[mi++] = mvACLNMixingStep(timetree, node_rates, root_rate, 0.5,
   false, n_branches)
```

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_ACLN_bears.Rev` in the `RevBayes_scripts` directory.

Estimate the Marginal Likelihood

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

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

Refer to the section describing this process for the GMC and UCLN models above. Write your own `Rev` language script to estimate the marginal likelihood under the ACLN model. Be sure to change the file names in all of the relevant places. Additionally, you may find that the power-posterior analysis runs too slow under this model, thus it may be advisable for you to decrease the number of cycles or tuning frequency in the burn-in period.

Once you have completed this analysis, record the marginal likelihoods under the ACLN model in Table 6.1.

5.1.6 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 6.1 to summarize the marginal log-likelihoods estimated using the stepping-stone and path-sampling methods.

Table 5.2: Estimated marginal likelihoods for different across-branch substitution rate models*.

	Marginal lnL estimates	
Clock Model	*Stepping-stone*	*Path sampling*
5.1.5 Global molecular clock (M_0)		
5.1.5 Uncorrelated lognormal (M_1)		
5.1.5 Autocorrelated lognormal (M_2)		

*you can edit this table

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 $\mathcal{K}$):

$$\mathcal{K} = \ln[BF(M_0, M_1)] = \ln[\mathbb{P}(\mathbf{X} \mid M_0)] - \ln[\mathbb{P}(\mathbf{X} \mid M_1)], \tag{5.1}$$

where $\ln[\mathbb{P}(\mathbf{X} \mid M_0)]$ is the *marginal lnL* estimate for model M_0. The value resulting from equation 6.3 can be converted to a raw Bayes factor by simply taking the exponent of $\mathcal{K}$

$$BF(M_0, M_1) = e^{\mathcal{K}}. \tag{5.2}$$

Alternatively, you can interpret the strength of evidence in favor of M_0 using the $\mathcal{K}$ and skip equation 6.4. In this case, we evaluate the $\mathcal{K}$ in favor of model M_0 against model M_1 so that:

$$\begin{array}{l}\text{if } \mathcal{K} > 1, \text{ then model } M_0 \text{ wins} \\ \text{if } \mathcal{K} < -1, \text{ then model } M_1 \text{ wins.}\end{array}$$

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

Using the values you entered in Table 6.1 and equation 6.3, calculate the ln-Bayes factors (using $\mathcal{K}$) for the different model comparisons. Enter your answers in Table 5.3 using the stepping-stone and the path-sampling estimates of the marginal log likelihoods.

5.1.7 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.

Table 5.3: Bayes factor calculation based on marginal likelihood estimates in Table 6.1*.

	ln-Bayes Factor ($\mathcal{K}$)	
Model comparison	*Stepping-stone*	*Path sampling*
M_0, M_1		
M_0, M_2		
M_1, M_2		
Supported model?		

*you can edit this table

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 and base rate 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, base_rate)
```

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")`

Useful Links

- RevBayes: https://github.com/revbayes/revbayes

- Tree Thinkers: http://treethinkers.org

Questions about this tutorial can be directed to:

- Tracy Heath (email: tracyh@berkeley.edu)
- Tanja Stadler (email: tanja.stadler@bsse.ethz.ch)
- Sebastian Höhna (email: sebastian.hoehna@gmail.com)

Bibliography

Abella J, Alba DM, Robles JM, Valenciano A, Rotgers C, Carmona R, Montoya P, Morales J. 2012. *Kretzoiarctos* gen. nov., the oldest member of the giant panda clade. PLoS One. 17:e48985.

Abella J, Montoya P, Morales J. 2011. Una nueva especie de *Agriarctos* (Ailuropodinae, Ursidae, Carnivora) en la localidad de Nombrevilla 2 (Zaragoza, España). Estudios Geológicos. 67:187–191.

Clark J, Guensburg TE. 1972. Arctoid Genetic Characters as Related to the Genus *Parictis*, volume 1150. Chicago, Ill.: Field Museum of Natural History.

dos Reis M, Inoue J, Hasegawa M, Asher R, Donoghue P, Yang Z. 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 TA, Huelsenbeck JP, Stadler T. 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, Drummond AJ. 2012. Calibrated tree priors for relaxed phylogenetics and divergence time estimation. Systematic Biology. 61:138–149.

Kishino H, Thorne JL, Bruno W. 2001. Performance of a divergence time estimation method under a probabilistic model of rate evolution. Molecular Biology and Evolution. 18:352–361.

Krause J, Unger T, Noçon A, et al. (11 co-authors). 2008. Mitochondrial genomes reveal an explosive radiation of extinct and extant bears near the Miocene-Pliocene boundary. BMC Evolutionary Biology. 8:220.

Rambaut A, Drummond AJ. 2009. Tracer v1.5. Edinburgh (United Kingdom): Institute of Evolutionary Biology, University of Edinburgh. Available from: http://beast.bio.ed.ac.uk/Tracer.

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.

Thorne J, Kishino H. 2002. Divergence time and evolutionary rate estimation with multilocus data. Systematic Biology. 51:689–702.

Thorne J, Kishino H, Painter IS. 1998. Estimating the rate of evolution of the rate of molecular evolution. Molecular Biology and Evolution. 15:1647–1657.

Wang X. 1994. Phylogenetic Systematics of the Hesperocyoninae (Carnivora, Canidae), volume 221.

Wang X, Tedford RH, Taylor BE. 1999. Phylogenetic Systematics of the Borophaginae (Carnivora, Canidae), volume 243.

Warnock RCM, Yang Z, Donoghue PCJ. 2012. Exploring the uncertainty in the calibration of the molecular clock. Biology Letters. 8:156–159.

Yang Z, Rannala B. 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 6

Simple Diversification Rate Estimation

6.1 Exercise: Estimating Speciation & Extinction Rates

6.1.1 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 1948; Thompson 1975; Nee, May and Harvey 1994; Rannala and Yang 1996; Yang and Rannala 1997; Aldous 2001; Popovic 2004; Aldous and Popovic 2005; Gernhard 2008; Stadler 2009). 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.

6.1.2 Getting Started

This tutorial assumes that you have already downloaded, compiled, and installed RevBayes. We also recommend that—if you are working on a Unix machine—you put the `rb` binary in your path.

For the exercises outlined in this tutorial, we will use `RevBayes` interactively by typing commands in the command-line console. The format of this exercise uses `lavender blush shaded boxes` to delineate important steps. The various `RevBayes` commands and syntax are specified using **`typewriter text`**. And the specific commands that you should type (or copy/paste) into `RevBayes` are indicated by shaded box and prompt. For example, after opening the `RevBayes` program, you can create a constant node **`x`**:

```
x <- 12.0
```

For this step, type (or paste) in the line contents: **`x <- 12.0`**. The prompt, **`>`**, that you see in the RevBayes console is not replicated here so that you do not inadvertently copy and paste it with the `Rev` syntax

Multi-line entries, particularly loops, will also be displayed in boxes without the **`>`** prompt so that they can be copied and pasted wholly.

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

This tutorial 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.

- Download data and `Rev` files from: http://bit.ly/1tEDwTg

6.1.3 The "Observed" Data

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. 6.1). 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.

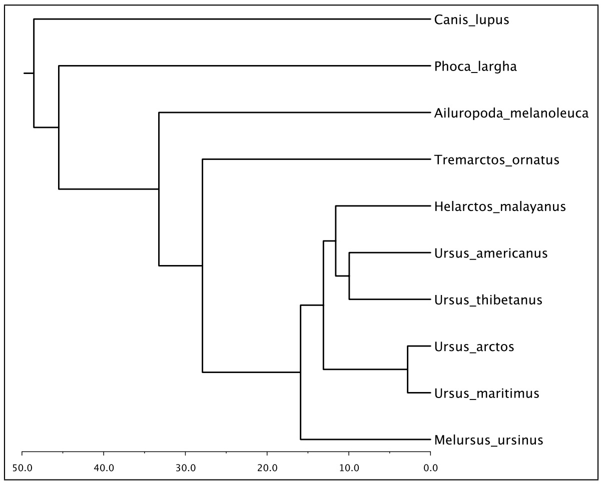


Figure 6.1: The relationships and median speciation times for bears and two outgroups estimated in the analysis by dos Reis et al. (2012).

6.1.4 Launch RevBayes

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 the program information, including the current version number the URL for the RevBayes Project: RevBayes.com. Additionally, RevBayes has a number of resources documenting the syntax, walkthroughs of methods, and general help tools. Some of these resources are given below, but please consult the website for more information.

Getting Help

In the RevBayes console, you can query the help system for any function by typing **`?`** followed by the function name. For example, if you would like detailed information about the **`mean()`** function type:

```
?mean
```

This will provide a description of the function as well as information about the argument and return types. The function help also gives a working example of the use of the function:

```
# compute the mean of some random draws from a normal distribution
x <- rnorm(n=10, 5,1)
mean(x)
```

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

6.1.5 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.

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, Wong and Stadler 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. 6.2).

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. 6.3). The graphical models in Figures 6.2 and 6.3 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.

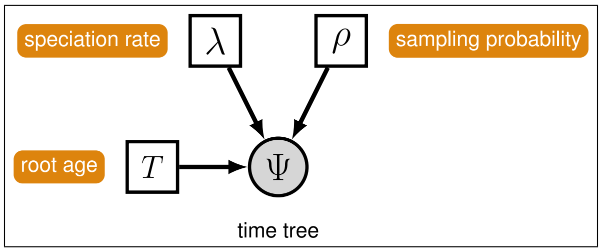


Figure 6.2: The graphical model representation of the pure-birth (Yule) process.

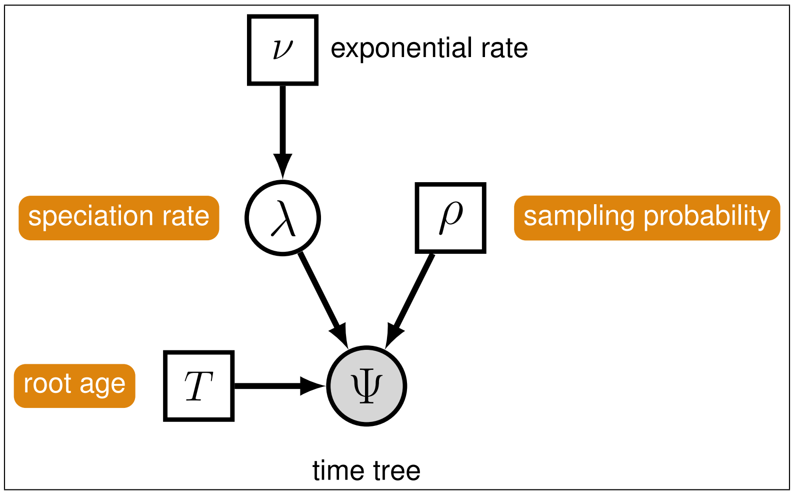


Figure 6.3: The graphical model representation of the pure-birth (Yule) process, where the speciation rate is treated as a random variable drawn from an exponential distribution with rate parameter ν.

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 6.3. In a Bayesian framework, we are interested in estimating the posterior probability of λ given that we observe a time tree.

$$\mathbb{P}(\lambda \mid \Psi) = \frac{\mathbb{P}(\Psi \mid \lambda)\mathbb{P}(\lambda \mid \nu)}{\mathbb{P}(\Psi)} \tag{6.1}$$

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 6.1. 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`.

Read the tree

Begin by reading in the observed tree from Figure 6.1.

```
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. 6.3). 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 6.1 and Figure 6.3, 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 6.4. Compare this figure to the model in Figure 6.3. You should notice that there are two extra nodes in the Figure 6.4. 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.

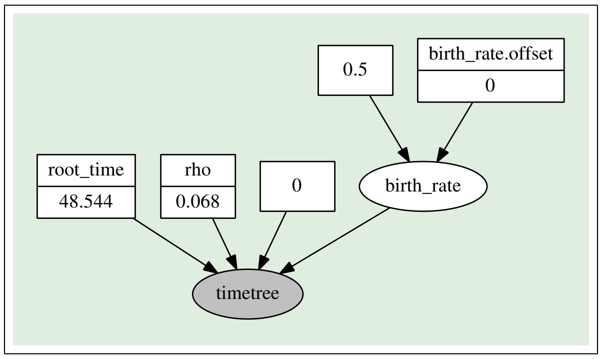


Figure 6.4: The graphical model representation of the pure-birth (Yule) process generated using the DOT language and the Graphviz program.

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. 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 6.1.

```
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 6.1.

```
ps.marginal()
```

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 (1948) 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, May and Harvey 1994). Importantly, this model assumes that all of the extant descendants of the process have been sampled at time 0.

Yang and Rannala (1997) and Stadler (2009) derived the probability of time trees under an extension of the birth-death model that accounts for incomplete sampling of the tips (Fig. 6.5). 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.

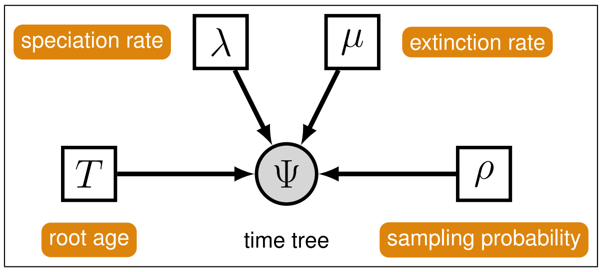


Figure 6.5: The graphical model representation of the birth-death process with uniform sampling and conditioned on the root age.

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:

$$\lambda = \frac{d}{1-r}, \quad \mu = \frac{rd}{1-r}. \tag{6.2}$$

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. 6.6). Note that if $\mu = 0$, then $d = \lambda$ in this parameterization.

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`.

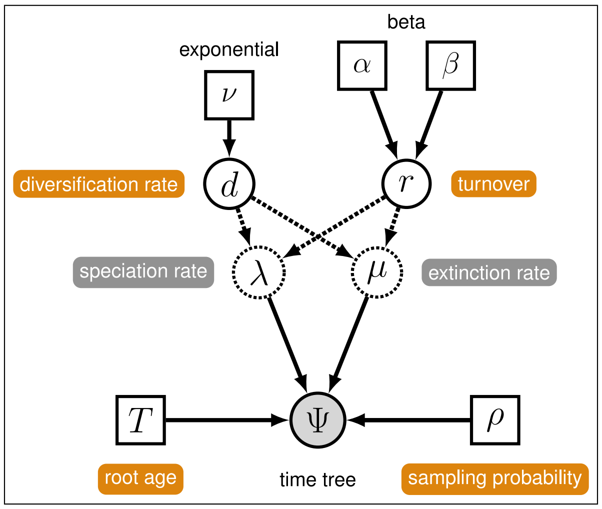


Figure 6.6: The graphical model representation of the birth-death process with uniform sampling parameterized using the diversification and turnover.

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 6.1.

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

Initialize the useful variables:

```
n_taxa <- T.ntips()
names <- T.names()
mi = 1
```

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)
```

Birth rate and death rate

The birth and death rates are both deterministic nodes. Refer to Equation 6.2. 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))
```

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
```

Root age

Get the value for the root from the dos Reis et al. (2012) tree.

```
root_time <- treeHeight(T)
```

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")
```

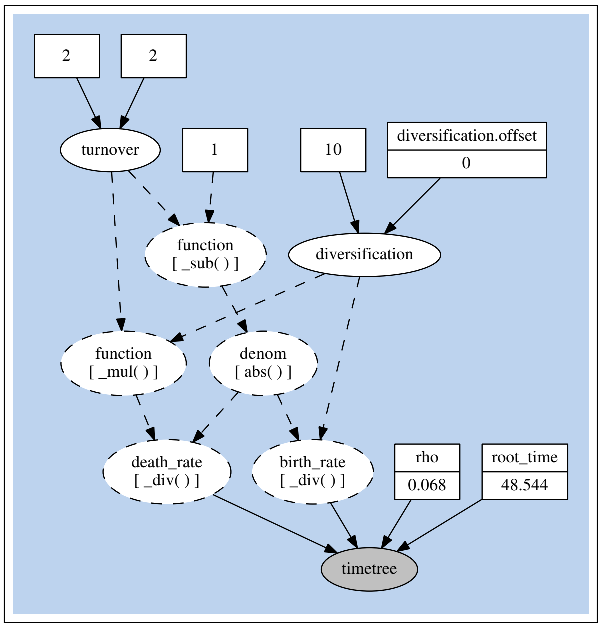


Figure 6.7: The graphical model representation of the birth-death process.

Estimating the marginal likelihood of the model

- 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 6.1.

```
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 6.1.

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

6.1.6 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 6.1 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 $\mathcal{K}$):

$$\mathcal{K} = \ln[BF(M_0, M_1)] = \ln[\mathbb{P}(\mathbf{X} \mid M_0)] - \ln[\mathbb{P}(\mathbf{X} \mid M_1)], \tag{6.3}$$

where $\ln[\mathbb{P}(\mathbf{X} \mid M_0)]$ is the *marginal lnL* estimate for model M_0. The value resulting from equation 6.3 can be converted to a raw Bayes factor by simply taking the exponent of $\mathcal{K}$

$$BF(M_0, M_1) = e^{\mathcal{K}}. \tag{6.4}$$

Alternatively, you can interpret the strength of evidence in favor of M_0 using the $\mathcal{K}$ and skip equation 6.4. In this case, we evaluate the $\mathcal{K}$ in favor of model M_0 against model M_1 so that:

$$\begin{array}{l}\text{if } \mathcal{K} > 1, \text{ then model } M_0 \text{ wins}\\ \text{if } \mathcal{K} < -1, \text{ then model } M_1 \text{ wins.}\end{array}$$

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

Using the values you entered in Table 6.1 and equation 6.3, calculate the ln-Bayes factors (using $\mathcal{K}$) for the different model comparisons. Enter your answers in Table 6.1 using the stepping-stone and the path-sampling estimates of the marginal log likelihoods.

Do these data support a model without extinction ($\mu = 0$)?

Table 6.1: Marginal likelihoods and Bayes factors*.

Estimate	*Stepping-stone*	*Path sampling*
6.1.5 Marginal likelihood Yule (M_0)		
6.1.5 Marginal likelihood birth-death (M_1)		
Eq. 6.3: $BF(M_0, M_1)$		
Supported model?		

*you can edit this table

6.1.7 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.

$$\mathbb{P}(\lambda, \mu \mid \Psi) = \frac{\mathbb{P}(\Psi \mid \lambda, \mu)\mathbb{P}(d \mid \nu)\mathbb{P}(r \mid \alpha, \beta)}{\mathbb{P}(\Psi)} \tag{6.5}$$

- The `Rev` file for performing this analysis: `mcmc_BD_bears.Rev` or `mcmc_Yule_bears.Rev`.

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 6.1.

```
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)
```

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.

Run MCMC

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

```
mymcmc.burnin(generations=10000,tuningInterval=1000)
mymcmc.run(generations=50000)
```

```
mymcmc.operatorSummary()
```

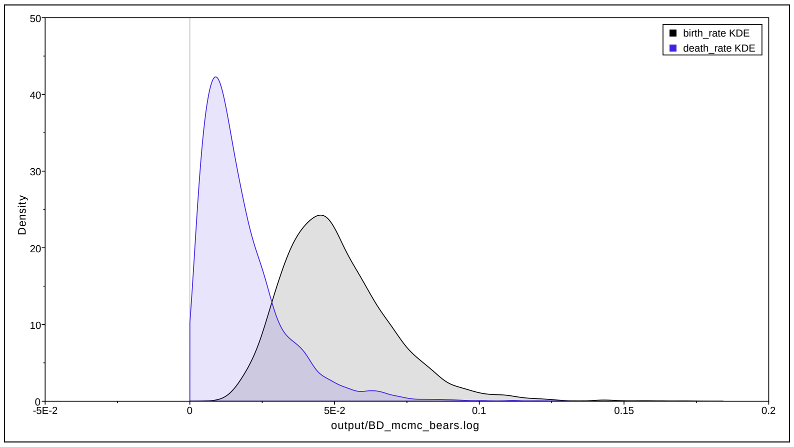


Figure 6.8: The marginal densities of **`birth_rate`** and **`death_rate`** estimated under the birth-death model in `RevBayes`.

- Visualize the MCMC samples of the birth rate and death rate parameters in Tracer.

Useful Links

- RevBayes documentation and project information: RevBayes.com
- RevBayes source: https://github.com/revbayes/revbayes
- TreePar: http://cran.r-project.org/web/packages/TreePar/index.html
- Tree Thinkers: http://treethinkers.org

Questions about this tutorial can be directed to:

- Tracy Heath (email: trayc7@gmail.com)
- Tanja Stadler (email: tanja.stadler@bsse.ethz.ch)
- Sebastian Höhna (email: sebastian.hoehna@gmail.com)

Bibliography

Aldous D. 2001. Stochastic models and descriptive statistics for phylogenetic trees, from Yule to today. Statistical Science. 16:23–34.

Aldous D, Popovic L. 2005. A critical branching process model for biodiversity. Advances in Applied Probability. 37:1094–1115.

dos Reis M, Inoue J, Hasegawa M, Asher R, Donoghue P, Yang Z. 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.

Gernhard T. 2008. The conditioned reconstructed process. Journal of Theoretical Biology. 253:769–778.

Hartmann K, Wong D, Stadler T. 2010. Sampling trees from evolutionary models. Systematic Biology. 59:465–476.

Kendall DG. 1948. On the generalized "birth-and-death" process. Annals of Mathematical Statistics. 19:1–15.

Nee S, May RM, Harvey PH. 1994. The reconstructed evolutionary process. Philosophical Transactions of the Royal Society B. 344:305–311.

Popovic L. 2004. Asymptotic genealogy of a critical branching process. Annals of Applied Probability. 14:2120–2148.

Rannala B, Yang Z. 1996. Probability distribution of molecular evolutionary trees: A new method of phylogenetic inference. Journal of Molecular Evolution. 43:304–311.

Stadler T. 2009. On incomplete sampling under birth-death models and connections to the sampling-based coalescent. Journal of Theoretical Biology. 261:58–66.

Thompson EA. 1975. Human Evolutionary Trees. Cambridge, England: Cambridge University Press.

Yang Z, Rannala B. 1997. Bayesian phylogenetic inference using DNA sequences: A Markov chain Monte Carlo method. Molecular Biology and Evolution. 14:717–724.

Yule GU. 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.

Chapter 7

Gene tree - Species tree estimation

7.1 Overview: Gene tree-species tree models

Ever since Zuckerkandl and Pauling (**?**), people have recognised 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 modelled. 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. (**?**)).

7.1.1 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 has comes from a real biological process that builds 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. 7.1). Incomplete lineage sorting will be discussed in more details in the following subsection.

Fig. 7.1 suggests that for all processes the gene tree can be seen as the product of a branching process operating inside the species tree. As a consequence, all processes are modelled as some type of birth-death process. 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 modelled with a birth-death type of model, the coalescent. All these models can be made heterogeneous, 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 allowing different parameters to different branches of the species tree, and the gene tree-species tree model could be combined to other types of models, for instance models of trait evolution.

7.1.2 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 from the species tree. In practice, this matters: for instance, one can prove that in the presence of

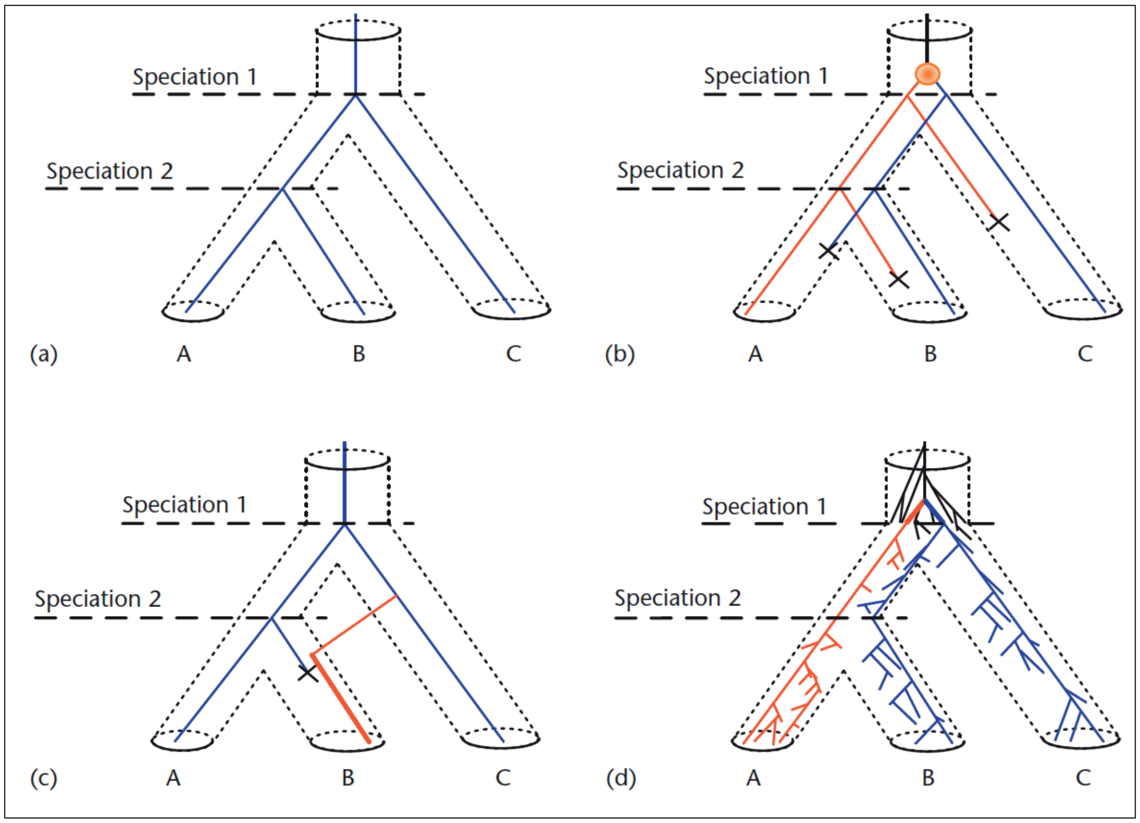


Figure 7.1: The processes of discord. The species tree is represented as a tubular structure. Gene trees are blue and red lines running along the species trees. a) A gene tree that perfectly matches the species tree. b) The gene tree and the species tree differ because of gene duplications and losses. 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 **?**.]

incomplete lineage sorting, in some particular area of the parameter space, concatenation will return an incorrect species tree. Another example might be found in prokaryotic phylogenetics, where the quest for a tree of life has been very frustrating, to the point that many doubt that one could find a meaningful species tree representing vertical descent. Recent models 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. They 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 are 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 a model of sequence evolution, models of the co-evolution between gene trees, models of trait evolution...

7.1.3 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, short of 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. 7.1d). 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 necessarily 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 (**?**), the multispecies coalescent (Fig. 7.2). This model is at the origin of most model-based approaches to reconstruct gene and species trees (**?**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 know what's before and after speciation events. The resulting subtrees are represented with the grey boxes in Fig. 7.2. 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 (alleles splitting), the coalescent model provides simple formulas for computing the probability of the gene subtree given other parameters. These subtree probabilities 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, irrespective of the fitness of this allele. 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 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.

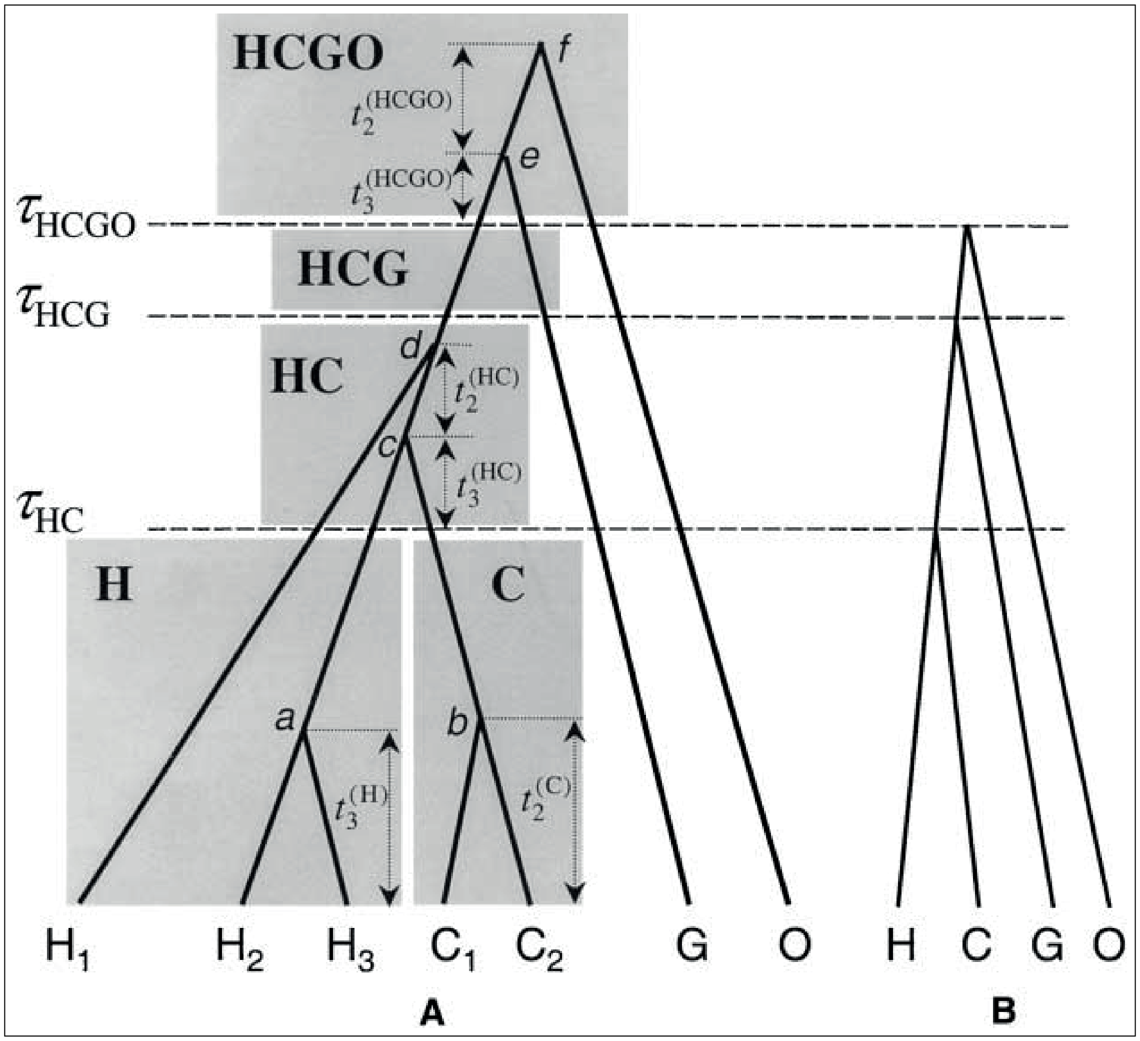


Figure 7.2: The multispecies coalescent. A) A gene tree, including 3 human alleles, 2 Chimp alleles, one Gorilla allele, and one Orang-outan allele. τ parameters are speciation times, t parameters are divergence time in the gene tree, the grey squares represent the ancestral populations, with their respeciive sizes. B) The corresponding species tree. In this model, the speciation times define minimal boundaries for allele divergence times. [Replicated from Fig. 1 in **?**.]

The exercises assume you have a working installation of RevBayes. The first exercise aims at understanding the impact of the parameters of the multispecies coalescent. The other exercises will introduce methods to reconstruct species trees and gene trees using the multispecies coalescent or closely-related models. Scripts are all placed in *tutorials/RB_GeneTreeSpeciesTree_Tutorial/RB_GeneTreeSpeciesTree_tutorial_files/RevBayes_scripts/*, in five folders. *It is highly recommended to start RevBayes from each of these folders when doing the exercises.*

First RevBayes exercise: simulating gene trees under the multispecies coalescent

1. Open RevBayes
2. Let's simulate a species tree with 10 taxa, 10 gene trees, 5 alleles per species (feel free to change these values).

```
n_species <- 10
n_genes <- 10
n_alleles <- 5

# Species names
for (i in 1:n_species) {
        s_names[i] <- "Species_"+i
}
```

3. We simulate a species tree topology according to a birth-death process with arbitrary parameter values (similar to **?**):

```
speciation ~ dnExp(10.0)
extinction ~ dnExp(10.0)
tree_height ~ dnUniform(0,1.0)
speciation.setValue(2)
extinction.setValue(0.3)
tree_height.setValue(0.8)
speciesTree ~ dnBDP(lambda=speciation, mu=extinction, origin=
   tree_height, nTaxa=n_species, names=s_names)
```

4. Then we can use the multispecies coalescent model to generate gene trees. These can be examined using Figtree or NJplot or any other tree viewer, but we can also directly compute symmetric differences between these from RevBayes. First, we simulate a set of gene trees, using a single effective population size for all branches, and after having constructed a map between species names and gene names:

```
# Build the mapping between sequence names and species names.
for (i in 1:n_species) {
        for (j in 1:n_alleles) {
                taxa[(i-1)*n_alleles+j] <- taxon(taxonName=
                   s_names[i]+"_"+j, speciesName=s_names[i])
        }
}
# Set the effective population size
Ne ~ dnGamma(shape=0.1,rate=0.1)
Ne.setValue(0.004)
# Simulate gene trees
for (i in 1:n_genes) {
   # The gene tree from the multispecies coalescent process
   # Note that if Ne had been a vector of effective population
      sizes instead of a single value,
   # allowing 1 parameter per branch of the species tree, the
      same line would work.
   geneTrees[i] ~ dnCoalMultiSpeciesConst(speciesTree=speciesTree
      , Ne=Ne, taxa=taxa)
}
```

We write the data to file, in the "dataFolder" directory.

```

# We need to save the species tree and the gene trees
write(speciesTree, filename=dataFolder+"speciesTree")

# Saving the gene trees
for (i in 1:(n_genes)) {
        write(geneTrees[i], filename=dataFolder+"geneTree_"+i+".
           tree")
}
```

5. We can compute symmetric differences between all these gene trees. The symmetric difference between two trees is the total number of partitions found in one tree but not in the other one. In our case, the maximal difference is as follows:

```
maxDiff <- 2 * (n_species*n_alleles - 2)
```

6. That will give us a reference for comparing with the values we get on our gene trees. We can build a function for computing all pairwise symmetric differences between our gene trees, and getting the mean. Caveat: in the current state of the Rev language, we have to use variable

names that have never been used before in the workspace (hence the use of "k" and "l" for loop indices instead of "i" and "j" that have been used before. Alternatively we could have done a "clear(i)" and a "clear(j)" if we really wanted to use these variable names).

```
function RealPos symDiffVector ( TimeTree[] vec ) {
        ndiff <- 1
        for (k in 1:(vec.size()-1)) {
                for (l in (k+1):vec.size()) {
                        diff[ndiff]<-symDiff (vec[k], vec[l])
                        ndiff <- ndiff+1
                }
        }
        return (mean(diff))
}


#We can then use this function on our gene trees:
symDiffVector(geneTrees)
```

7. Now to get a sense of how population size and branch lengths alter the gene tree distribution, we can relaunch the multispecies coalescent simulation (step 4) and look at the resulting gene trees after having rescaled the species tree or changed the effective population size. To do these little changes:

```
# Changing Ne:
Ne.setValue(0.08)
#Rescaling the species tree:
speciesTree.rescale(0.1)
```

Using the functions above, it is possible to look at the species tree in coalescent time units (which is very convenient). How would you do that?

Do these observations seem coherent with the multispecies coalescent presentation above?

7.2 Alternatives to the multispecies coalescent model

7.2.1 Strengths and weaknesses of the multispecies coalescent

The multispecies coalescent model is an elegant model. As we have seen above, we can easily simulate data under this model. Inference using this model, combined with a model of sequence evolution, is also possible, and when it works, is very informative. Not only can we get a dated species tree and dated gene trees from the multispecies coalescent, we can also get extant and ancestral effective population sizes (**?**Heled and Drummond 2010). However inferring many parameters is difficult, and convergence can be difficult to reach with such models. In particular, the strong interdependence that exists between the gene trees and the species tree makes it easy for algorithms to fall into local maxima. As a consequence, there are ongoing efforts to develop methods for which inference would be easier, albeit at the cost of approximations and simplifications.

7.2.2 Alternatives to the multispecies coalescent

An easy way to simplify the problem is to consider that parts of it are already solved. For instance, several approaches assume that rooted gene trees are available. The problem then becomes markedly easier, but inference is then highly dependent on the quality of the input gene trees. Often, such methods also make other simplifying assumptions, and e.g. do not try to estimate separately time and effective population sizes, but instead directly work with coalescent time units. These methods usually are much faster than the multispecies coalescent, should be more robust against local maxima, but are less ambitious about the amount of information they can get from the data, and are can be sensitive to the quality of the input gene trees.

Another approach is to use methods that mathematically bypass estimating gene trees altogether. To our knowledge, there are three such approaches: SNAPP (**?**), POMO (**?**), and SVDQuartets (**?**). They differ in the way the algorithms work, but they are all based on the same idea, which is integrating out gene trees. To achieve that they extend the model of sequence evolution, which usually models substitution events, to also model population-level processes. In RevBayes, so far only the POMO model has been implemented and will be discussed in the following part.

7.2.3 The POMO model

POMO models allele frequency changes along with mutations with a single transition matrix. It extends the usual 4×4 DNA substitution matrix to incorporate polymorphic states. In doing so, it makes a first important approximation: it only considers biallelic states. For instance, it considers sites at which either an A or a C is found in a population, but it won't consider sites at which 3 different states, e.g. A, C, T, are observed in a population. Then it makes a second approximation, which introduces a single virtual population size in lieu of the branchwise effective population sizes. This virtual population size is not inferred, but is fixed to some low number. In practice, **?** consider that a virtual population size of 10 individuals should produce good results. This virtual population size directly constrains the types of polymorphic states that can be considered. With 10 individuals for instance, only frequencies such as $(100\%A)$, $(10\%A, 90\%C)$, $(20\%A, 80\%C)$, $(30\%A, 70\%C)$, ..., $(90\%A, 10\%C)$, $(100\%C)$ can be considered. The POMO matrix models transitions among all these states, polymorphic ones as well as monomorphic ones, and has a size that depends on the virtual population size. For instance, with a virtual population size of 10 individuals, the POMO square matrix contains 58 rows: 4 monomorphic states, plus 6 types of biallelic states (AC, AG, AT, CG, CT, GT) times 9 frequencies. Additional assumptions of the POMO model include total independence among sites (no linkage among sites), and absence of mutations in biallelic states: transitions among biallelic states or from biallelic states to monoallelic states only occur through population-level changes in allele frequencies, not through mutation of one allele into another. Mutations only occur to transit from a monoallelic state to a biallelic state.

The POMO model therefore makes several approximations to avoid estimating gene trees. Fewer parameters need to be estimated, as neither gene trees nor population sizes are estimated, but other parameters can be introduced into the model. For instance **?** estimate 4 base fitness parameters, which they use to model GC-biased gene conversion, which tends to increase the GC content of recombining sequences. In the RevBayes implementation of POMO, base fitnesses can be estimated as well.

7.3 Inference using the multispecies coalescent, the POMO model, and basic concatenation

In this section we will perform inference using the multispecies coalescent, the POMO model, and concatenation. Depending on your machine and on the size of the data, successful inference may take some time. If this tutorial is done in a classroom environment, it may be wise to convene with friends that one tries the multispecies coalescent model while another one tries the POMO model, and the third one tries the concatenation approach, and that results will be shared. In case you want to compare the results between machines or between RevBayes sessions, you have to make sure that you are using the same random seed in all sessions. This is easily done by setting the seed using

```
seed(x, y)
```

where x and y are two integer numbers of your choice.

First we will simulate data, and then we will perform inference using the three different approaches.

7.3.1 Simulating data

We are going to do inference on simulated data, as produced during the previous section. However in addition to the species tree and the gene trees, we will be producing sequence data. Below is the RevBayes code necessary to simulate all the data. As before, parameters of the simulation can be changed as seems fit, but here we chose parameters that resemble **?**.

WARNING: The following script may be less up to date than the simulation scripts named "UnderstandingMultiSpeciesCoalescent/MultiSpeciesCoalescentSimulatingTreesAndAlignments.Rev".

First we can set the folder in which we will save the output of our work.

```
dataFolder<-"/PATH/TO/A/FOLDER/WHERE/TO/SAVE/THE/OUTPUT"
setwd(dataFolder)
```

Let's simulate a species tree with 6 taxa, 10 gene trees, 5 alleles per species, and along each gene tree one gene alignment 200 bases long. It's a small data set, designed for manageable inference during a tutorial, but patient users may want to simulate a larger data set.

```
n_species <- 10
n_genes <- 10
n_alleles <- 5
n_sites <- 100
n_branches <- 2 * n_species - 3 # number of branches in a rooted tree
```

The species-tree birth-death model:

```
# Species names
for (i in 1:n_species) {
        s_names[i] <- "Species_"+i
}
speciation ~ dnExp(10.0)
extinction ~ dnExp(10.0)
tree_height ~ dnUniform(0,1.0)
speciation.setValue(2)
extinction.setValue(0.3)
tree_height.setValue(0.008)
# Species tree from birth-death process
speciesTree ~ dnBDP(lambda=speciation, mu=extinction, origin=
   tree_height, nTaxa=n_species, names=s_names)
# Making a backup for future reference:
trueSpeciesTree <- speciesTree
```

The gene-tree multispecies coalescent model

```
# Build the mapping between sequence names and species names.
for (i in 1:n_species) {
        for (j in 1:n_alleles) {
                taxa[(i-1)*n_alleles+j] <- taxon(taxonName=s_names[i
                   ]+"_"+j, speciesName=s_names[i])
        }
}
# Set the effective population size
Ne ~ dnGamma(shape=0.5,rate=0.5)
Ne.setValue(0.004)
# Simulate gene trees
for (i in 1:n_genes) {
   # The gene tree from the multispecies coalescent process
   # Note that if Ne had been a vector of effective population sizes,
   # allowing 1 parameter per branch of the species tree, the same
      line would work.
   geneTrees[i] ~ dnCoalMultiSpeciesConst(speciesTree=speciesTree, Ne
      =Ne, taxa=taxa)
}
# Making a backup for future reference:
trueGeneTrees <- geneTrees
trueNe <- Ne
```

Substitution models will all be based on the GTR model. However, the parameters of the models change from one gene family to the next.

```
for (i in 1:n_genes) {
  er_prior[i] <- v(1,1,1,1,1,1)
  er[i] ~ dnDirichlet(er_prior[i])
  sf_prior[i] <- v(1,1,1,1)
  sf[i] ~ dnDirichlet(sf_prior[i])
  Q[i] := fnGTR(er[i],sf[i])
}
# Making a backup for future reference:
for (i in 1:n_genes) {
        trueEr[i] <- er[i]
        trueSf[i] <- sf[i]
}
```

Then we assume a strict clock Model, but of course we could assume a relaxed clock. Each gene family can have its own rate of evolution, drawn from an exponential distribution.

```
for (i in 1:n_genes) {
  familyRates[i] ~ dnExp(1.0)
}
# Making a backup for future reference:
for (i in 1:n_genes) {
  trueFamilyRates[i] <- familyRates[i]
}
```

We add a model of among-site rate variation, handled by a discretized Gamma distribution, one for each gene family:

```
for (i in 1:n_genes) {
  shape_prior[i] <- 0.05
  shape[i] ~ dnExponential( shape_prior[i] )
  norm_gamma_rates[i] := fnDiscretizeGamma( shape[i], shape[i], 4,
     false )
}
```

Finally we link it all using the PhyloCTMC model, which simulates gene alignments:

```
for (i in 1:n_genes) {
  alns[i] ~ dnPhyloCTMC(tree=geneTrees[i], Q=Q[i],  branchRates=
     familyRates[i], siteRates=norm_gamma_rates[i], nSites=n_sites,
     type="DNA")
}
```

We can then save the simulated data to the folder chosen at the beginning of this script. We need to save the species tree, the gene trees, and the gene alignments

```
write(speciesTree, filename="speciesTree")
for (i in 1:n_genes) {
        write(geneTrees[i], filename="geneTree_"+i)
}
for (i in 1:n_genes) {
        writeFasta(alns[i], filename="alignment_"+i+".fasta")
}
```

7.3.2 Inference using the multispecies coalescent

WARNING: The following script may be less up to date than the simulation script named "MultiSpeciesCoalescentWithSequences/MultiSpeciesCoalescentWithSequencesMCMC.Rev". In the following we perform inference from the alignments, but we could also do inference directly from the gene trees, as can be seen in the script "MultiSpeciesCoalescent/MultiSpeciesCoalescentMCMC.Rev".

Now that we have simulated data, we can run inference under the multispecies coalescent, using the sequences as input data.

First we clamp the alignments:

```
for (i in 1:n_genes) {
        alns[i].clamp(alns[i])
}
```

Then we change the starting values, because we want to start from random values, not the values used in the simulation.

```
# Redrawing the parameters of the birth-death process:
speciation.redraw()
extinction.redraw()
tree_height.redraw()
# Redrawing the species tree:
speciesTree.redraw()
# Redrawing the parameter Ne:
Ne.redraw()
# Redrawing the gene trees:
for (i in 1:n_genes) {
   geneTrees[i].redraw()
}
# Redrawing the parameters of the substitution models:
for (i in 1:n_genes) {
  er[i].redraw()
  sf[i].redraw()
}
# Idem for the family-wise rates of sequence evolution:
for (i in 1:n_genes) {
  familyRates[i].redraw()
}
# Idem for the family-wise across-site rate variation parameters:
for (i in 1:n_genes) {
  shape[i].redraw()
}
```

We need to set up moves for the birth-death parameters, the species tree topology, the gene tree topologies, the parameter Ne, the parameters of the substitution models, the rates on the gene trees.

```
moveIndex <- 0
# moves for the birth-death parameters
moves[moveIndex++] <- mvScale(speciation,1,true,1.0) # In the
   revLanguage, table indices start at 1
moves[moveIndex++] <- mvScale(extinction,1,true,1.0)
moves[moveIndex++] <- mvSlide(tree_height,delta=1.0,true,2.0)
# moves on the tree topology and node ages
moves[moveIndex++] <- mvNNI(speciesTree, 1.0)
moves[moveIndex++] <- mvFNPR(speciesTree, 1.0)
moves[moveIndex++] <- mvSubtreeScale(speciesTree, 5.0)
moves[moveIndex++] <- mvTreeScale(speciesTree, delta=1.0, tune=true,
   weight=3.0)
moves[moveIndex++] <- mvNodeTimeSlideUniform(speciesTree, 10.0)
moves[moveIndex++] <- mvRootTimeSlide(speciesTree, 1.0, true, 3.0)
# moves on the gene tree topologies and node ages
for (i in 1:n_genes) {
   moves[moveIndex++] <- mvNNI(geneTrees[i], 1.0)
   moves[moveIndex++] <- mvFNPR(geneTrees[i], 1.0)
   moves[moveIndex++] <- mvSubtreeScale(geneTrees[i], 5.0)
   moves[moveIndex++] <- mvTreeScale(geneTrees[i], delta=1.0, tune=
      true, weight=3.0)
   moves[moveIndex++] <- mvNodeTimeSlideUniform(geneTrees[i], 10.0)
   moves[moveIndex++] <- mvRootTimeSlide(geneTrees[i], 1.0, true,
      3.0)
}
# move on Ne, the effective population size
moves[moveIndex++] <- mvScale(Ne,1,true,1.0)
# moves on the parameters of the substitution models
for (i in 1:n_genes) {
  moves[moveIndex++] <- mvSimplexElementScale(er[i], alpha=10, tune=
     true, weight=3)
  moves[moveIndex++] <- mvSimplexElementScale(sf[i], alpha=10, tune=
     true, weight=2)
}
# moves on the family-wise rates
for (i in 1:n_genes) {
  moves[moveIndex++] <- mvScale(familyRates[i], lambda=0.8, tune=true
     , weight=3.0)
}
# moves on the across-sites rate variation parameters:
for (i in 1:n_genes) {
  moves[moveIndex++] <- mvScale(shape[i], lambda=0.8, tune=true,
     weight=3.0)
}
```

Then we define a few monitors to keep track of how things go. First, basic monitors:

```
mntrIndex <- 0
# One monitor to backup the parameters, in case we want to stop and
   restart the analysis:
monitors[mntrIndex++] <- mnModel(filename="
   multispeciesCoalescent_clock.log",printgen=10, separator = " ")
# One monitor to print the species trees sampled in the course of the
    MCMC:
monitors[mntrIndex++] <- mnFile(filename="
   multispeciesCoalescent_clock_species.trees",printgen=10, separator
    = "        ", speciesTree)
# One monitor for each gene family tree:
for (i in 1:n_genes) {
   monitors[mntrIndex++] <- mnFile(filename="
      multispeciesCoalescent_clock_gene_"+ i +".trees",printgen=10,
      separator = "        ", geneTrees[i])
}
```

We will also compare our reconstructed parameters (trees or other variables) to the true values that were used in the simulation. First, examining the species tree and the gene trees.

```
distSpeciesTree := symDiff (trueSpeciesTree, speciesTree)
for (i in 1:n_genes) {
        distGeneTree[i] := symDiff (trueGeneTrees[i], geneTrees[i])
}
# We get one average value for all gene trees:
meanDistGeneTree := mean(distGeneTree)
```

We can also look at the values of some other variables:

```
# For Ne:
distNe := Ne - trueNe
# For equilibrium values of the GTR matrices, we want one index of
   how far we are.
# To achieve this we need to write some functions:
clear(i)
distEqFreq := diffVectorsOfVectors(trueSf, sf)
function RealPos diffVectors ( Real[] xvec, Real[] yvec ) {
        DI <- 0.0
        for (i in 1:xvec.size()) {
                DI <- DI + (xvec[i] - yvec[i])*(xvec[i] - yvec[i])
        }
        return DI
}
function RealPos diffVectorsOfVectors ( Real[][] xvecvec, Real[][]
   yvecvec ) {
        DA <- 0.0
        for (i in 1:xvecvec.size()) {
                DA <- DA + diffVectors(xvecvec[i], yvecvec[i])
        }
        return DA
}
function RealPos diffVectorsOfVectors ( Simplex[] xvecvec, Simplex[]
   yvecvec ) {
        DA <- 0.0
        for (i in 1:xvecvec.size()) {
                DA <- DA + diffVectors(xvecvec[i], yvecvec[i])
        }
        return xvecvec[2]
}
# Same thing for exchangeability parameters:
distExchange := diffVectorsOfVectors(trueEr, er)
monitors[mntrIndex++] <- mnScreen(printgen=10, distExchange)
# Same thing for the family-wise rates
distRates := diffVectors(trueFamilyRates, familyRates)
monitors[mntrIndex++] <- mnScreen(printgen=10, distRates)
# We can use one monitor that will output on the screen one parameter
   , Ne, distNe, distSpeciesTree, meanDistGeneTree, and distEqFreq:
monitors[mntrIndex++] <- mnScreen(printgen=10, Ne, distNe,
   distSpeciesTree, meanDistGeneTree, distEqFreq)
# We could do similar things for the few remaining parameters, but
   really I think that's enough.
```

We then define our model and the mcmc object. We can use any node of our model as a handle, here we choose to use the species tree.

```
mymodel <- model(speciesTree)
# We create the MCMC object
mymcmc <- mcmc(mymodel, monitors, moves)
```

We finally launch the analysis.

```
# If we want to specify the amount of burnin:
# mymcmc.burnin(generations=200,tuningInterval=100)
mymcmc.run(generations=400)
```

After the analysis, we analyze the output. We analyze the tree trace as saved in one of the output files.

```
# Let us start by reading in the tree trace
 treetrace <- readTreeTrace("multispeciesCoalescent_clock_species.
    trees")
# and get the summary of the tree trace
 treetrace.summarize()

# We output the Maximum A Posteriori tree
 mapTree(treetrace,"primates_clock_MAP.tre")
```

Of course, we could do the same analysis with each of the gene trees, possibly using a "for" loop in the rev language.

7.3.3 Inference using the POMO model

WARNING: The following script may be less up to date than the simulation script named "MultiSpeciesCoalescentAndPomo/MultiSpeciesCoalescentPomoMCMC.Rev".

We run the POMO model on the same data set simulated under the multispecies coalescent.

First, we need to prepare the data for analysis using POMO: concatenating the alignments, then transforming the state space from 4 bases to the POMO state space. We arbitrarily decide to use a virtual population size of size 10, as suggested in **?**.

```
# First, concatenating the alignments:
concatenate<-alns[1]
for (i in 2:n_genes) {
        concatenate <- concatenate + alns[i]
}

virtual_population_size <- 10

# Now, we need to convert the DNA alignment into an alignment in the
   correct POMO state space.

pomo_concatenate <- pomoStateConvert(concatenate,
   virtual_population_size, taxa)
concatenate_n_sites <- pomo_concatenate.nchar()[1]
```

Now we have the data for doing inference using the POMO model. To define a POMO model, one needs several components. First, one needs to define a transition matrix for DNA mutations. Here we are going to use a GTR matrix.

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

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

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

Second, one can have different fitnesses for A, C, G, T. Here, we are going to assume that all 4 bases have the same fitness.

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

Now we have all the elements to build a POMO matrix to model the evolution of a population of alleles along a species tree.

```
P := fnPomo(Q, base_fitnesses, virtual_population_size)
```

We also need to define root frequencies for all the states. To do that we need two variables:

```
# First, the proportion of polymorphic sites at the root
root_polymorphism_proportion ~ dnBeta(alpha=1,beta=1)

# Second, one needs the root frequencies (we could use those of the
   GTR matrix but choose to have a non-stationary model):
root_base_frequencies ~ dnDirichlet(sf_prior)
```

Now we have all the elements to construct root frequencies for all the states:

```
root_frequencies := pomoRF (root_base_frequencies,
   root_polymorphism_proportion, Q, virtual_population_size)
#simplex_root_frequencies := simplex(root_frequencies[1],
   root_frequencies[2], root_frequencies[3], root_frequencies[4])
```

Adding an across site rate variation model, the usual Gamma distribution discretized in 4 categories:

```
shape_prior <- 0.05
shape ~ dnExponential( shape_prior )
norm_gamma_rates := fnDiscretizeGamma( shape, shape, 4, false )

# We do not assume variation among branches in rates
branch_rates <- 1.0

# We do not assume a proportion of invariant
p_inv ~ dnBeta(alpha=1,beta=1)
p_inv.setValue(0.000001)
```

Combining it all with the PhyloCTMC Model and clamp the model to the data:

```
aln ~ dnPhyloCTMC(tree=speciesTree, Q=P, rootFreq=root_frequencies,
   branchRates=branch_rates, siteRates=norm_gamma_rates, pInv=p_inv,
   nSites=concatenate_n_sites, type="Standard")

aln.clamp(pomo_concatenate)
```

Changing the starting values, we want to start from random values, not the values used in the simulation:

```
# Redrawing the parameters of the birth-death process:
speciation.redraw()
extinction.redraw()
tree_height.redraw()

# Redrawing the species tree:
speciesTree.redraw()
```

Then we need to set up moves for the birth-death parameters, the species tree topology, the gene tree topologies, the parameter Ne, the parameters of the substitution models, the rates on the gene trees.

```
moveIndex <- 0

# moves for the birth-death parameters
moves[moveIndex++] <- mvScale(speciation,1,true,1.0) # In the
   revLanguage, table indices start at 1
moves[moveIndex++] <- mvScale(extinction,1,true,1.0)
moves[moveIndex++] <- mvSlide(tree_height,delta=1.0,true,2.0)


# moves on the tree topology and node ages
moves[moveIndex++] <- mvNNI(speciesTree, 1.0)
moves[moveIndex++] <- mvFNPR(speciesTree, 1.0)
moves[moveIndex++] <- mvSubtreeScale(speciesTree, 5.0)
moves[moveIndex++] <- mvTreeScale(speciesTree, delta=1.0, tune=true,
   weight=3.0)
moves[moveIndex++] <- mvNodeTimeSlideUniform(speciesTree, 10.0)
moves[moveIndex++] <- mvRootTimeSlide(speciesTree, 1.0, true, 3.0)

# moves on the parameters of the substitution model
moves[moveIndex++] <- mvSimplexElementScale(er, alpha=10, tune=true,
   weight=3)
moves[moveIndex++] <- mvSimplexElementScale(sf, alpha=10, tune=true,
   weight=2)

# moves on the 4 fitness values
# SH: These are not stochastic nodes, so moves cannot operate on them
   !
#moves[moveIndex++] <- mvScale(base_fitnesses[1],1,true,1.0)
#moves[moveIndex++] <- mvScale(base_fitnesses[2],1,true,1.0)
#moves[moveIndex++] <- mvScale(base_fitnesses[3],1,true,1.0)
#moves[moveIndex++] <- mvScale(base_fitnesses[4],1,true,1.0)

# moves on the parameters of the root frequencies
moves[moveIndex++] <- mvSimplexElementScale(root_base_frequencies,
   alpha=10, tune=true, weight=3)
#moves[moveIndex++] <- mvScale(root_polymorphism_proportion, lambda
   =10, tune=true, weight=2)
moves[moveIndex++] <- mvSlide(root_polymorphism_proportion, delta=10,
    tune=true, weight=2)

# moves on the across-sites rate variation parameter:
moves[moveIndex++] <- mvScale(shape, lambda=0.8, tune=true, weight
   =3.0)
```

Then we define a few monitors to keep track of how things go:

```
mntrIndex <- 0

# One monitor to backup the parameters, in case we want to stop and
   restart the analysis:
monitors[mntrIndex++] <- mnModel(filename="pomo_clock.log",printgen
   =10, separator = "    ")

# One monitor to print the species trees sampled in the course of the
    MCMC:
monitors[mntrIndex++] <- mnFile(filename="pomo_clock_species.trees",
   printgen=10, separator = "   ", speciesTree)

# We also want to monitor how far we are from the true values, which
   we have because we rely on simulations.
# First, we can compute the distance between the reconstructed and
   the true species tree:
distSpeciesTree := symDiff (trueSpeciesTree, speciesTree)


# We can use one monitor that will output on the screen one parameter
   , Ne, distNe, distSpeciesTree, meanDistGeneTree, and distEqFreq:
monitors[mntrIndex++] <- mnScreen(printgen=10, distSpeciesTree)
```

We define our model:

```
# First we need to get rid of several variables that are not part of
   the Pomo model.
clear(alns)
clear(geneTrees)
clear(Ne)
clear(familyRates)
clear(trueEr)
clear(trueFamilyRates)
clear(trueGeneTrees)
clear(trueNe)
clear(trueSf)

# We can use any node of our model as a handle, here we choose to use
    the species tree.

mymodel <- model(speciesTree)
```

We create and run the MCMC object

```
mymcmc <- mcmc(mymodel, monitors, moves)
# mymcmc.burnin(generations=200,tuningInterval=100)
mymcmc.run(generations=40000)
```

7.3.4 Inference using basic concatenation

WARNING: The following script may be less up to date than the simulation scripts named "MultiSpeciesCoalescentAndConcatenation/MultiSpeciesCoalescentConcatenationMCMC.Rev".

We can also concatenate all alignments and run a partitioned GTR model on it.

```
# Birth-Death process priors.
speciation ~ dnExp(10.0)
extinction ~ dnExp(10.0)
tree_height ~ dnUniform(0.0,100.0)
speciation.setValue(2)
extinction.setValue(0.3)

# Species tree from birth-death process
#We assume 1000,000 generations
speciesTree ~ dnBDP(lambda=speciation, mu=extinction, rootAge=
   tree_height, nTaxa=n_species, names=s_names)
```

Second, one needs to define a transition matrix for DNA mutations, and we use a GTR matrix for that.

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

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

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

Adding an across site rate variation model, the usual Gamma distribution discretized in 4 categories.

```
shape_prior <- 0.05
shape ~ dnExponential( shape_prior )
norm_gamma_rates := fnDiscretizeGamma( shape, shape, 4, false )

# We do not assume variation among branches in rates
branch_rates <- 1.0

# We do not assume a proportion of invariant
p_inv ~ dnBeta(alpha=1,beta=1)
p_inv.setValue(0.000001)
```

To link all the parts of the model together we use the phyloCTMC object and clamp it to the concatenate.

```
aln ~ dnPhyloCTMC(tree=speciesTree, Q=Q, branchRates=branch_rates,
   siteRates=norm_gamma_rates, pInv=p_inv, type="DNA")

aln.clamp( concatenate )
```

Then we define the moves.

```
moveIndex <- 0

# moves for the birth-death parameters
moves[moveIndex++] <- mvScale(speciation,lambda=1,tune=true,weight
   =1.0) # In the revLanguage, table indices start at 1
moves[moveIndex++] <- mvScale(extinction,1,true,1.0)
moves[moveIndex++] <- mvSlide(tree_height,delta=1.0,true,2.0)

# moves on the tree topology and node ages
moves[moveIndex++] <- mvNNI(speciesTree, 2.0)
moves[moveIndex++] <- mvNarrow(speciesTree, 5.0)
moves[moveIndex++] <- mvFNPR(speciesTree, 2.0)
moves[moveIndex++] <- mvSubtreeScale(speciesTree, 5.0)
moves[moveIndex++] <- mvTreeScale(speciesTree, tree_height, delta
   =1.0, tune=true, weight=3.0)
moves[moveIndex++] <- mvNodeTimeSlideUniform(speciesTree, 10.0)

# moves on the parameters of the substitution model
moves[moveIndex++] <- mvSimplexElementScale(er, alpha=10, tune=true,
   weight=3)
moves[moveIndex++] <- mvSimplexElementScale(sf, alpha=10, tune=true,
   weight=2)

# moves on the across-sites rate variation parameter:
moves[moveIndex++] <- mvScale(shape, lambda=0.8, tune=true, weight
   =3.0)
```

We define a few monitors to keep track of how things go.

```
mntrIndex <- 0

# One monitor to backup the parameters, in case we want to stop and
   restart the analysis:
monitors[mntrIndex++] <- mnModel(filename=dataFolder+"
   concatenation_clock.log",printgen=10, separator = "        ")

# One monitor to print the species trees sampled in the course of the
    MCMC:
monitors[mntrIndex++] <- mnFile(filename=dataFolder+"
   concatenation_clock_species.trees",printgen=10, separator = "
         ", speciesTree)

# We also want to monitor how far we are from the true values, which
   we have because we rely on simulations.
# First, we can compute the distance between the reconstructed and
   the true species tree:
distSpeciesTree := symDiff (trueSpeciesTree, speciesTree)


# We can use one monitor that will output on the screen one parameter
   , Ne, distNe, distSpeciesTree:
monitors[mntrIndex++] <- mnScreen(printgen=10, distSpeciesTree)
```

We define the model and launch the MCMC.

```
mymodel <- model(speciesTree)

# We create the MCMC object
mymcmc <- mcmc(mymodel, monitors, moves)

#mymcmc.burnin(generations=200,tuningInterval=100)
mymcmc.run(generations=4000)
```

7.4 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`

7.5 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 (**?**). 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?

7.6 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)

Version dated: December 16, 2014

Bibliography

Heled J, Drummond A. 2010. Bayesian inference of species trees from multilocus data. Molecular Biology and Evolution. 27:570.

Chapter 8

Biogeography

8.1 Introduction

This lab describes how to perform Bayesian inference of historical biogeography using RevBayes. The analysis jointly infers the posterior range evolution parameters and ancestral ranges using the data augmentation approach described in Landis et al. (2013). To examine the posterior, we'll use Python scripts, Tracer by Andrew Rambaut, to summarize MCMC output files, and Phylowood, a Javascript web service that animates and filters phylogenetic biogeography reconstructions (Landis and Bedford 2014).

Outline

I. Introduction
- a) Workspace setup
- b) Range evolution models and data augmentation

II. DEC in RevBayes
- a) Tree, range, and geographic input
- b) DEC as a graphical model
- c) Monitors, moves, and MCMC
- d) Model selection with Bayes factors

III. Output and Analysis
- a) MCMC output in Tracer
- b) Ancestral range output
- c) Animate ancestral ranges

8.1.1 Handy links for this lab

RevBayes	`https://github.com/revbayes/revbayes`
Lab zip file	`https://github.com/revbayes/revbayes/.RB_biogeo_files.zip`
Phylowood software	`http://mlandis.github.io/phylowood`
Phylowood manual	`https://github.com/mlandis/phylowood/wiki`
Tracer	`http://tree.bio.ed.ac.uk/software/tracer`

8.1.2 Setting up your workspace

The practical part of the lab will analyze a small dataset of 19 taxa distributed over 4 biogeographic areas. Parts of the lab will require entering terminal commands, which will assume you are using the Unix shell `bash`. Just ask for help if the commands don't seem to work.

8.2 Model and method

This section contains a brief description of the data, model, parameters, and method used in BayArea.

First, we define the range for taxon i as the bit vector X_i, where $X_{i,j} = 1$ if the taxon is present in area j and $X_{i,j} = 0$ if the taxon is absent. Each taxon range is a bit vector of length N areas. For example, if taxon B is present only in areas 2 and 3 out of $N = 3$ areas, its range is represented as $X_B = (0, 1, 1)$, which is translated to the bit string $X_B = 011$ for short. The data matrix, $\mathbf{X}$, is analogous to a multiple sequence alignment where each element in the data matrix reports a discrete value for a homologous character shared by all taxa at column j.

Next, we need a model of range evolution. 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. This technique of matrix exponentiation 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 range evolution events into the allowed character transitions of $\mathbf{Q}$ and parameterize the events so that we may infer their relative importance to generating our observed ranges. We'll take a simple model of range expansion (e.g. $011 \rightarrow 111$) and range contraction (e.g. $011 \rightarrow 001$). (Range expansion may also be referred to as dispersal or area gain and range contraction as extirpation or area loss.) The rates in the transition matrix for three areas might appear as

$$\mathbf{Q} = \begin{array}{c|cccccccc}
 & 000 & 001 & 010 & 011 & 100 & 101 & 110 & 111 \\
\hline
000 & - & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
001 & \lambda_0 & - & 0 & \lambda_1 & 0 & \lambda_1 & 0 & 0 \\
010 & \lambda_0 & 0 & - & \lambda_1 & 0 & 0 & \lambda_1 & 0 \\
011 & 0 & \lambda_0 & \lambda_0 & - & 0 & 0 & 0 & \lambda_1 \\
100 & \lambda_0 & 0 & 0 & 0 & - & \lambda_1 & \lambda_1 & 0 \\
101 & 0 & \lambda_0 & 0 & 0 & \lambda_0 & - & 0 & \lambda_1 \\
110 & 0 & 0 & \lambda_0 & 0 & \lambda_0 & 0 & - & \lambda_1 \\
111 & 0 & 0 & 0 & \lambda_0 & 0 & \lambda_0 & \lambda_0 & -
\end{array},$$

where λ_1 and λ_0 are the rates of area gain and loss, respectively. This matrix can be represented compactly as the rate function

$$q_{\mathbf{y},\mathbf{z}}^{(a)} = \begin{cases} \lambda_0 & \text{if } z_a = 0 \\ \lambda_1 & \text{if } z_a = 1 \\ 0 & \mathbf{y} \text{ and } \mathbf{z} \text{ differ in more than one area} \end{cases}.$$

where $\mathbf{y}$ and $\mathbf{z}$ are the "from" and "to" ranges and a is the area that changes. For example, $q_{011,111}^{(1)}$ is the rate of range expansion for $011 \rightarrow 111$ to gain area 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.

Lastly, 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

$$q_{\mathbf{y},\mathbf{z}}^{(a)} = \begin{cases} \lambda_0 & \text{if } z_a = 0 \\ \lambda_1 \eta(\mathbf{y},\mathbf{z},a,\beta) & \text{if } z_a = 1 \\ 0 & \mathbf{y} = 00...0 \\ 0 & \mathbf{y} \text{ and } \mathbf{z} \text{ differ in more than one area} \end{cases}.$$

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.

In addition to dispersal and extinction, the DEC model allows cladogenic range evolution events. For each internal node in the tree, one of four cladogenic events can occur: narrow sympatry, subset sympatry, allopatry, and wide sympatry. Say the range of a species approaching an internal node, i.e. that is about to speciate, is 1000. Since the species range is size one, this always results in narrow sympatry, where both daughter lineages inherit the ancestral species range. Now suppose the ancestral range is 1110. Under subset sympatric cladogenesis, one lineage identically inherits the ancestral species range, 1110, while the other lineage inherits only a single area, e.g. 0100. Under allopatric cladogenesis, the ancestral range is split evenly among daughter lineages, e.g. one lineage may inherit 1100 and the other inherit 0010. Both daughter lineages inherit the entire ancestral species range following widespread sympatric cladogenic events. For a general description of state transitions for cladogenic events, see Matzke (2013).

The DEC model ignores speciation events hidden by extinction or incomplete taxon sampling and assigns equal probabilities to all cladogenic event types. Since a range of size zero typically means the species is exinct, the probability of cladogenic events would ideally be linked to a birth-death process, as it is in the GeoSSE model (Goldberg, Lancaster and Ree 2011). Unfortunately, since this sort of model scales poorly, DEC models remain the only option when the geography has more than two or three areas.

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 1024×1024, which approaches the largest size matrix that can be exponentiated in a practical amount of time. This is problematic, meaning we need an alternative method to integrate over historical range evolution events.

You may wonder why matrix exponentiation works fine for molecular substitution models and large multiple sequence alignments. Molecular substitution models typically assume each site in the multiple sequence alignment evolves independently, which may be justified because recombination degrades linkage disequilibrium over geological timescales. Conveniently, this keeps $\mathbf{Q}$ small even for datasets with many sites.

Remember matrix exponentiation integrates over all *unobserved* transition events during time t. The likelihood of beginning in character i and ending in character 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 an adaptation of the data-augmented phylogenetic method first described by Robinson et al. (2003), and was first applied to tertiary structure-dependent evolution of protein-coding nucleotide sequences.

This is the strategy we will use to infer the posterior distribution approximated is
$\text{Prob}(\mathbf{X}_{aug}, \theta \mid \mathbf{X}_{obs}, T, M)$, 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. Ancestral range reconstructions are often of primary interest in phylogenetic biogeographic analyses, which are generated with support values as a by-product of the MCMC analysis.

The rest of this tutorial will describe how to assemble the input, run the analysis, assess the output, and visualize the results.

8.3 Input

For this tutorial, we'll use a dataset for 19 species of *Psychotria* whose range spans the Hawaiian archipelago. The dataset was originally reported in Nepokroeff et al. (2003) and analyzed using the maximum likelihood method, LAGRANGE, by Ree and Smith (2008). We'll use this dataset for three reasons. First, it is relatively small, meaning we can produce results quickly. Second, the Hawaiian archipelago can be broken into naturally discrete areas and has a well-characterized geographical history that is uncomplicated to model. Third, it has previously been analyzed, which provides some basis for comparison to other methods. To simplify things, this model will assume we live in a gentler world where presence-absence characters are known without error.

For larger datasets to play with, see `input/sim_aus_50tip_33area.*` for a simulated model where all cladogenic events were sympatric (wide and narrow only).

8.3.1 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;

Begin trees;
        TREE tree1 = ((((((((P_hawaiiensis_WaikamoiL1:0.9656850499,
        P_mauiensis_Eke:0.9656850499):0.7086257935,(P_fauriei2:1.23
        0218511,P_hathewayi_1:1.230218511):0.4440923324):0.17671155
        ...
        89):0.4630447802,P_hexandra_Oahu:2.826939991):2.372081244);
End;
```

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 range data, while columns give in which areas each taxon is present (1) or absent (0). For example, taxon `P_hexandra_Oahu` is present only in area 2.

The `trees` block gives the tree describing the shared ancestry of the study species. Because range evolution occurs in units of geological time, the analysis in this tutorial requires a high-quality time-calibrated phylogeny. This typically requires a multiple sequence alignment over several loci plus fossils for calibration. Since this data availability is often the limiting factor for which taxa to include for your analysis, it is best to produce the phylogeny first. Only afterwards should you begin assembling data for your data matrix. If your phylogeny cannot be calibrated (e.g. it has no fossils) your best alternative is to proceed with a time tree resulting from a divergence time estimation analysis. For this tutorial, the phylogeny is assumed to contain no uncertainty.

8.3.2 Atlas file

The geography used in this tutorial represents the Hawaiian archipelago. Beneath Hawaii, currently the largest and youngest island, is a volcanic hotspot that periodically creates new islands. The ages of these islands are fairly well known, meaning we can model range availability as a function of time. Following Ree and Smith (2008), we will lump groups of smaller islands into single areas to simplify the analysis, leaving us with four areas: Hawaii (H), Oahu (O), Maui (M; this includes Molokai and Lanai), and Kauai (K; this includes Niihau). These areas are modeled have arisen 0.5, 1.9, 3.7, and 5.5 million years ago, respectively.

Although the model will use discrete-state biogeographic ranges, geographical area is naturally continuous. This means we must impose some discretization upon the geography to designate a set of biogeographically meaningful characters called areas. Different methods use different criteria for this discretization, so it is best to perform the discretization yourself rather than blindly using the discretization given from a previous study or method (but do blindly use the dataset included in this tutorial). Some geographies have natural discretizations: for instance, the Hawaiian archipelago forms naturally discrete areas on the basis of islands. For many geographies, however, it may unclear how to perform this discretization. Much like morphological analyses, you might choose to choose areas based on expert opinion, based on some model, or using some "naive" uniform discretization. This procedure is not part of the tutorial, but you should be aware that area definitions are not always obvious or objective.

```
{
        "name":"HawaiianArchipelago5my",
        "epochs": [
                {
                        "name":"epoch1",
                        "start_age":100.0,
                        "end_age":3.7,
                        "areas":
                        [{ "name":"Kauai",  "latitude":19.5667, "longitude
                            ": -155.5000, "dispersalValues": [ 1,0,0,0 ] },
                        { "name": "Oahu",   "latitude":19.5667, "longitude
                            ": -155.5000, "dispersalValues": [ 0,0,0,0 ] },
                        { "name": "Maui",   "latitude":19.5667, "longitude
                            ": -155.5000, "dispersalValues": [ 0,0,0,0 ] },
                        { "name": "Hawaii", "latitude":19.5667, "longitude
                            ": -155.5000, "dispersalValues": [ 0,0,0,0 ] }]
                },
                {
                        "name":"epoch2",
                        ...
                },
                {
                        "name":"epoch3",
                        ...
                },
                {
                        "name":"epoch4",
                        "start_age":0.5,
                        "end_age":0.0,
                        "areas":
                        [{ "name":"Kauai",  "latitude":22.0833, "longitude
                            ": -159.5000, "dispersalValues": [ 1,1,1,1 ] },
                        { "name": "Oahu",   "latitude":21.4722, "longitude
                            ": -157.9772, "dispersalValues": [ 1,1,1,1 ] },
                        { "name": "Maui",   "latitude":20.8000, "longitude
                            ": -156.3333, "dispersalValues": [ 1,1,1,1 ] },
                        { "name": "Hawaii", "latitude":19.5667, "longitude
                            ": -155.5000, "dispersalValues": [ 1,1,1,1 ] }]
                }
        ]
}
```

This is called the Atlas file, which uses a file format called JSON. 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 area's centroid. If a centroid does not represent the distance between areas, splitting the area into multiple smaller areas is reasonable. The data augmentation approach used in this analysis allows you to use more areas as desired, whereas matrix exponentiation methods are limited to approximately ten areas. The distance between any two areas' coordinates inform to distance-dependent dispersal parameter (β from the $\eta(\cdot)$ function) for range expansion events, so coordinates roughly close to the center of the area suffice. 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 ȅpoch1 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.

8.4 RevBayes Analysis

There are six major parts to the analysis.

First, we need to read in the input files and assign analysis settings. Second, we need to construct our model. Third, we need to assign moves and monitors to our model parameters for use with Markov chain Monte Carlo (MCMC). Fourth, we will run an MCMC analysis assuming a complex model. Fifth, we will compare the complex model with a simple model using Bayes factors. Finally, we'll analyze our MCMC output.

8.4.1 Analysis settings

```
$ revbayes
```

First, we'll assign all our input files to `String` variables.

```
RevBayes > in_fp    <- "./input/"
RevBayes > data_fn  <- "psychotria_range.nex"
RevBayes > area_fn  <- "hawaii_dynamic.atlas.txt"
RevBayes > out_fp   <- "./output/"
RevBayes > out_str  <- "bg_2rate"
```

Then we'll create our range data, tree, and atlas objects

```
RevBayes > data  <- readDiscreteCharacterData(in_fp + data_fn)
RevBayes > tree  <- readTrees(in_fp + data_fn)[1]
RevBayes > atlas <- readAtlas(in_fp + area_fn)
```

8.4.2 Creating the model

Here, we will compose our rate matrix, **Q**, parameterized by the transition rates, λ, and the distance dependent dispersal power parameter, β.

First, for λ, we will create a vector of two rates, where `glr[1]` corresponds to the rate of area loss (local extinction) and `glr[2]` corresponds to the rate of area gain (dispersal). Each rate will be drawn from an exponential distribution with rate 10.0 (mean 0.1). Because our tree is in units of millions of years, this means our prior expectation is that any given species undergoes one dispersal or extinction event per area per ten million years.

To introduce this to the model, type

```
RevBayes > for (i in 1:2) glr[i] ~ dnExponential(10.0)
```

Next, we will create `dp`, 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 rate 10.0 (mean 0.1) as a prior for `dp`.

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.

```
RevBayes > dp ~ dnExponential(10.0)
RevBayes > grm := fnBiogeoGRM(atlas=atlas, distancePower=dp, useAvailable=
    true, useDistance=true)
```

Now we need a deterministic node to represent the rate matrix, **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, `glr`, and our geographical rate modifier, `grm`. 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).

```
RevBayes > Q := fnBiogeoDE(gainLossRates=glr, geoRateMod=grm, numAreas=4,
    forbidExtinction=true)
```

To extract information for the frequencies of different cladogenic event types, we will create a Dirichlet-distributed stochastic node. The simplex is over three events, subset sympatry (index 0), allopatry (index 1), and widespread sympatry (index 2), but not over narrow sympatry whose range size is one. The prior parameter `[1,1,1]` is known as a flat prior, meaning all event types are expected to occur at equal frequency. If there is information in the data of a dominant cladogenic mode of range evolution, the posterior simplex values in `csf` will reflect this.

```
RevBayes > csf ~ dnDirichlet([1,1,1])
```

For the model's final node, we create the stochastic node for the continuous-time Markov chain (CTMC). This node's distribution is `dnPhyloDACTMC` where `DA` indicates the CTMC uses data-augmentation to compute the likelihood rather than Felsenstein's pruning algorithm. To create the distribution, we must pass it our `tree` and `Q` 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 Ree and Smith (2008) (`useCladogenesis=true`), and that it will assign zero probability to a transition away from the null range state.

```
RevBayes > M ~ dnPhyloDACTMC(tree=tree, Q=Q, C=csf, 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 will prime the model with data-augmented character histories. Now `M` has a defined likelihood value.

```
RevBayes > M.clamp(data)
RevBayes > M.lnProb
    -56.0288
```

Finally, we encapsulate our graphical model into a `Model` object, which can learn the model's structure and dependencies from any model parameter.

```
RevBayes > my_model <- model(glr)
```

8.4.3 Running an MCMC analysis

Now that we have our `Model` object, we can soon run an MCMC analysis. Remember that MCMC approximates the posterior distribution by repeatedly proposing new model parameter values, accepting or rejecting those new parameter values based on the model likelihood (and on biases in the proposal distribution), then reporting the sampled parameter values.

First, let's assign moves to our model parameters. These parameters are all supported for real positive values, which is appropriate for use with scale-multipler proposal, `mvScale()`. To inspect our model parameter types and the proposal argument types, enter

```
RevBayes > type(dp)
   RealPos
RevBayes > type(glr)
   RealPos[]
RevBayes > mvScale
   Move_Scale function (RealPos x, RealPos lambda, Bool tune, RealPos weight
      )
```

The arguments for `mvScale` are fairly typical as far as RevBayes `Move` objects go: `x` is the stochastic node the `Move` will update, `lambda` is proportional to how radically different proposed parameter values will tend to be, `tune` allows `lambda` to be adjusted automatically as the MCMC runs, and `weight` tells the MCMC how many times to perform the `Move` during a single MCMC generation (e.g., `weight=2.0)` means each generation will call that `Move` for the parameter `x` twice).

```
RevBayes > moves[1] <- mvScale(x=glr[1], lambda=0.5, tune=false, weight=5.0)
RevBayes > moves[2] <- mvScale(x=glr[2], lambda=0.5, tune=false, weight=5.0)
RevBayes > moves[3] <- mvScale(x=dp,     lambda=0.5, tune=false, weight=5.0)
RevBayes > moves[4] <- mvSimplexElementScale(csf, alpha=10.0, tune=false,
   weight=4.0)
```

In addition to proposing new model parameter values, we must also propose new data-augmented states and events to properly integrate over the space of possible range histories. The major challenge to sampling character histories is ensuring the character histories are consistent with the observations at the tip of the tree. The proposals in this tutorial use Nielsen (2002)'s rejection sampling algorithm, with some modifications to account for cladogenic events and epoch-based rate matrices.

The basic idea is simple. Each time a character history proposal is called, it selects a node at random from the tree. Path history proposals (`mvPathCHRS()`) propose a new character history for the lineage leading to that node. Node history proposals (`mvNodeCHRS()`) propose a new character history for the node and for the three lineages incident to that node. The character history proposal also samples some number of areas to update, ranging from one to all of the areas. 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.

Because these `Move` objects update the character histories stored in the data-augmented CTMC node, e.g. `M`, they require access to a `TimeTree` object to know which lineages are sisters and whether the lineages span various epochs, and a `RateMap_Biogeography` object to propose new character histories. The `lambda` argument gives what proportion of areas' character histories to update. Here, if `lambda=0.2`, then the proposal will redraw character histories for each area with probability 0.2 (in addition to one random area with probability 1). Below, we use two moves of each type with `lambda=0.2` and `lambda=1.0` for partial and full character history updates, respectively. Indicating `type="biogeo"` informs the `Move` object to be aware of special character history constraints, such as cladogenic events and forbidden null ranges. The `weight` parameter should be assigned a value proportional to the number of nodes in the analysis to ensure proper mixing.

Let's create the character history moves as follows.

```
RevBayes > n_nodes   <- tree.nnodes()
RevBayes > moves[5] <- mvNodeCHRS(ctmc=M, qmap=Q, tree=tree, lambda=0.2,
   type="biogeo", weight=2.0*n_nodes)
RevBayes > moves[6] <- mvPathCHRS(ctmc=M, qmap=Q, tree=tree, lambda=0.2,
   type="biogeo", weight=2.0*n_nodes)
RevBayes > moves[7] <- mvNodeCHRS(ctmc=M, qmap=Q, tree=tree, lambda=1.0,
   type="biogeo", weight=n_nodes)
RevBayes > moves[8] <- mvPathCHRS(ctmc=M, qmap=Q, tree=tree, lambda=1.0,
   type="biogeo", weight=n_nodes)
```

Now that we have moves for all our parameters and the character histories, we'll proceed with assigning `Monitor` objects to record their values. The first two `Monitor` objects are fairly standard and found in most RevBayes MCMC analyses. `mnScreen` reports the values for any nodes assigned to `RevObject ...` every `printgen` generations to the terminal screen. `mnModel` reports the values for all nodes in the `Model` object every `printgen` generations to the file assigned to `filename`, which is delimited by the `separator` character.

```
RevBayes > mnScreen
   Mntr_Screen function (RevObject ..., Natural printgen, Bool posterior,
      Bool
   likelihood, Bool prior)
RevBayes > monitors[1] <- mnScreen(printgen=10, glr, dp, csf)
RevBayes > mnModel
   Mntr_Model function (String filename, Natural printgen, String separator,
   Bool posterior, Bool likelihood, Bool prior, Bool append, Bool
   stochasticOnly)
RevBayes > monitors[2] <- mnModel(filename=out_fp+params_fn, 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 `Mntr_CharacterHistoryNewickFile` objects 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 ??.

```
RevBayes > mnCharHistoryNewick
   Mntr_CharacterHistoryNewickFile function (String filename,
   AbstractCharacterData ctmc, TimeTree tree, Natural printgen, String
   separator, Bool posterior, Bool likelihood, Bool prior, Bool append,
      String
   style = events|counts
   , String type = biogeo
   )
RevBayes > monitors[3] <- mnCharHistoryNewick(filename=fp+out_str+".events.
   txt", ctmc=M, tree=tree, printgen=100, style="events")
RevBayes > monitors[4] <- mnCharHistoryNewick(filename=fp+out_str+".counts.
   txt", ctmc=M, tree=tree, printgen=100, style="counts")
```

As our last monitor, the `Mntr_CharacterHistoryNhxFile` 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.

```
RevBayes > mnCharHistoryNhx
   Mntr_CharacterHistoryNhxFile function (String filename,
      AbstractCharacterData
   ctmc, TimeTree tree, RlAtlas atlas, Natural samplegen, Natural maxgen,
   Probability burnin, String separator, Bool posterior, Bool likelihood,
      Bool
   prior, String type = biogeo
   )
RevBayes > monitors[5] <- mnCharHistoryNhx(filename=fp+out_str+".nhx.txt",
   ctmc=M, tree=tree, atlas=atlas, samplegen=100, maxgen=25000, burnin=0.2)
```

Before we run the MCMC, we'd like to get the node index of the ancestor When analysing the output, we'll take some special interest in the branch for the most recent common ancestor of *P. kaduana* and *P. hatheway*. We will identify this lineage by its index, 23, which is meaningful only for a fixed tree topology,

```
RevBayes > names = tree.names()
RevBayes > names[15]
   P_kaduana_PuuKukuiAS
RevBayes > names[16]
   P_hathewayi_1
RevBayes > mrcaIndex(tree=tree,clade=clade(names[15],names[16]))
   23
```

8.4.4 Running an MCMC analysis

Now all that's left is to configure and run our MCMC analysis. For this, we create an `Mcmc` object, which we give our `Move` vector, our `Monitor` vector, and our `Model object`

```
RevBayes > mcmc
   MCMC function (Model model, Monitor[] monitors, Move[] moves, String
    moveschedule = sequential|random|single
    )
RevBayes > my_mcmc <- mcmc(my_model, monitors, moves)
```

MCMC typically requires some period of burn-in before it reaches stationarity, i.e. from a random starting point, it takes some time for the chain to produce valid samples from the posterior distribution. By running **`burnin()`**, we tell the `Mcmc` object to propose and reject new states but *not* to record anything to file. After burn-in is complete, we call **`run()`**, where we begin recording valid posterior samples under our model.

```
RevBayes > my_mcmc.burnin(generations=1000, tuningInterval=50)
RevBayes > my_mcmc.run(generations=25000)
```

Everything we've done is contained in the file `biogeography_DEC_2rate.Rev`. You can modify this file as you like then re-run the analysis by typing

```
RevBayes > source("./scripts/biogeography__DEC__2rate.Rev")
```

8.4.5 Model selection using Bayes factors

Bayes factors (BFs) are used to select which of two models better describes the observed data, $\mathbf{X}_{obs}$, and are computed as the ratio of marginal likelihoods for those two models. One might prefer to analytically compute the marginal likelihood, but it's the same intractable quantity we intentionally avoid computing when using MCMC in a Bayesian context. Instead, we must estimate the marginal likelihood from our posterior distribution samples. Here, we will use thermodynamic integration (Lartillot and Philippe 2006) and stepping-stone approximation (Xie et al. 2010). The exact details of these techniques will not be covered here, but there is an important practical point to mention: both methods rely on computing a number of "power posterior" distributions. Computing more power posteriors increases the marginal likelihood estimator's accuracy at the cost of computational time.

Moving on, we'll compute the Bayes factor to compare a simple one-rate model, which asserts the rate of area gain and loss are always equal, to a two-rate model which allows these rates to vary independently. Rather than specifying the model manually, we will load (source) the model definition from a file then enter the commands to compute its marginal likelihood. For faster results, we will use two separate RevBayes sessions, one for each model. For each session, the power posterior analysis run for 1000 generations during burn-in then 1000 generations per each of 30 power posterior categories.

First session:

```
RevBayes > source("./scripts/biogeography_DEC_1rate.Rev")
RevBayes > pp_fn <- out_fp + out_str + ".pp.txt"
RevBayes > pow_p <- powerPosterior(my_model, moves, pp_fn, cats=30)
RevBayes > pow_p.burnin(generations=1000,tuningInterval=100)
RevBayes > pow_p.run(generations=5000)
```

Second session:

```
RevBayes > source("./scripts/biogeography_DEC_2rate.Rev")
RevBayes > pp_fn <- out_fp + out_str + ".pp.txt"
RevBayes > pow_p <- powerPosterior(my_model, moves, pp_fn, cats=30)
RevBayes > pow_p.burnin(generations=1000,tuningInterval=100)
RevBayes > pow_p.run(generations=5000)
```

Each power posterior analysis will write their contents to the file given in `pp_fn`. These files are `bg_1rate.pp.txt` and `bg_2rate.pp.txt` for the simple and complex models, respectively. This may take a few minutes. When complete, the power posterior files may then be used to compute marginal likelihoods. For example, from the RevBayes session analyzing the simple one-rate model

```
RevBayes > ss <- steppingStoneSampler(file=pp_fn)
RevBayes > ss.marginal()
RevBayes > ps <- pathSampler(file=pp_fn)
RevBayes > ps.marginal()
```

For a given model, the path sampling and stepping stone sampling methods should produce similar marginal likelihood estimates. Values should be within one log likelihood unit of one another. If the values are extremely different, this may indicate `powerPosterior` should be re-run with a larger number of `cats`. We chose `cats=30` which should suffice, and we see no problem. Then from the complex two-rate model RevBayes session using the same commands as above. Finally, we can compute the Bayes factor, which is simply the ratio of marginal likelihoods.

```
RevBayes > exp(-51.7202)/exp(-52.3158)
   1.81412
```

A value of one would mean both models had equal marginal likelihoods. A value less than one would indicate the first model, the simple model, had a larger marginal likelihood, and was therefore favored by model testing. But that's not the case, the value is greater than one, and the complex two-rate model is favored. Similar to frequentist interpretations of significance for p-values, there is no universal and objective criterion of significance with Bayes factors, but most would agree a factor of 1.8 (or 1.0/1.8) indicates weak support for one model over the other.

8.5 Output

8.5.1 Sampled parameters from `ScreenMonitor`

The `mnScreen` monitor reports model parameter values to the screen, where each row corresponds to the current accepted MCMC state, and each column reports some model feature, such as the model likelihood or a parameter value. Every 20 iterations, this monitor re-prints the column headers.

```
RevBayes > my_mcmc.run(generations=25000)

Running MCMC simulation for 25000 iterations
The simulator uses 8 different moves in a random
move schedule with 241 moves per iteration

Iteration    |    Posterior   |           dp   |       glr[1]   |       glr[2]   |
        csf[1]   |       csf[2]   |       csf[3]
-------------------------------------------------------------------------------------------


0            |     -51.3307   |    0.0570518   |     0.175137   |    0.0580957   |
      0.330891   |     0.255308   |     0.413801
10           |     -54.4257   |    0.0416423   |     0.166936   |     0.178402   |
   0.0549136   |     0.148854   |     0.796233
20           |     -58.0696   |    0.0991853   |     0.136495   |     0.122135   |
      0.308418   |     0.448892   |     0.242689
30           |     -46.5049   |      0.10676   |    0.0958918   |    0.0959592   |
      0.543837   |     0.363871   |    0.0922922
40           |     -42.8697   |     0.173549   |     0.158565   |    0.0662419   |
      0.569416   |     0.126439   |     0.304145
50           |     -43.5319   |     0.117868   |     0.196497   |    0.0523307   |
      0.440269   |     0.257171   |      0.30256

...
```

For the complex 2-rate model, our model parameters are `dp`, the distance power parameter, and the rates of area loss and gain, `glr[1]` and `glr[2]`, respectively, and the frequencies for subset sympatry, allopatry, and widespread sympatry `csf[1]`, `csf[2]`, and `csf[3]`, respectively. If you notice the value of some parameter is rarely updated from iteration to iteration, the MCMC is probably mixing poorly therefore it's not generating samples from the posterior distribution (the MCMC's stationary distribution). In this case, you may want to re-run the analysis with different arguments for the `Move` object assigned to that parameter.

8.5.2 Sampled parameters from `ModelMonitor`

This tab-delimited file contains parameter samples from the posterior distribution. As with the `ScreenMonitor`, columns are model or parameter values and rows are MCMC cycles.

```
Iteration   Posterior   Likelihood     Prior   glr[1]   glr[2]   dp   csf[1]    csf
    [2]   csf[3]
0             -51.3307     -56.0288   4.69806   0.175137   0.0580957   0.057051
    0.330891   0.255308   0.413801
10            -54.4257     -58.1568   3.73110   0.166936   0.1784020   0.041642
    0.054913   0.148854   0.796233
20            -58.0696     -62.0923   4.02274   0.136495   0.1221350   0.099185
    0.308418   0.448892   0.242689
30            -46.5049     -51.1197   4.61480   0.095891   0.0959592   0.106760
    0.543837   0.363871   0.092292
40            -42.8697     -46.4870   3.61735   0.158565   0.0662419   0.173549
    0.569416   0.126439   0.304145
50            -43.5319     -47.4659   3.93394   0.196497   0.0523307   0.117868
    0.440269   0.257171   0.302560

...
```

Here, we see a strong negative correlation between the posterior probability and the area gain rate, which is expected. Next, click the Estimates tab then select the three `csf` parameters.

8.5.3 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.

```
Iteration   Posterior   Likelihood     Prior         t_s0      t_s1      t_c0
    t_c1      t_c2      t_c3      b0_s0     b0_s1     b0_c      ...
0             -51.3307     -56.0288   4.69806        9         9         18        0
              0         0         1         1         0         ...
10            -54.4257     -58.1568   3.73110        9         10        17        0
              0         1         1         1         0         ...
20            -58.0696     -62.0923   4.02274        11        9         15        2
              1         0         2         1         1         ...
30            -46.5049     -51.1197   4.61480        8         8         18        0
              0         0         1         1         0         ...
40            -42.8697     -46.4870   3.61735        7         7         18        0
              0         0         1         1         0         ...
50            -43.5319     -47.4659   3.93394        7         7         18        0
              0         0         1         1         0         ...
...
```

For example, `b2_s1` gives the number of areas that are gained for the branch leading to the node indexed 2. `b2_c` gives the cladogenic event type that gives rise to the node indexed 2, where narrow sympatry, subset sympatry, allopatry, and widespread sympatry 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`, and `t_c2` give the total number of narrow sympatric, subset sympatric, allopatric, and widespread sympatric cladogenic events over the entire tree.

Because the expected number of gain events should be proportional to the area gain rate, we expect to see the same negative correlation between posterior probability and number of events as we did with the posterior and rate in the `parameters.txt` file.

Open Tracer, select the fields for the posterior probability and the number of gained areas over the tree, `t1`, then click the Joint-Marginal tab.

One interesting facet of this output is there are never fewer than six events. In fact, since we assume a stratified geography and that only one event may occur per instant, it is impossible to describe the data we see at the tips with fewer than six gain events. That is, six gain events is part of the maximum parsimony solution.

8.5.4 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.

```
Iteration   Posterior   Likelihood   Prior   Tree
Iteration         Posterior         Likelihood        Prior    Tree
0         -51.3307          -56.0288          4.69806  (((((((
    P_hawaiiensis_WaikamoiL1[&index=18;nd=0010;pa=0010;ev={}]:0.96  ...
10        -54.4257          -58.1568          3.7311   (((((((
    P_hawaiiensis_WaikamoiL1[&index=18;nd=0010;pa=0010;ev={}]:0.96  ...
20        -58.0696          -62.0923          4.02274  (((((((
    P_hawaiiensis_WaikamoiL1[&index=18;nd=0010;pa=0010;ev={}]:0.96  ...
30        -46.5049          -51.1197          4.6148   (((((((
    P_hawaiiensis_WaikamoiL1[&index=18;nd=0010;pa=0010;ev={}]:0.96  ...
40        -42.8697          -46.4870          3.61735  (((((((
    P_hawaiiensis_WaikamoiL1[&index=18;nd=0010;pa=0010;ev={}]:0.96  ...
50        -43.5319          -47.4659          3.93394  (((((((
    P_hawaiiensis_WaikamoiL1[&index=18;nd=0010;pa=0010;ev={}]:0.96  ...

...
```

Each iteration records the data-augmented character history (stochastic mapping) using metadata labels, which, for an internal node, looks like

```
[&index=23;nd=0110;pa=0010;ch0=0010;ch1=0110;cs=s;bn=16;ev={{t:0.2513,a
    :1.1195,s:1,i:1}}
```

`index=23` indicates this branch leads to the node indexed 23. The branch began in the ancestral state `pa=0100` and terminated in the state `nd=0110`. Since this node is not a tip node, it represents a speciation event, so the daughter ranges are also given, `ch0=0010` and `ch1=0110`. 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 19 are recorded in `events`, where each event has a time (relative to the absolute branch length), absolute age, state (into), and character index (`t`, `a`, `s`, `i`, resp.). For this posterior sample of the character history for the branch leading to node 22, the species range expanded into Oahu at age 1.1195.

To manipulate this data format, we'll use Python scripts. Below are a few examples of interesting posterior features.

```
> cd scripts
> python27

...

>>> from bg_parse import *
>>> dd=get_events(fn="../output/bg_2rate.events")
```

By default, `get_events()` extracts a dictionary where each node index maps to a branch's character history as reported in `./input/bg_2rate.events.txt`. Each branch is a dictionary whose keys are various parts of the MCMC state and whose values the MCMC samples.

```
>>> dd[23].keys()
['ch1', 'iteration', 'bn', 'nd', 'ch0', 'prior', 'posterior', 'cs', 'ev', '
    likelihood']
>>> dd[23]['posterior'][0:5]
[-48.6952, -60.1832, -53.2286, -57.5778, -53.4633]
```

To get the $n = 1$ highest-valued sample for a branch by its posterior value

```
>>> get_best(dd[23],n=1,p='posterior')
{'prior': [4.48225], 'iteration': [14890], 'bn': [22], 'nd': [[0, 1, 1, 0]],
     'ch0': [[0, 1, 1, 0]], 'ch1': [[0, 0, 1, 0]], 'posterior': [-34.7139], '
    pa': [[0, 1, 0, 0]], 'cs': ['subset_sympatry'], 'ev': [[{'age': 1.5637, '
    state': 1, 'idx': 2, 'time': 0.8611}]], 'likelihood': [-39.1962]}
```

To get the probability that area i and area j are both part of the species range as the branch for node 23 terminates, just before the speciation event

```
>>> get_area_pair(dd[23])
[[0.0816, 0.0188, 0.0628, 0.0000],
 [0.0188, 0.7081, 0.4390, 0.0000],
 [0.0628, 0.4390, 0.7141, 0.0000],
 [0.0000, 0.0000, 0.0000, 0.0000]]
```

showing area 3 was occupied nearly with probability 0.71 and both areas 2 and 3 were occupied with probability 0.44. Note, Hawaii was submerged until approximately 0.5 million years ago, and thus the probability of being in that area is 0.0.

If the range is size one during a speciation event, the cladogenic event state is always narrow sympatric, **'narrow_sympatry'**. But given the opportunity for non-sympatric events, i.e. that the range is larger than size one, we can get the probability of cladogenic state using For the probability for cladogenic event state given the range was larger than size one

```
>>> get_clado_state(dd[23])
{'allopatry': 0.0224, 'subset_sympatry': 0.1463, 'widespread_sympatry':
    0.3183, 'narrow_sympatry': 0.5130}
>>> get_clado_state(dd[23],minSize=2)
{'allopatry': 0.0460, 'subset_sympatry': 0.3005, 'widespread_sympatry':
    0.6535, 'narrow_sympatry': 0.0000}
>>> get_clado_state(get_best(dd[23],n=100),minSize=2)
{'allopatry': 0.1290, 'subset_sympatry': 0.6774, 'widespread_sympatry':
    0.1936, 'narrow_sympatry': 0.0000}
```

Depending on your question, different aspects of the posterior cladogenic state will interest you. Narrow sympatry is the favored ancestral state, but wide sympatry is favored for ranges of size $n > 1$. However, when we look at the 100 most probable samples, subset sympatry becomes most favored.

More script functions are found in `./scripts/bg_parse.py`.

8.5.5 New Hampshire extended format file (`./output/bg_2rate.nhx`)

Because this data is very high-dimensional, we'll use an external data exploration tool to look at range evolution.

This file summarizes the input and output from a BayArea analysis using NEXUS format containing a New Hampshire eXtended (NHX) tree string. NHX allows you to annotate nodes in a Newick string with meta-information, which BayArea uses to report the probabilities in the `my_run.area_probs.txt` file. The `geo` block gives the geographical latitudes and longitudes for the areas in the order they are reported as probabilities. Like the `my_run.area_probs.txt` file, this file is not written until the analysis is complete. This annotation is used for the two visualization programs covered in the next section, Phylowood and BayArea-Fig. The anatomy of the Phylowood and BayArea-Fig settings blocks will also be explained there.

8.6 Visualization

Here we'll explore two options for visualizing ancestral range reconstructions. I'll walk you through some of the basic functionality, but feel free to play around as you like.

8.6.1 Phylowood

Phylowood generates interactive animations to explore biogeographic reconstructions.

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.

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.

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.

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.

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.

Bibliography

Goldberg EE, Lancaster LT, Ree RH. 2011. Phylogenetic inference of reciprocal effects between geographic range evolution and diversification. Systematic Biology. 60:451–465.

Landis MJ, Bedford T. 2014. Phylowood: interactive web-based animations of biogeographic and phylogeographic histories. Bioinformatics. 30:123–124.

Landis MJ, Matzke NJ, Moore BR, Huelsenbeck JP. 2013. Bayesian analysis of biogeography when the number of areas is large. Systematic biology. 62:789–804.

Lartillot N, Philippe H. 2006. Computing Bayes factors using theromodynamic integration. Systematic Biology. 55:195–207.

Matzke NJ. 2013. Probabilistic historical biogeography: new models for founder-event speciation, imperfect detection, and fossils allow improved accuracy and model-testing. Frontiers of Biogeography. 5.

Nepokroeff M, Sytsma KJ, Wagner WL, Zimmer EA. 2003. Reconstructing ancestral patterns of colonization and dispersal in the hawaiian understory tree genus psychotria (rubiaceae): a comparison of parsimony and likelihood approaches. Systematic Biology. 52:820–838.

Nielsen R. 2002. Mapping mutations on phylogenies. Systematic Biology. 51:729–739.

Ree RH, Smith SA. 2008. Maximum likelihood inference of geographic range evolution by dispersal, local extinction, and cladogenesis. Systematic Biology. 57:4–14.

Robinson DM, Jones DT, Kishino H, Goldman N, Thorne JL. 2003. Protein evolution with dependence among codons due to tertiary structure. Molecular Biology and Evolution. 20:1692–1704.

Xie W, Lewis PO, Fan Y, Kuo L, Chen MH. 2010. Improving marginal likelihood estimation for bayesian phylogenetic model selection. Systematic Biology. p. syq085.

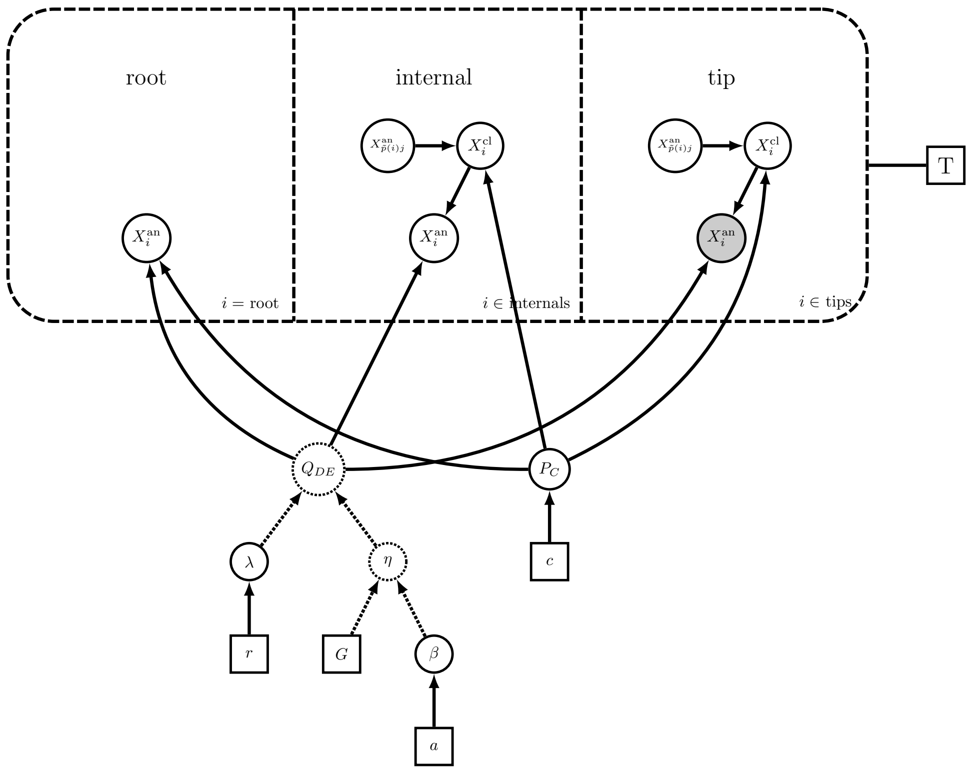


Figure 8.1: Graphical model of DEC. The tree plate's topology is fixed by T, where each internal node has both an anagenic and cladogenic random variable (X_i^{an} and X_i^{cl}, resp.) that represents an ancestral species before and after it speciated. Anagenic change is modeled by a continuous time Markov process, where Q_{DE} is the instantaneous rate matrix of area gain and loss, as parameterized by λ. The geographic distance rate modifier function, η, takes in the geographical distances and strata as G, and the distance power parameter, β. Cladogenic change is modeled by P_C, a Dirichlet-distributed simplex with a flat prior.

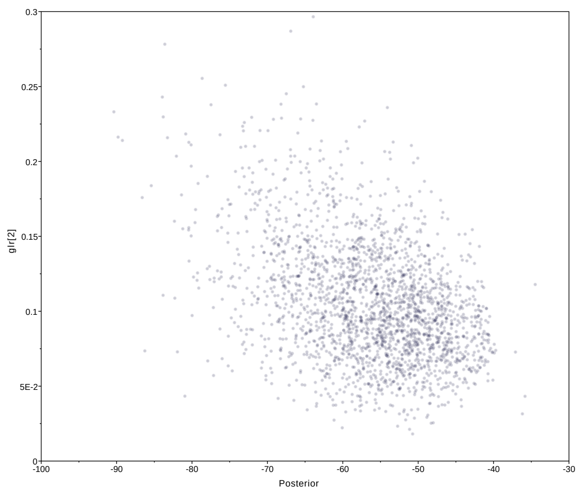


Figure 8.2: Joint-marginal distribution of posterior and area gain rate, λ_1.

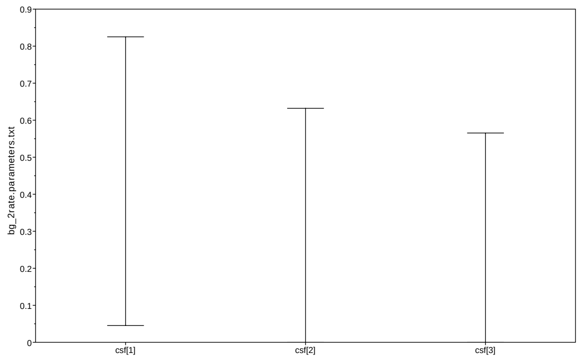


Figure 8.3: Mean values for the cladogenic state frequency simplex, where `csf[1]`, `csf[2]`, and `csf[3]` correspond to subset sympatry, allopatry, and wide sympatry whose mean posterior values are 0.45, 0.30, and 0.25, respectively.

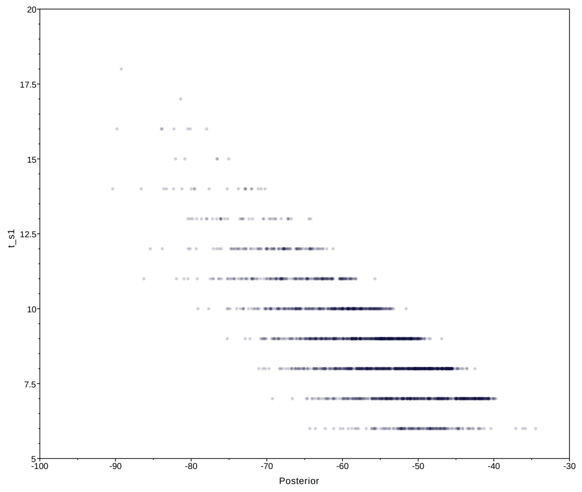


Figure 8.4: Joint-marginal distribution of posterior and number dispersal events summed over the tree.

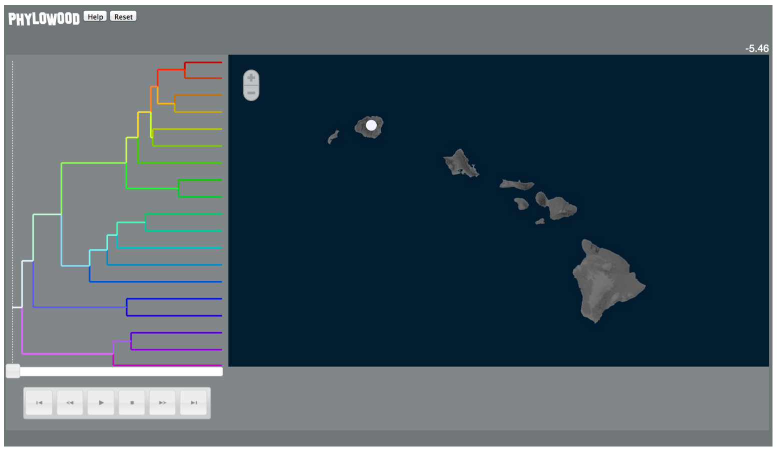


Figure 8.5: Phylowood frame showing posterior ancestral range of root node.

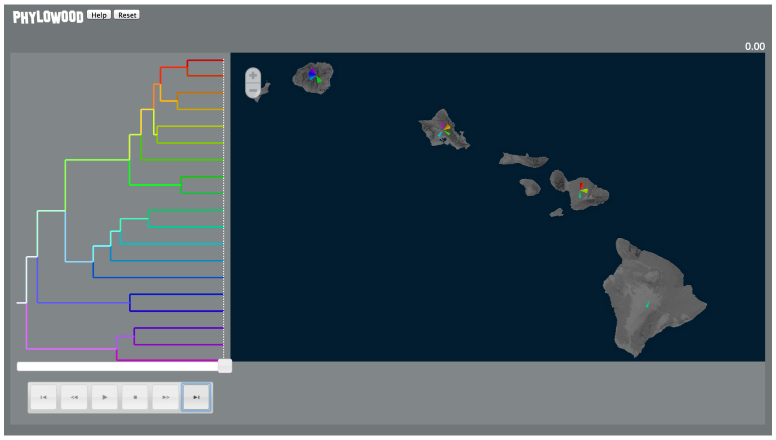
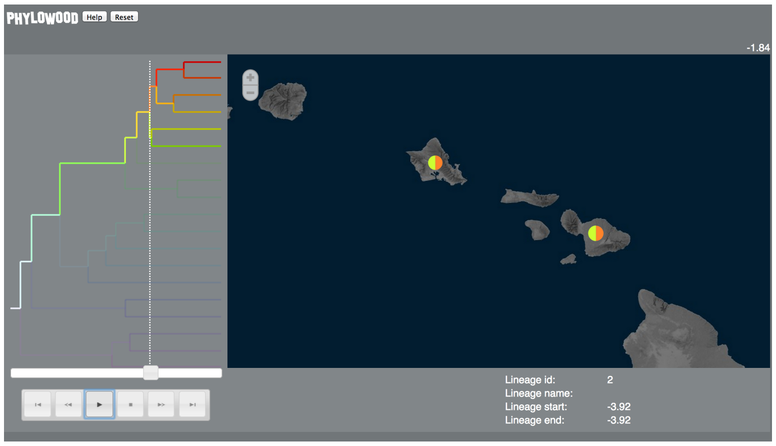


Figure 8.6: Phylowood frame showing distribution of extant taxon ranges.



```
Lineage id:       2
Lineage name:
Lineage start:    -3.92
Lineage end:      -3.92
```

Figure 8.7: Phylowood frame highlighting the posterior range for the most recent common ancestor of *P. mauiensis* and *P. hawaiiensis*.

Chapter 9

Phylogenetic Comparative Analyses: Continuous Trait Evolution

9.1 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 (see ?; for a good introduction).

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 contrast method (?). 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.

9.2 Data and files

In the **`data`** folder, you will find the following files

- **`plac40lhtlog.nex`**: 3 life-history traits (age at sexual maturity, body mass, maximum recorded lifespan) for 40 placental mammals (taken from the Anage database, ?). The traits have been log-transformed.
- **`plac40_4fold.nex`**: an alignment made of a concatenation of 17 nuclear genes in 40 placental mammals (from ?), with only the four-fold degenerate third coding positions.
- **`chronoplac40.tree`**: a time-calibrated phylogeny, which has been obtained by running another software program (PhyloBayes, ?). On the cluster, and if you are logged under an X-terminal, you can visualize this tree using the **`njplot`** command:
  `njplot chronoplac40.tree`
- **`archaeaRNA.nex`**: an alignment of rRNA sequences of 33 archaeal species.
- **`archaeaTemp.nex`**: optimal growth temperatures for the 33 archaeal species.

9.3 Univariate Brownian evolution of quantitative traits

As a first preliminary exercise, we wish to reconstruct the evolution of body mass in placental mammals 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 **`chronoplac40.tree`**.

You may want to take the time to visualize the tree given in **`chronoplac40.tree`** as well as the matrix of quantitative traits specified by the **`plac40lhtlog.nex`** file, before going into the modeling work described below.

9.3.1 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 $\sigma^2 T$:

$$x(T) \quad \sim \quad \text{Normal}\left(x(0), \sigma^2 T\right).$$

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:

$$\sigma \quad \sim \quad \frac{1}{\sigma}.$$

Concerning the initial value $x(0)$ of the Brownian process at the root of the phylogeny, the current version of RevBayes only implements a uniform prior. This is done by default (no need to explicitly define it).

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 $x(t)$:

$$\begin{aligned} \sigma \quad &\sim \quad \frac{1}{\sigma}, \\ x(0) \quad &\sim \quad \text{Uniform}, \\ x(t) \mid \Psi, \sigma \quad &\sim \quad \text{Brownian}\left(x(0),\, \psi,\, \sigma\right). \end{aligned}$$

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.

9.3.2 Programming the model in RevBayes

In **`tutorials/NESCent/RevBayes_scripts/`**, you will find a script called **`placentaliaMass.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, by

first going to the **`tutorials/NESCent/`** folder and then giving the script as an argument to the **`RevBayes`** program:
`RevBayes RevBayes_scripts/placentaliaMass.Rev`.

This script essentially reformulates what has been explained in the last subsection, now in the Rev language:

- load trait data:
  `contData <- readCharacterData("data/plac40lhtlog.nex")`
- load the time-tree from file:
  `treeArray <- readTrees("data/chronoplac40.tree"`
  `psi <- treeArray[1]`
- define σ, with a truncated log-uniform prior:
  `sigma ~ dnLogUniform(min=0.001,max=1000)`
  to accelerate convergence, it can be useful to force initialization of σ to a small value:
  `sigma.setValue(0.1)`
- define the multivariate Brownian process, which we will call **`logmass`**:
  `logmass ~ dnBrownian(psi,sigma)`
- condition the Brownian model on empirically observed values for body mass in extant taxa. Here, we need to specify that body mass is the second column of the dataset:
  `logmass.clampAt(contData,2)`

The model is now entirely specified. We can define the moves on its parameters:

- initialize a running index for storing moves:
  `index <- 1`
- push a scaling move on σ:
  `moves[index] <- mvScale(sigma, lambda=2.0, tune=true, weight=3.0)`
  `index <- index + 1`
- a sliding move on the Brownian process
  `moves[index] <- mvRealNodeValTreeSliding(process=logmass, lambda=10, tune=true, weight=10`
  `index <- index + 1`
- a global translation move on the Brownian process:
  `moves[index] <- mvRealNodeValTreeTranslation(process=logmass,lambda=1,`
  `tune=true,weight=1)`
  `index <- index + 1`
- before creating the model, we define summary statistics, to be monitored during the MCMC: the mean and the standard deviation of the trait across the tree:
  `meanlogmass := logmass.mean()`
  `stdevlogmass := logmass.stdev()`
- as well as the value of the log of body mass for the root:
  `rootlogmass := logmass.rootVal()`

- now, create the model
  ```
  mymodel <- model(sigma)
  ```
- make a screen monitor that tracks the summary statistics of interest:
  ```
  monitors[1] <- mnScreen(printgen=10, sigma, rootlogmass, meanlogmass, stdevlogmass)
  ```
- a file monitor that does the same thing, but directly into a file:
  ```
  monitors[2] <- mnFile(filename="output/placmass.trace", printgen=10, separator = " ",
  sigma, rootlogmass, meanlogmass, stdevlogmass)
  ```
- a file monitor for the ancestral reconstruction of traits along the entire tree (in newick format):
  ```
  monitors[3] <- mnFile(filename="output/placmass.logmass", printgen=10, separator = "
  ", logmass)
  ```
- and a general model monitor:
  ```
  monitors[4] <- mnModel(filename="output/placmass.log", printgen=10, separator = " ")
  ```

We can finally create a mcmc, and run it for a good 100 000 cycles:

```
mymcmc <- mcmc(mymodel, monitors, moves)
mymcmc.burnin(generations=100,tuningInterval=100)
mymcmc.run(100000)
```

Exercises

- run the model, check convergence and obtain a sample from the posterior distribution
- using `Tracer`, visualize the posterior distribution on ancestral placental body mass
- calculate the 95% credible interval for ancestral body mass
- calculate the 95% credible interval for the rate of evolution of the log of body mass (σ)

9.4 Correlated evolution of multiple traits

Next, we would like to estimate the correlation between the $K = 3$ life-history traits given in the **`plac40lhtlog.nex`** file, while properly taking into account phylogenetic inertia. To do so, we will assume that the traits jointly evolve along the phylogeny as a *multivariate* Brownian process. We will estimate the *covariance matrix* of this process and assess the empirical support in favor of positive or negative correlations between pairs of traits in terms of posterior probabilities of having positive or negative entries in this covariance matrix. At this stage of the tutorial, we will again ignore phylogenetic uncertainty.

9.4.1 The model and the priors

A multivariate Brownian process $X(t)$, of dimension K (here $K = 3$), is entirely parameterized by its starting value ($X(0)$ at the root of the phylogeny, which a vector of dimension K) and a $K \times K$ symmetric positive matrix (the covariance matrix), which we will call Σ. A positive entry between two traits, say $\Sigma_{12} > 0$, means that when trait 1 increases, trait 2 also tends to increase. Conversely, a negative entry means that the two traits tend to undergo variation in opposite directions. As for the diagonal entries (e.g. Σ_{11}), they represent the variance per unit of time (i.e. the rate of evolution) of each trait considered marginally, thus very much like σ^2 (*not* σ) in the univariate model of the previous section.

On Σ, we will assume an inverse-Wishart prior:

$$\Sigma \sim W^{-1}(\Sigma_0, d),$$

where Σ_0 is a multiple of the identity matrix (i.e. $\Sigma_0 = \kappa I_K$), for some positive real number κ. Using a prior centered on a diagonal matrix means that we want to be indifferent with respect to either positive or negative correlations among traits. As for the parameter κ, it will set the amplitude of the variation per unit of time of the traits. Since we have no idea about the scale of this parameter, we can use a log-uniform prior:

$$\kappa \sim \frac{1}{\kappa}.$$

This completes our model:

$$\begin{aligned} \kappa &\sim \frac{1}{\kappa}, \\ \Sigma \mid \kappa &\sim W^{-1}(\Sigma_0 = \kappa I_K,\, d = K+2), \\ X(0) &\sim \text{Uniform}, \\ X(t) \mid X(0), \Psi, \Sigma &\sim \text{Brownian}(X(0),\, \psi,\, \Sigma). \end{aligned}$$

As in the univariate case, we can then clamp X at the tips of the phylogeny and sample from the joint distribution over the parameters of the model by MCMC. Once this is done, we can estimate marginal posterior probabilities (e.g. for positive or negative covariance among traits) and infer ancestral traits.

Programming the model in RevBayes

You may find it convenient to program this multivariate model by first duplicating the script of the univariate model:

```
cp placentaliaMass.Rev placentaliaTraits.Rev
```

Then, you can edit the new script, **placentaliaTraits.Rev**, and introduce the modifications that would change the univariate model into its multivariate counterpart.

In the following, only those aspects of the multivariate model that differ from the univariate case are outlined. Essentially, instead of a univariate Brownian motion parameterized by a scalar parameter, you now need to:

- define κ:
  ```
  kappa ~ dnLogUniform(min=0.001,max=1000)
  ```
- define the number of degrees of freedom as $d = K + 2$:
  ```
  df <- nTraits+2
  ```
- define the covariance matrix Σ as inverse Wishart:
  ```
  Sigma ~ dnInvWishart(dim=nTraits, kappa=kappa, df=df)
  ```
- define the multivariate Brownian process:
  ```
  X ~ dnBrownianMultiVariate(psi,Sigma)
  ```
- condition the Brownian model on quantitative trait data. This needs to be done separately for each trait:

`for (i in 1:nTraits) { X.clampAt(contData,i,i) }`
Here, we give twice the index `i` to the `clampAt` function: the first corresponds to the entry of the Brownian process, and the second one to the column of the data matrix. In some cases (as we will see below), the Brownian process and the data matrix may not be of same dimension, and therefore, it will be useful to be able to specify arbitrary maps between them.

The model is now entirely specified. We can define the moves on its parameters.

- initialize a running index for storing moves:
  `index <- 1`
- push a scaling move on κ:
  ```
  moves[index] <- mvScale(kappa, lambda=2.0, tune=true, weight=3.0)
  index <- index + 1
  ```
- a sliding move on the Brownian process
  ```
  moves[index] <- mvMultivariateRealNodeValTreeSliding(process=X, lambda=10,
  tune=true,weight=100)
  index <- index + 1
  ```
- a global translation move on the Brownian process (component-wise, that is, a random global translation across the entire phylogeny is applied to one trait taken at random):
  ```
  moves[index] <- mvMultivariateRealNodeValTreeTranslation(process=X, lambda=1,
  tune=true, weight=1)
  index <- index + 1
  ```
- finally, a conjugate Gibbs move for Σ: as it turns out, conditional on κ and the Brownian process X, it is possible to directly resample Σ from its conditional posterior distribution (?). In RevBayes, this is implemented as follows:
  ```
  moves[index] <- mvConjugateInverseWishartBrownian(sigma=Sigma, process=X,
  kappa=kappa, df=df, weight=1)
  index <- index + 1
  ```

Before creating the model, we need to define a few summary statistics, which we want to track during MCMC, either to monitor convergence or for obtaining interesting outputs. First, suppose you are specifically interested in the covariance and the correlation coefficient associated with the joint variation of body-size (trait 2) and longevity (trait 3). You may also be interested in the *partial* correlation coefficient between body mass and longevity, i.e. while controlling for variation in age at sexual maturity. These three quantities can be singled out and named as follows:

- the covariance:
  `cov23 := Sigma.covariance(2,3)`
- the correlation coefficient:
  `cor23 := Sigma.correlation(2,3)`
- the variance per unit of time of, say, log body mass, which is given by the diagonal entry:
  `var2 := Sigma.covariance(2,2)`

- we can also get all correlation coefficients into a single vector (you can skip this part during the session and leave it as homework):

  ```
  # initialize a running index
  corrindex <- 1
  # loop over all pairs of traits
  for (i in 1:(nTraits-1))    {
      for (j in (i+1):nTraits) {
          correl[corrindex] := Sigma.correlation(i,j)
          corrindex <- corrindex + 1
      }
  }
  ```

- we could be interested in tracking several summary statistics also for the Brownian motion, in particular the mean along the tree, separately for each trait:

  ```
  for (i in 1:nTraits)    {
          meanX[i] := X.mean(i)
  }
  ```

After creating the model, all these new variables (**cor12**, **correl**, **meanX**, etc) can be monitored, along with the other parameters of the model:

- create the model
  mymodel <- model(kappa)
- make a screen monitor that tracks correlation coefficients and mean Brownian values:
  monitors[1] <- mnScreen(printgen=10, correl, meanX)
- a file monitor that does the same thing, but directly into a file:
  monitors[2] <- mnFile(filename="output/plactraits.trace", printgen=10, separator = " ", correl, meanX)
- a file monitor for Σ:
  monitors[3] <- mnFile(filename="output/plactraits.cov", printgen=10, separator = " ", Sigma)
- a file monitor for the ancestral reconstruction of traits:
  monitors[4] <- mnFile(filename="output/plactraits.traits", printgen=10, separator = " ", X)
- and a general model monitor:
  monitors[5] <- mnModel(filename="output/plactraits.log", printgen=10, separator = " ")

We can finally create the mcmc and run it:
mymcmc <- mcmc(mymodel, monitors, moves)
mymcmc.burnin(generations=100,tuningInterval=100)
mymcmc.run(100000)

Exercises

- using **`Tracer`**, visualize the posterior distribution on the correlation coefficient between mass and longevity.
- estimate the posterior mean, median and 95% credible interval for this correlation coefficient.
- does the credible inrerval overlap 0? What does that say about the empirical support for the correlation between body mass and longevity?
- what proportion of the variation in longevity among placental mammals is explained by body mass?

9.5 Accounting for uncertainty in divergence times

Starting from the model implemented in the last section, we now want to account for phylogenetic uncertainty. As first pointed out by **?**, 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 (here, with a uniform prior, but this could be a birth death or anything else)
- 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
- the substitution model is conditioned on the molecular sequence data.

Instead of remaining fixed to a pre-defined value, the tree should now be moved during the MCMC. Ideally, we would like to move both the toplogy and the divergence times. Mixing over tree topologies under a Brownian model is relatively challenging, however (it works, but it requires rather long MCMC runs). For that reason, in the following, we will mix over divergence times only, under the constraint of a fixed tree topology. The features of the model that would need to be modified in order to also mix over topologies will nevertheless be indicated. You may want try them after the workshop.

9.5.1 Programming the model in RevBayes

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):

- instead of having a fixed tree, we should now define a **`random`** tree. We could use a birth death prior, whose speciation and extinction rates are themselves endowed with some diffuse exponential prior:
  ```
  speciation ~ dnExp(0.1)
  extinction ~ dnExp(0.1)
  sampling_fraction := 0.01     # 40 out of the ~4000 placental mammals
  psi ~ dnBDP(lambda=speciation, mu=extinction, rho=sampling_fraction, rootAge=1.0, nTaxa=nTax
  names=names)
  ```

- we still want to work on a fixed, pre-specified, tree topology (thus, the birth-death prior will be used here only for averaging over uncertainty about divergence times):
```
treeArray <- readTrees("data/chronoplac40.tree")
fixedTree <- treeArray[1]
psi.setValue(fixedTree)
```
- create a substitution model, just like what you probably did in previous sessions. In a first step, you can use a simple GTR model, without any rate variation, neither among sites nor among branches.
- load the sequence data matrix specified in `data/plac40_4fold.nex` and condition (or clamp) the substitution model to this dataset.
- in the moves section, you should add moves for divergence times:
```
moves[index] <- mvSubtreeScale(psi, weight=5.0)
index <- index + 1
moves[index] <- mvNodeTimeSlideUniform(psi, weight=10.0)
index <- index + 1
```
- you would add topology moves here (again, only in a second step):
```
moves[index] <- mvNNI(psi, weight=5.0)
index <- index + 1
moves[index] <- mvFNPR(psi, weight=5.0)
index <- index + 1
```
- finally, you should add moves for the parameters of the substitution model.

Note that, here, we do not have included any fossil information: we are merely doing *relative* dating. We will see at the end of this tutorial how fossil information can be integrated.

Write this model and make sure that it runs when you give it to `RevBayes`. Once this is the case, don't spend too much time analyzing the results and quickly turn to the model introduced in the next section.

9.6 Autocorrelated relaxed molecular clock

In the previous model, no consideration was given to the problem of rate variation among lineages – we bluntly used a strict clock. This is of course problematic, in particular at the phylogenetic scale considered here (placental mammals), where we know that there is substantial rate variation. In addition, we know that substitution rates across branches are *auto-correlated* in the present case: typically, entire orders, such as rodents, are fast evolving, whereas other orders like Cetartiodactyla are slowly evolving. In other words, nearby lineages along the phylogeny tend to be characterized by similar substitution rates.

You have perhaps already seen an autocorrelated relaxed clock model in the molecular dating session (ACLN). You could easily recruit it in the present context (a good exercise to try after the workshop: modify the model suggested in the previous section so as to replace the strict clock by the ACLN model).

Here, however, we will derive the autocorrelated clock in a slightly different way. This derivation will be less straightforward, but more useful for what we want to do next. Specifically, we will first model the logarithm of the instant substitution rate as a Brownian motion, just like we did for body mass in section 9.3. Then, we will exponentiate this Brownian process and take branch-specific averages, which we will finally plug into the substitution model as the `branchRates` argument.

9.6.1 Programming the model in RevBayes

Compared to the model described in the last section, you should:

- delete the `clockRate` variable
- based on what you have done with body mass in section 9.3, you should be able to create a univariate Brownian process, which you could call `lograte`
- you can then exponentiate and average this Brownian process over branches using `expBranchTree`: `branchrates := expBranchTree(tree=psi, process=lograte)`
- plug these rates into the substModel object, as the branchRates parameter vector.
- condition the model on the sequence and trait data and run the program.

Again, write this model by duplicating and adapting the last script that you have written. Make sure that the model runs when given to `RevBayes`, before turning to the next model now introduced.

9.7 Rates, dates and traits

We have just seen that the logarithm of the substitution rate can be seen as a quantitative trait. But then, this raises one further obvious question: why considering the substitution rate and the quantitative traits as separate Brownian motions? Why not instead considering them as a joint multivariate Brownian process? Doing so would have one major advantage: the correlated evolution of rates and traits will be automatically estimated, as a by-product of the model.

To do so, we just need to define a multivariate Brownian process of dimension 4. By convention, we will consider that the first dimension of this process corresponds to the log of the substitution rate, while the other 3 dimensions of the process (2 to 4) will map to the quantitative traits defined by the data matrix (?).

9.7.1 Programming the model in RevBayes

You now have all the tools to implement this model entirely by yourself, except for one little detail: you now need to exponentiate one specific component of a multivariate process (as opposed to exponentiating a univariate process, as we did in the previous section). Thus, assuming that `X` is your 4-dimensional process, you need to tell the `expBranchTree` function that you want to exponentiate the first component of the process (with the `traitIndex=1` option):

```
branchrates := expBranchTree(tree=psi, process=x, traitIndex=1)
```

Also, be careful with the mapping of the quantitative traits: you need to map trait i to entry $i+1$ of the Brownian process:

```
for (i in 1:nTraits) { X.clampAt(contData,i+1,i) }
```

9.7.2 Exercises

- write the model and run it on the placental example
- investigate the correlation between substitution rate and life-history traits

- multiple regression: controlling for body mass, do you still get some support for a correlation between longevity and substitution rate variation?
- conversely, controlling for longevity, do you get supported correlations of the substitution rate and body mass (or with age at sexual maturity?)
- compare the credible interval obtained here for the body mass of the last common ancestor of placentals with what was obtained in the very first model (section 9.3).

9.8 A comparative analysis of variation in GC content

Apart from the overall substitution rate, any other aspect of the substitution process (transition-transversion ratio, dN/dS, equilibrium frequencies, etc) could in principle display variation among lineages. These various aspects of the substitution process could therefore be modeled exactly like the substitution rate, i.e. as Brownian processes – or as components of a multivariate Brownian process. In this section, we will focus on compositional variation, and more particularly variation in equilibrium GC content between species.

We first start with a simple T92 model of sequence evolution:

$$Q = \left(\begin{array}{c|cccc} & A & C & G & T \\ \hline A & - & \frac{\gamma}{2} & \kappa\,\frac{\gamma}{2} & \frac{1-\gamma}{2} \\ C & \frac{1-\gamma}{2} & - & \frac{\gamma}{2} & \kappa\,\frac{1-\gamma}{2} \\ G & \kappa\,\frac{1-\gamma}{2} & \frac{\gamma}{2} & - & \frac{1-\gamma}{2} \\ T & \frac{1-\gamma}{2} & \kappa\,\frac{\gamma}{2} & \frac{\gamma}{2} & - \end{array}\right)$$

The model has two parameters: the transition-transversion rate κ and the equilbrium GC content γ. In the following, we will assume that κ is constant across the tree (although unknown, and thus endowed with a diffuse prior). In contrast, γ will be allowed to vary among lineages, jointly with the overall substitution rate. Technically, since γ is strictly between 0 and 1, its log-it transform $\ln\frac{\gamma}{1-\gamma}$ will range over the entire real line. Therefore, we could propose that the log-it transform of γ evolves according to a Brownian motion.

Putting everything together, we will therefore propose a multivariate Brownian motion $X(t)$, of dimension $K+2$, where K is the number of quantitative traits, and such that:

$$\begin{aligned} X_1(t) &= \ln r(t) \\ X_2(t) &= \ln \frac{\Gamma(t)}{1-\Gamma(t)} \\ k = 1..K, \quad X_{k+2}(t) &= \ln C_k(t) \end{aligned}$$

where $r(t)$ is the instant substitution rate and $\gamma(t)$ the instant equilibrium GC composition and $C_k(t)$ is the kth. quantitative trait. Equivalently, we may re-write this as follows:

$$\begin{aligned} r(t) &= e^{X_1(t)} \\ \gamma(t) &= \frac{e^{X_2(t)}}{1+e^{X_2(t)}} \\ \dots \end{aligned}$$

In other words, the instant rate of substitution $r(t)$ is the exponential of the first component $X_1(t)$ of the Brownian process (as above), while the instant equilibrium GC $\gamma(t)$ is the *hyperbolic tangent* of the second component $X_2(t)$ of the Brownian process.

There is a slight complication here: in a non-homogeneous model, independently of the rate matrices across branches, we also need to specify the nucleotide frequencies from which the sequence at the root of the tree is sampled. We will call this frequency vector π, and we will put a Dirichlet prior on π.

This model has been described in **?**.

Programming the model in RevBayes

Assuming that `X` is the multivariate Brownian process:

- as above, define the branch rates as the exponential of the first component:
  `branchrates := expBranchTree(tree=psi, process=X, traitIndex=1)`
- define the branch equilibrium GC as the hyperbolic tangent of the second component:
  `branchGC := tanhBranchTree(tree=psi, process=X, traitIndex=2)`
- for $k = 1..K$, map trait k onto entry $k + 2$ of X:
  `for (k in 1:nTraits) { X.clampAt(contData,k+2,k) }`
- define the transition-transversion ratio; usually, this ratio is of the order of 1-10, so we will use an exponential prior of mean 10:
  `tstv ~ dnExp(0.1)`
- define a vector of branch-specific T92 substitution matrices:
  `branchMatrices := t92GCBranchTree(tree=psi,branchGC=branchGC,tstv=tstv)`
- create a Dirichlet-distributed vector of equilbirium frequencies over nucleotides:
  ```
  bf <- v(1,1,1,1)
  pi ~ dnDirichlet(bf)
  ```
- finally, create the substitution model:
  ```
  seq ~ dnPhyloCTMC(tree=psi, Q=branchMatrices, rootFrequencies=pi,
  branchRates=branchrates, nSites=nSites, type="DNA")
  ```

Exercises

- program the model in RevBayes
- run the model on the placental dataset
- investigate the correlation between GC and body mass
- how do you explain these correlations?
- run the model on the archaeal rRNA dataset `archaea.nex`, using temperature as the trait. In that case, no phylogeny is provided, so you may try to run the model without any constraint on the topology.

- assess the correlation between GC and temperature
- how much of the variation in GC is explained by temperature?
- what could be the underlying biological cause?

9.9 Towards integrative macro-evolution modeling

The modeling approach proposed above is just one example of the integration of multiple domains of macroevolutionary studies that could be done with RevBayes. In the following, we outline some possible extensions or variations, based on the integrative modeling philosophy.

9.9.1 Using fossil data

Fossils have much to say about several aspects of the models and questions we have considered thus far. They represent a valuable source of information about divergence times but also about ancestral traits. In particular, fossil calibrations could be used in the context of each of the models that have been considered in sections 3 to 6, thus allowing us to do not just relative, but absolute, dating. In principle, most of the approaches that you have seen during the dating session of the workshop could be adapted to the present situation. It is just a matter of gathering the relevant information about mammalian fossils.

More ambitiously, the *total evidence dating* method (**?**) could be extended so as to now include, not just discrete morphological characters, but also quantitative traits, for both extant and extinct taxa. Morphological characters would be modeled using discrete M_k models, while quantitative traits would be described by multivariate Brownian processes, just as in the previous sections.

9.9.2 Beyond Brownian models

Throughout this tutorial, we have exclusively considered undirected Brownian models. However, many other models could be used, and this, both for quantitative traits and for substitution rates or substitution parameters. Right now, there are at least two other models available in RevBayes: the Brownian model with systematic trend and the Ornstein-Uhlenbeck process.

One possible application of the Brownian model with trend would be to test for the existence of a systematic trend in increasing body size (i.e. Cope's rule) during animal, vertebrate or mammalian evolution (**?**). Note, however, that systematic trends cannot be estimated using only extant taxa (at least using purely anagenetic processes of evolution, such as considered here): the model would not be identifiable. If we have fossil data, on the other hand, we can estimate a trend: the model will then essentially rely on the average-mass-through-time distribution across the entire geological range.

Technically, to model body size evolution with drift, we would just need to:

- define a drift parameter, with a diffuse prior centered on 0:
  `copestrend ~ dnNorm(0,10)`
- create a (univariate) Brownian motion with drift:
  `logmass ~ dnBrownian(psi,sigma,drift=copestrend)`
- move the trend parameter during the MCMC, using a regular sliding move:
  ```
  moves[index] <- mvSlide(copestrend, delta=2.0, tune=true, weight=3.0)
  index <- index + 1
  ```

After running the model, the posterior distribution on Cope's trend parameter can be visualized and quantified, and the empirical support in favor of Cope's rule can be assessed by estimating the posterior probability that this trend parameter is positive.