Phylogenetic Inference using RevBayes

Relaxed-Clocks & Calibrated Time Trees

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

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 and fossil calibration 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`.

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 data and output files from: http://bit.ly/1oplDTb

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 1).

Table 1: Fossil species used for calibrating divergence times in the caniform tree.

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

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

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.

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 1. This part of the exercise will involve specifying a birth-death model with clamped stochastic nodes representing the observation times of two fossils descended from internal nodes in our tree: (1) *Parictis montanus*, the oldest fossil in the family Ursidae, a stem fossil bear, and (2) *Kretzoiarctos beatrix*, the fossil Ailuropodinae, a crown fossil bear. Additionally, we will use the canid fossil, *Hesperocyon gregarius*, to offset the age of the root of the tree.

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

In `RevBayes`, calibration densities are applied in a different way, treating fossil observation times like data. The graphical model in Figure 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. 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. 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

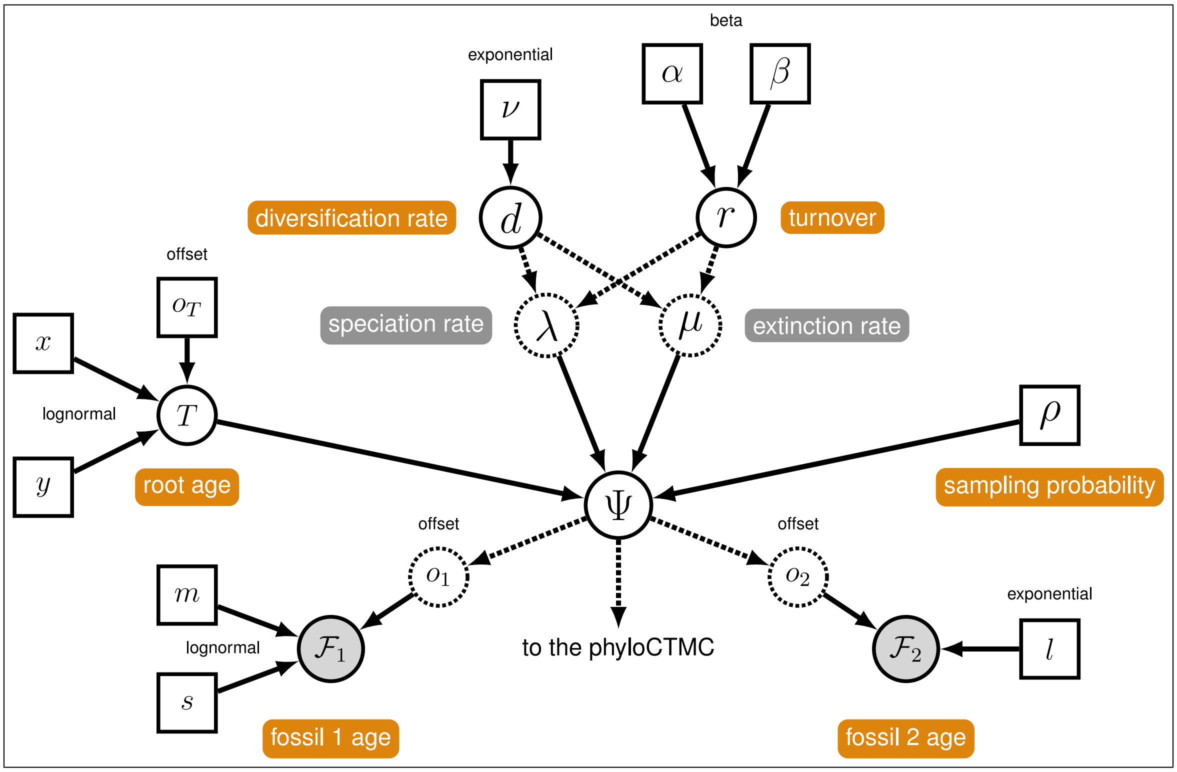

Figure 1: The graphical model representation of the node-calibrated birth-death process in `RevBayes`.

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()
```

1.4.1 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 `RevBayes`, 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")`**.

Diversification

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

Turnover

```
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
```

1.4.2 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 ~ dnLnorm(mu_ra, stdv_ra, offset=tHesperocyon)
```

1.4.3 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)
```

1.4.4 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)
```

1.4.5 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.

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.

1.5.1 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. 2).

Create the Rev File

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.

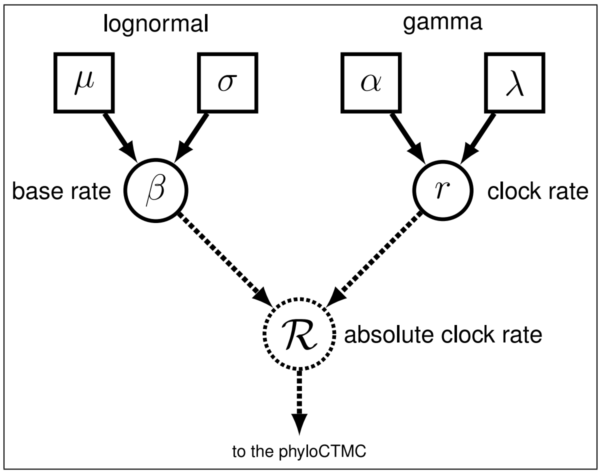

Figure 2: The graphical model representation of the global molecular clock model used in this exercise.

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 2.

1.5.2 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. 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 3, we can represent the vector of N branch rates using the plate notation. Additionally, each branch rate is rescaled by the base rate.

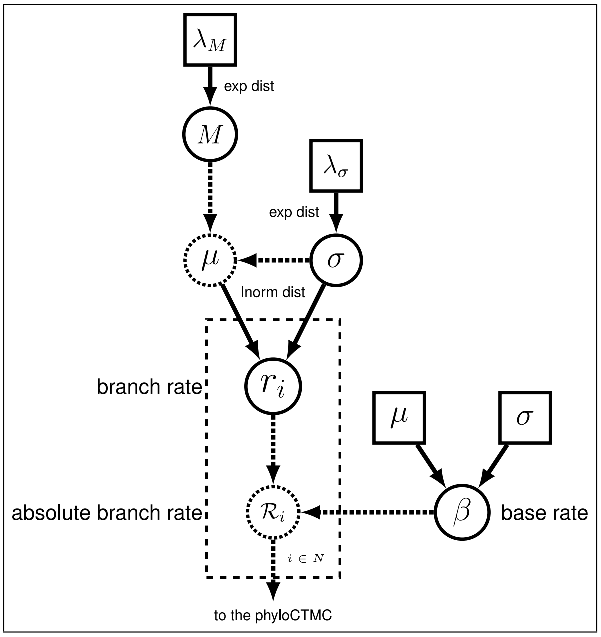

Figure 3: The graphical model representation of the UCLN model used in this exercise.

Create the Rev File

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

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

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 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 2.

1.5.3 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 et al. 1998; Kishino et al. 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

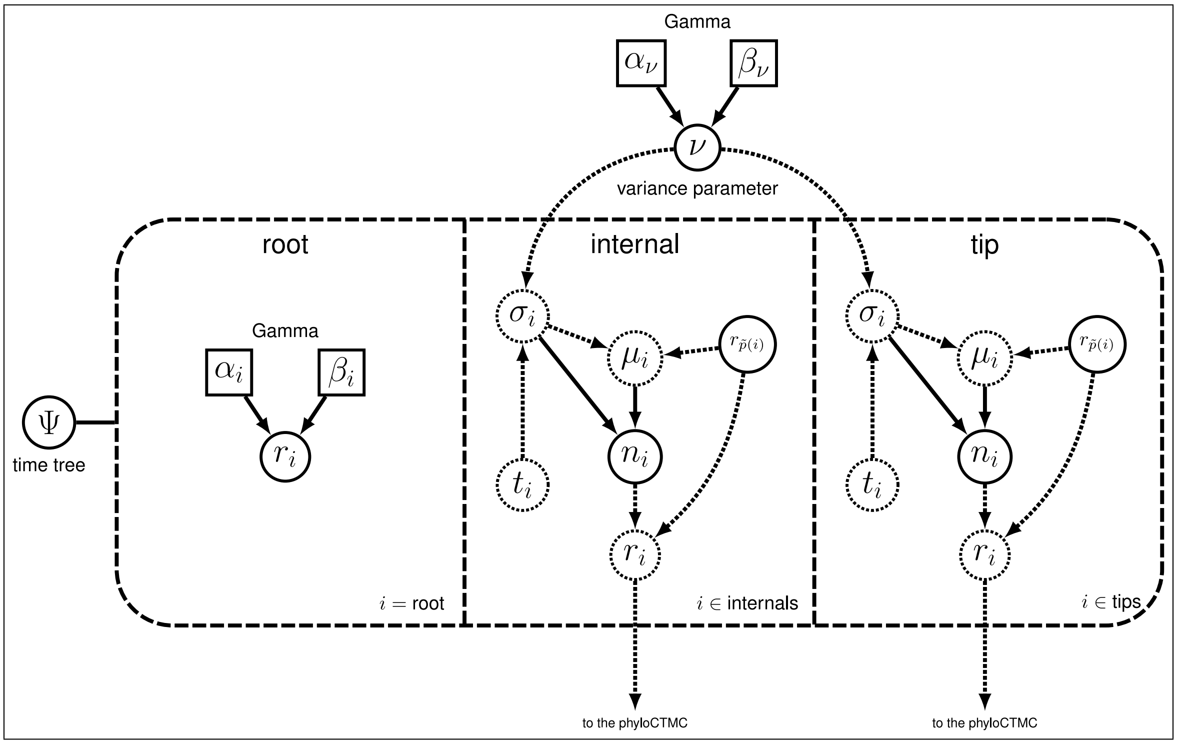

Figure 4: The graphical model representation of the ACLN model used in this exercise.

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 4.

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 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 et al. 1998; Kishino et al. 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.

```
physSeq ~ dnPhyloCTMC(tree=timetree, Q=Q, branchRates=branch_subrates, nSites
   =n_sites, type="DNA")
attach the observed sequence data
physSeq.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 far too slow under this model, thus it may be advisable for you to decrease the number of steps (`cats`) or the length of the burn-in period or run.

Once you have completed this analysis, record the marginal likelihoods under the ACLN model in Table 2.

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

Table 2: Estimated marginal likelihoods for different partition configurations*.

	Marginal lnL estimates	
Partition	*Stepping-stone*	*Path sampling*
Global molecular clock (M_0)		
Uncorrelated lognormal (M_1)		
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 $\mathcal{K}$):

$$\mathcal{K} = \ln[BF(M_0, M_1)] = \ln[\mathbb{P}(\mathbf{X} \mid M_0)] - \ln[\mathbb{P}(\mathbf{X} \mid M_1)], \tag{1}$$

where $\ln[\mathbb{P}(\mathbf{X} \mid M_0)]$ is the *marginal lnL* estimate for model M_0. The value resulting from equation 1 can be converted to a raw Bayes factor by simply taking the exponent of $\mathcal{K}$

$$BF(M_0, M_1) = e^{\mathcal{K}}. \tag{2}$$

Alternatively, you can interpret the strength of evidence in favor of M_0 using the $\mathcal{K}$ and skip equation 2. In this case, we evaluate the $\mathcal{K}$ in favor of model M_0 against model M_1 so that:

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

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

Using the values you entered in Table 2 and equation 1, calculate the ln-Bayes factors (using $\mathcal{K}$) for the different model comparisons. Enter your answers in Table 3 using the stepping-stone and the path-sampling estimates of the marginal log likelihoods.

Table 3: Bayes factor calculation*.

	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

1.7 Estimate the Topology and Branch Times

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

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

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

```
### Load the sequence alignment
D <- readDiscreteCharacterData(file="data/bears_irbp.nex")

### get helpful variables from the data
n_sites <- D.nchar(1)

### initialize an iterator for the moves vector
mi = 1
```

This is how you should begin your MCMC analysis file. The next step is to source the birth-death model. However, if you're interested in estimating the tree topology, then you must add proposals that will do this. These moves can be added right after the birth-death model is sourced.

```
### set up the birth-death model from file
source("RevBayes_scripts/m_BDP_bears.Rev")

### and moves for the tree topology
moves[mi++] = mvNNI(timetree, weight=8.0)
moves[mi++] = mvNarrow(timetree, weight=8.0)
moves[mi++] = mvFNPR(timetree, weight=8.0)
```

Next load the file containing your favorite model (where the wildcard **`*`** indicates the name of the model you prefer: **`GMC`**, **`UCLN`**, or **`ACLN`**).

```
### load the model from file
source("RevBayes_scripts/m_*_bears.Rev")

### workspace model wrapper ###
mymodel = model(er)
```

MCMC Monitors

Before you instantiate the MCMC workspace object, you need to create a vector of "monitors" that are responsible for monitoring parameter values and saving those to file or printing them to the screen.

First, create a monitor of all the model parameters except the **`timetree`** using the model monitor: **`mnModel`**. This monitor takes *all* of the named parameters in the model DAG and saves their value to a file. Thus, every variable that you gave a name in your model files will be written to your log file. This makes it very easy to get an analysis going, but can generate very large files with a lot of redundant output.

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

If the model monitor is too verbose for your needs, you should use the file monitor instead: **`mnFile`**. For this monitor, you have to provide the names of all the parameters you're interested in after the file name and print interval. (Refer to the example files for how to set up the file monitor for model parameters.)

In fact, we use the file monitor for saving the sampled chronograms to file. It is important that you *do not* save the sampled trees in the same file with other numerical parameters you would like to summarize. That is because tools for reading MCMC log files—like Tracer (Rambaut and Drummond 2009)—cannot load files with non-numerical states. Therefore, you must save the sampled trees to a different file.

```
monitors[2] = mnFile(filename="output/TimeTree_bears_mcmc.trees", printgen
   =10, timetree)
```

Finally, we will create a monitor in charge of writing information to the screen: **`mnScreen`**. We will report the root age 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/code
- 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: tracyh@berkeley.edu)
- Tanja Stadler (email: tanja.stadler@bsse.ethz.ch)
- Sebastian Höhna (email: sebastian.hoehna@gmail.com)

Version dated: November 11, 2014

Relavent References

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

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

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

dos Reis, M, J Inoue, M Hasegawa, R Asher, P Donoghue, and Z Yang. 2012. Phylogenomic datasets provide both precision and accuracy in estimating the timescale of placental mammal phylogeny. *Proceedings of the Royal Society B: Biological Sciences* 279: 3491–3500.

Heath, TA, JP Huelsenbeck, and T Stadler. 2014. The fossilized birth-death process for coherent calibration of divergence-time estimates. *Proceedings of the National Academy of Sciences, USA* 111: E2957–E2966.

Heled, J and AJ Drummond. 2012. Calibrated tree priors for relaxed phylogenetics and divergence time estimation. *Systematic Biology* 61: 138–149.

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

Krause, J, T Unger, A Noçon, AS Malaspinas, SO Kolokotronis, M Stiller, L Soibelzon, H Spriggs, PH Dear, AW Briggs, et al. 2008. Mitochondrial genomes reveal an explosive radiation of extinct and extant bears near the Miocene-Pliocene boundary. *BMC Evolutionary Biology* 8: 220.

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

Rannala, B and Z Yang. 2003. Bayes estimation of species divergence times and ancestral population sizes using DNA sequences from multiple loci. *Genetics* 164: 1645–1656.

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

Thorne, J, H Kishino, and IS Painter. 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). Bulletin of the AMNH.* Vol. 221.

Wang, X, RH Tedford, and BE Taylor. 1999. *Phylogenetic Systematics of the Borophaginae (Carnivora, Canidae). Bulletin of the AMNH.* Vol. 243.

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

Yang, Z and B Rannala. 2006. Bayesian estimation of species divergence times under a molecular clock using multiple fossil calibrations with soft bounds. *Molecular Biology and Evolution* 23: 212–226.