

Statistical Phylogenetic Inference using RevBayes

Sebastian Höhna, Michael J. Landis, Tracy A. Heath, Bastien Boussau, Will A. Freyman, Will Pett, April Wright, Mike R. May, Nicolas Lartillot, Brian R. Moore, Fredrik Ronquist, and John P. Huelsenbeck

July 19, 2017

Contents

Preface	xv
I Getting Started with RevBayes	1
1 Introduction	3
1.1 Overview	4
1.2 Getting Started	4
1.3 Probabilistic Graphical Models	5
1.4 Rev: The RevBayes Language	6
1.4.1 Specifying Models	7
1.5 Getting help in RevBayes	9
1.5.1 RevBayes Users' Forum	10
2 Introduction to RevBayes and Rev	11
2.1 Basic Rev Commands	12
2.1.1 Introduction	12
2.1.2 Variable Declaration	14
2.2 Exercise: Poisson Regression Model for Airline Fatalities	22
2.2.1 Model and Data	22
2.2.2 Problems	22
2.3 Exercise: Poisson Regression Model for Coal-mine Accidents	28

2.3.1	A model for disasters	28
2.3.2	Batch Mode	29
3	Reading and manipulating data	31
3.1	Overview	32
3.2	Molecular Sequence Data	32
3.2.1	Getting Started	32
3.2.2	Loading Molecular Sequence Data	33
3.2.3	Querying Dataset Attributes	33
3.2.4	Concatenating Sequences	33
3.2.5	Excluding/Including Taxa	34
3.2.6	Excluding/Including Sites or Genes	35
3.3	Biogeographical Data	36
3.3.1	Nexus file	36
3.3.2	Atlas file	37
II	Inference	39
4	Introduction to Markov chain Monte Carlo Algorithms	41
4.1	Overview	42
4.2	A Coin Flipping (Binomial) Model	42
4.3	Writing an MCMC from Scratch	42
4.3.1	The Metropolis-Hastings Algorithm	43
4.3.2	Reading in the data	44
4.3.3	Initializing the Markov chain	44
4.3.4	Writing the MH Algorithm	45
4.3.5	Exercise 1	46
4.4	More on Moves: Tuning and weights	46

4.4.1	Uniform move	47
4.4.2	Sliding-window move	48
4.4.3	Scaling move	49
4.4.4	Exercise 2	50
4.5	The Metropolis-Hastings Algorithm with the <i>Real RevBayes</i>	51
4.5.1	Exercise 3	52
4.5.2	Exercise 4	53
III	Basic Phylogeny Estimation	55
5	Continuous Time Markov Model for Discrete Character Evolution	57
5.1	Overview	58
5.1.1	Requirements	58
5.2	Data and files	58
5.3	Example: Character Evolution under the Jukes-Cantor Substitution Model	59
5.3.1	Getting Started	59
5.3.2	Loading the Data	60
5.3.3	Jukes-Cantor Substitution Model	61
5.3.4	Tree Topology and Branch Lengths	62
5.3.5	Putting it All Together	65
5.3.6	Performing an MCMC Analysis Under the Jukes-Cantor Model	66
5.3.7	Exercise 1	67
5.4	The Hasegawa-Kishino-Yano (HKY) 1985 Substitution Model	70
5.4.1	Exercise 2	72
5.5	The General Time-Reversible (GTR) Substitution Model	73
5.5.1	Exercise 3	75
5.6	The Discrete Gamma Model of Among Site Rate Variation	76

5.6.1	Setting up the Gamma Model in RevBayes	77
5.6.2	Exercise 4	78
5.7	Modeling Invariable Sites	79
5.7.1	Exercise 5	79
IV Model Selection, Model Averaging, and Model Testing		81
6	Model Selection and Bayes Factors	83
6.1	Overview	84
6.1.1	Requirements	84
6.2	Data and files	84
6.3	Introduction	84
6.3.1	Substitution Models	86
6.3.2	Estimating the Marginal Likelihood	86
6.3.3	Exercises	89
6.4	Computing Bayes Factors and Model Selection	90
6.5	For your consideration...	91
6.5.1	Accommodating Model Uncertainty	91
6.5.2	Assessing Model Adequacy	91
V More on Substitution Models		93
7	Partitioned Data Analysis	95
7.1	Overview	96
7.1.1	Requirements	96
7.2	Data and files	97
7.3	Introduction & Background	97
7.4	Concatenated, Non-partitioned	98

7.4.1	Setting up the model	98
7.4.2	Putting it All Together	102
7.5	Partitioning by Gene Region	104
7.5.1	Specify the Parameters by Looping Over Partitions	105
7.5.2	Putting it all together	108
7.5.3	Create monitors	108
7.6	Partitioning by Codon Position and by Gene	109
7.6.1	Exercises	110
VI	Divergence Time Estimation	113
8	Overview	115
8.1	Background	116
8.1.1	Modeling lineage-specific substitution rates	116
8.1.2	Priors on node times	117
8.1.3	Calibration to absolute time	117
8.1.4	Integrating Fossil Occurrence Times in the Speciation Model	118
9	Node and Fossil Calibration	121
9.1	Overview	122
9.2	Data and files	122
9.3	Divergence time estimation using an informative prior on the root age	123
9.3.1	Getting Started	123
9.3.2	Loading the Data	123
9.3.3	General Time Reversible (GTR) Substitution Model	124
9.3.4	Setting up the Gamma Model	125
9.3.5	Tree Prior: Tree Topology and Node Ages	126
9.3.6	Monitoring specific clade ages	128

9.3.7	The Global Molecular Clock Model	129
9.3.8	Putting it All Together	129
9.3.9	Performing an MCMC Analysis Under the Global Clock Model	130
9.3.10	Exercise 1	131
9.4	Informative node calibration	132
9.4.1	Adding node calibrations	132
9.4.2	Exercise 2	133
9.5	Hard-bounded node calibrations	134
9.5.1	Exercise 3	135
9.6	Soft-bounded node calibrations	136
9.6.1	Exercise 4	137
10	Relaxed Clocks	141
10.1	Overview	142
10.2	Data and files	142
10.3	Divergence time estimation using uncorrelated, lognormal distributed branch rates	143
10.3.1	Getting Started	143
10.3.2	Loading the Data	143
10.3.3	General Time Reversible (GTR) Substitution Model	144
10.3.4	Setting up the Gamma Model	145
10.3.5	Tree Prior: Tree Topology and Node Ages	146
10.3.6	Calibrating node ages	147
10.3.7	The Uncorrelated Lognormal Relaxed Clock Model	149
10.3.8	Putting it All Together	150
10.3.9	Performing an MCMC Analysis Under the Global Clock Model	151
10.3.10	Exercise 1	152
10.4	Random local clock	153

VII Diversification Rate Estimation	155
11 Overview	157
11.1 Overview: Diversification Rate Estimation	158
11.1.1 Types of Hypotheses for Estimating Diversification Rates	158
11.2 Models	159
11.2.1 The birth-death branching process	159
12 Basic Diversification Rate Estimation	165
12.1 Estimating Constant Speciation & Extinction Rates	166
12.1.1 Outline	166
12.1.2 Requirements	166
12.2 Data and files	166
12.3 Pure-Birth (Yule) Model	166
12.3.1 Read the tree	168
12.3.2 Specifying the model	168
12.3.3 Running an MCMC analysis	170
12.3.4 Exercise 1	171
12.4 Estimating the marginal likelihood of the model	171
12.4.1 Exercise 2	173
12.5 Birth-Death Process	173
12.5.1 Diversification and turnover	174
12.5.2 Birth rate and death rate	175
12.5.3 The time tree	175
12.5.4 Exercise 3	175
13 Episodic Diversification Rate Estimation	177
13.1 Estimating Speciation & Extinction Rates Through Time	178

13.1.1	Outline	178
13.1.2	Requirements	178
13.2	Data and files	178
13.3	Episodic Birth-Death Model	178
13.3.1	Read the tree	179
13.3.2	Specifying the model	180
13.3.3	Running an MCMC analysis	184
13.3.4	Exercise 1	185
13.3.5	Exercise 2	187
14	Environmental-correlated Diversification Rate Estimation	189
14.1	Estimating Environmental-dependent Speciation & Extinction Rates	190
14.1.1	Outline	190
14.1.2	Requirements	190
14.2	Data and files	190
14.3	Environmental-dependent Diversification Rates	191
14.3.1	Read the tree	193
14.3.2	Set up the environmental data	193
14.3.3	Specifying the model	194
14.3.4	Running an MCMC analysis	198
14.3.5	A brief discussion on estimated diversification rates	200
14.3.6	Exercise 1	200
14.4	Testing for correlation using reversible-jump MCMC	202
14.4.1	Exercise 2	203
15	Diversification Rate Estimation with Missing Taxa	205
15.1	Estimating Speciation & Extinction Rates Through Time	206
15.1.1	Outline	206

15.1.2 Requirements	206
15.2 Data and files	206
15.3 Episodic Birth-Death Model	207
15.3.1 Specifying the model in Rev	208
15.4 Uniform Taxon Sampling	211
15.4.1 Exercise 1	212
15.4.2 Summarizing and plotting diversification rates through time	212
15.5 Diversified Taxon Sampling	213
15.5.1 Exercise 2	214
15.6 Empirical Taxon Sampling	216
15.6.1 Exercise 3	218
16 Branch-Specific Diversification Rate Estimation	221
16.1 Estimating Branch-Specific Speciation & Extinction Rates	222
16.1.1 Outline	222
16.1.2 Requirements	222
16.2 Data and files	222
16.3 Branch-Specific Birth-Death Model	222
16.4 Testing for Branch-Specific-Diversification Rates	224
16.4.1 Read the tree	224
16.4.2 Specifying the model	225
16.4.3 Running a marginal likelihood estimation	229
16.4.4 Exercise 1	230
16.5 Estimating Branch-Specific Diversification Rates	230
16.5.1 Read the tree	230
16.5.2 Specifying the model	231
16.5.3 Running an MCMC analysis	234

16.5.4 Exercise	236
VIII Gene tree - Species tree estimation	237
17 Overview	239
17.1 Overview: Gene tree-species tree models	240
17.1.1 Processes of discord	240
17.1.2 Gene tree discordance is a problem for species tree reconstruction	240
18 Estimating species tree using gene concatenation	243
18.1 Concatenating genes to model species evolution	244
19 Multispecies Coalescent	249
19.1 Modeling incomplete lineage sorting: the multispecies coalescent	250
19.2 Things to think about	257
IX Biogeography	259
20 Dispersal, Extirpation and Cladogenesis (DEC)	261
20.1 Introduction	262
20.2 Dispersal-Extinction-Cladogenesis model	262
20.2.1 Range characters	262
20.2.2 Modeling anagenic range evolution	262
20.2.3 Modeling cladogenic range evolution	263
20.2.4 Exercises	270
20.2.5 (Advanced) Joint inference of phylogeny and historical biogeography	270
20.3 Epoch models and ancestral range reconstruction	275
20.3.1 Ancestral range reconstruction	275
20.3.2 Data exploration with Phylowood	275

20.3.3 Epoch models	278
21 Many Area DEC	283
21.1 Large numbers of areas	284
21.1.1 Data augmentation	284
21.1.2 Large rate matrices and distance-dependent dispersal	284
21.1.3 Specifying a data augmented DEC model	285
21.1.4 Analysis output	290
21.1.5 Biogeographic event histories from <code>mnCharHistoryNewick</code>	291
21.1.6 Exercises	292
X Phylogenetic Comparative Method	293
22 Phylogenetic Comparative Analyses: Continuous Trait Evolution	295
22.1 Introduction	296
22.2 Data and files	296
22.3 Univariate Brownian evolution of quantitative traits	296
22.3.1 The model and the priors	297
22.3.2 Programming the model in RevBayes	297
22.3.3 Phylogenetic Independent Contrasts using the reduced likelihood (REML)	298
22.3.4 Phylogenetic covariance matrix	301
22.3.5 Data augmentation	302
22.3.6 Brief intermediate summary	304
22.4 Multiple independently evolving traits	305
22.4.1 Independent site rates	306
22.4.2 Partially shared site rates	307
22.4.3 Model selection	308
22.5 Estimating the phylogeny from continuous character data	309

22.6 Including information about the tree from molecular data	311
22.6.1 Programming the model in RevBayes	311

Preface

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

Part I

Getting Started with RevBayes

Chapter 1

Introduction

Overview

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

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

Getting Started

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

- \$ `rb`

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

- \$ `rb my_analysis.Rev`

then `RevBayes` will run all commands specified in your file.

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

- \$ `mpirun -np 4 rb-mpi`

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

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

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

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

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

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

Most tutorials also includes hyperlinks: bibliographic citations are medium sea green and link to the full citation in the references, external URLs are cerulean, and internal references to figures and equations are forest green.

The various exercises in this tutorial take you through the steps required to perform phylogenetic analyses of the example datasets. In addition, we have provided the `Rev` scripts and output files for some exercises 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.)

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 (Gillks et al. 1994; Lunn et al. 2000; Jordan 2004; Koller and Friedman 2009; Lunn et al. 2009). The graphical model framework allows for flexible model specification and implementation and reduces redundant code. This framework provides a set of symbols for depicting a *directed acyclic graph* (DAG). Höhna et al. (2014) described the use of probabilistic graphical models for phylogenetics. The different nodes and components of a phylogenetic graphical model are shown in Figure 1.1 (Fig. 1 from Höhna et al. 2014).

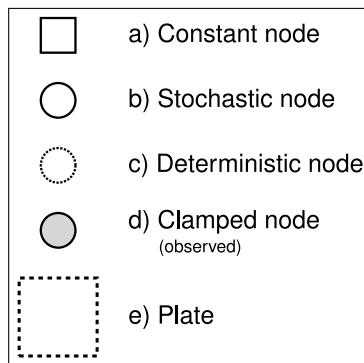


Figure 1.1: The symbols for a visual representation of a graphical model. a) Solid squares represent constant nodes, which specify fixed-valued variables. b) Stochastic nodes are represented by solid circles. These variables correspond to random variables and may depend on other variables. c) Deterministic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

To represent the DAG, nodes are connected with arrows indicating dependency. A simple, albeit abstract,

graphical model is shown in Figure 1.2. In this model, we observe a set of states for parameter x . We assume that the values of x are samples from a lognormal distribution with a location parameter (log mean) μ and a standard deviation σ . It is more straightforward to model our uncertainty in the expectation of a lognormal distribution, rather than μ , thus we place a gamma distribution on the mean M . This gamma hyperprior has two parameters that we specify with fixed values (constant nodes): the shape α and rate β . The variable M is a stochastic node with this prior density. The standard deviation, σ , is also a stochastic node with an exponential prior density with rate parameter λ . For any value of M and any value of σ we can compute the deterministic variable μ using the formula $\mu = \ln(M) - \frac{\sigma^2}{2}$. This formula is known from using simple algebra on the equation for the mean of any [lognormal distribution](#). With this model structure, we can then calculate the probability of the data conditional on the model and parameter values (the likelihood): $\mathbb{P}(\mathbf{x} | \mu, \sigma)$. Next we can get the posterior probability using Bayes' theorem:

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

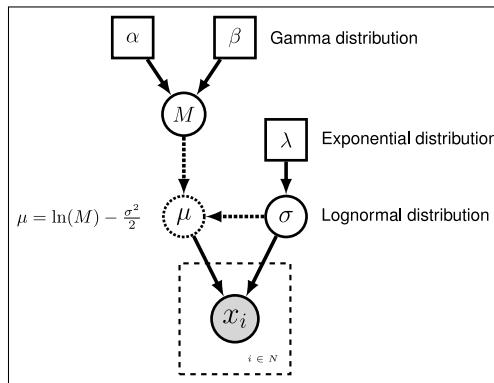


Figure 1.2: Graphical model representation of a simple lognormal model. A total of N observations of variable x are observed and occupy a clamped node. This parameter is log-normally distributed with parameters μ and σ (log mean and standard deviation, respectively). The parameter μ is a deterministic 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 hyperprior 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
<code><-</code>	constant variable
<code>~</code>	stochastic variable
<code>:=</code>	deterministic variable
<code>node.clamp(data)</code>	clamped variable
<code>=</code>	inference (<i>i.e.</i> , non-model) variable
<code>for(i in 1:N){...}</code>	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 hyperparameters 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 ~ dnBimodalLognormal(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 ~ dnBimodalLognormal(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.

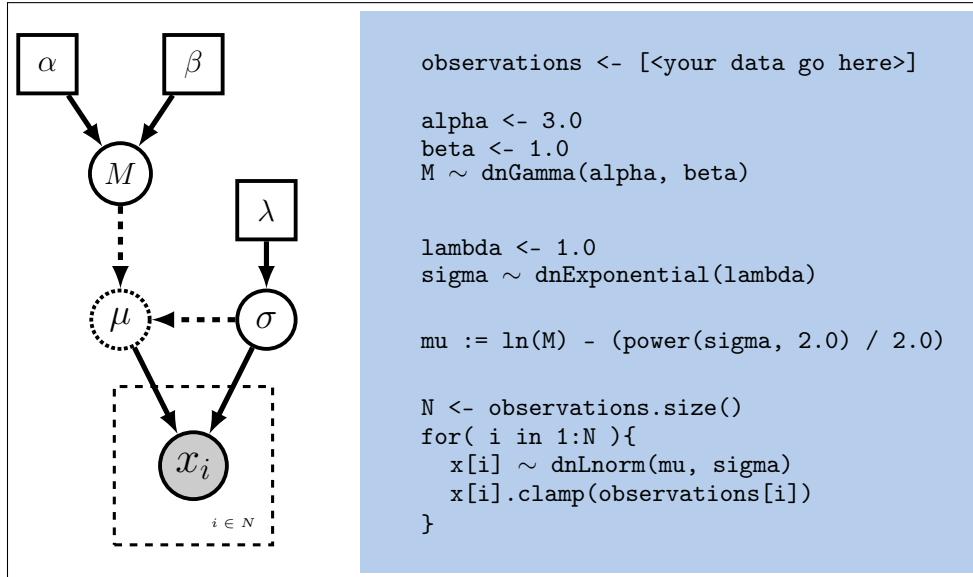


Figure 1.3: Specifying a model with **Rev**. The graphical model of the observed parameter x is shown on the left. In this example, x is log-normally distributed with a location parameter of μ and a standard deviation of σ , thus $x \sim \text{Lognormal}(\mu, \sigma)$. The expected value of x (or mean)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

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

```

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

```

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

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

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

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

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

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

Getting help in RevBayes

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

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

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

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

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

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

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

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

RevBayes Users' Forum

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

- [revbayes-users](#)

Bibliography

- Gilks, W., A. Thomas, and D. Spiegelhalter. 1994. A language and program for complex Bayesian modelling. *The Statistician* 43:169–177.
- Höhna, S., T. A. Heath, B. Boussau, M. J. Landis, F. Ronquist, and J. P. Huelsenbeck. 2014. Probabilistic graphical model representation in phylogenetics. *Systematic Biology* 63:753–771.
- Jordan, M. 2004. Graphical models. *Statistical Science* 19:140–155.
- Koller, D. and N. Friedman. 2009. *Probabilistic Graphical Models: Principles and Techniques*. The MIT Press, Cambridge.
- Lunn, D., D. Spiegelhalter, A. Thomas, and N. Best. 2009. The BUGS project: Evolution, critique and future directions. *Statistics in Medicine* 28:3049–3067.
- Lunn, D. J., A. Thomas, N. Best, and D. Spiegelhalter. 2000. WinBUGS – a Bayesian modelling framework: Concepts, structure, and extensibility. *Statistics and Computing* 10:325–337.

Chapter 2

Introduction to RevBayes and Rev

Basic Rev Commands

Introduction

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

The first section of this tutorial involves

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Moreover, we provide functions for the common statistical distributions.

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

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

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

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

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

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

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

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

```
dnorm
```

Variable Declaration

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Variables that are not part of a model are assigned with **=**, for example, **i = 0**.

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

Simple variable manipulation and other types of assignments

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

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

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

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

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

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

Vectors

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

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

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

Second, you can use the square bracket notation.

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

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

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

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

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

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

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

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

```
v.methods()
```

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

```
v.size()
```

Control Structures

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

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

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

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

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

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

Another example using a **for** loop is the computation of the [Fibonacci number](#) for a given integer.

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

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

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

User Defined Functions

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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 [Gelman et al. \(2003\)](#). A summary is given in Table 2.1.

Table 2.1: Airline fatalities from 1976 to 1985. Reproduced from ([Gelman et al. 2003](#); 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 $P(\alpha, \beta | y)$. We will construct this algorithm explicitly, without using the high-level functions existing in `RevBayes` to perform MCMC. In the next section, we will repeat the same analysis, this time using the high-level functions. (More background on MCMC is provided in the [Introduction to Markov Chain Monte Carlo Algorithms tutorial](#).)

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

A similar distribution should be used for β' .

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

After you look at the output of the MCMC (later), 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 in 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 ([Rambaut and Drummond 2011](#)).

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

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

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

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

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

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

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

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

Similarly, we compute the likelihood ratio for each observation.

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

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

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

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

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

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

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

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

In this section of the first exercise we wrote our own little Metropolis algorithm in Rev. This becomes very cumbersome, difficult and slow if we'd 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 a 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 to reset the values if the proposals are rejected. We use the same sliding window move as we implemented above by ourselves.

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

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

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

We can now create the model by pulling 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.

Plot the curve of $m(x) = \text{E}[\exp(\alpha + \beta * x)|y]$ 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 $m_sample(x)[i] = E[\exp(\alpha[i] + \beta[i] * x)|y]$ and taking the mean of those samples (`m(x) = mean(m_sample(x))`) afterwards. Since RevBayes provides you with the samples of $m(x) = E[\exp(\alpha + \beta * x)|y] = \lambda_x$ 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 approximate this distribution by recording the expected value for the number of fatalities in 2014 ($E[\exp(\alpha + \beta * x)|y]$) 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 $x = 2014$, but this time for $x = 2016$ and estimate the probability of no fatality.

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 numbers 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 $y_t \sim \text{Poisson}(\lambda_t)$ 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 λ_t is defined by three unknown parameters: β , γ and θ . A hierarchical choice of priors is given by

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

which brings an additional parameter η in the model. For θ we have used a uniform prior over the years, but excluded year 1851 in order to make sure at least one year has rate β . The hierarchical prior carries 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, 0, 0, 2, 1, 0, 0,
  0, 1, 1, 0, 2, 3, 3, 1, 1, 2, 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

Bibliography

- Gelman, A., J. Carlin, H. Stern, and D. Rubin. 2003. Bayesian Data Analysis. 2 ed. Chapman & Hall/CRC.
- Höhna, S., T. A. Heath, B. Boussau, M. J. Landis, F. Ronquist, and J. P. Huelsenbeck. 2014. Probabilistic graphical model representation in phylogenetics. *Systematic Biology* 63:753–771.
- Rambaut, A. and A. J. Drummond. 2011. Tracer v1.5. <http://tree.bio.ed.ac.uk/software/tracer/>.

Chapter 3

Reading and manipulating data

Overview

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

Requirements

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

- RB_Getting_Started

Molecular Sequence Data

Getting Started

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

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

```
#NEXUS

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

Loading Molecular Sequence Data

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

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

Querying Dataset Attributes

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

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

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

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

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

Concatenating Sequences

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

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

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

Table 3.1: Available member functions for the **data** variable.

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

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

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

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

Excluding/Including Taxa

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

(**data**) by typing:

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

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

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

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

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

Excluding/Including Sites or Genes

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

```
data.excludeCharacter([1])
```

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

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

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

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

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

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

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

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

We can easily return the excluded cytb sequences by typing:

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

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

Biogeographical Data

For concreteness, this section focuses on the Hawaiian *Psychotria* dataset used in [Ree and Smith \(2008\)](#).

Nexus file

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

```
#NEXUS

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

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

Atlas file

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

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

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

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

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

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

Bibliography

- Ree, R. H. and S. A. Smith. 2008. Maximum likelihood inference of geographic range evolution by dispersal, local extinction, and cladogenesis. *Systematic Biology* 57:4–14.

Part II

Inference

Chapter 4

Introduction to Markov chain Monte Carlo Algorithms

Overview

This very basic tutorial provides an introduction to Bayesian inference and Markov chain Monte Carlo (MCMC) algorithms. The tutorial explains the fundamental concepts of an MCMC algorithm, such as *moves* and *monitors*, which are ubiquitous in every other tutorial. After the tutorial you should be somewhat familiar with Bayesian inference (*e.g.*, what is a prior distribution, posterior distribution, and likelihood function) and MCMC simulation (*e.g.*, what are moves and monitors and why do we need them).

A Coin Flipping (Binomial) Model

We'll begin our exploration of Bayesian inference with a simple coin-flipping model. In this model, we imagine flipping a coin n times and count the number of heads, x ; each flip comes up heads with probability p . This model gives rise to the Binomial probability distribution, with parameters n and p :

$$P(x | n, p) = \binom{n}{x} p^x (1-p)^{n-x}$$

Simple intuition suggests that, given that we observe x heads in n coin tosses, the maximum-likelihood estimate (MLE) of p is simply $\frac{x}{n}$: if we flip a coin 100 times and observe 70 heads, we assume the probability the coin comes up heads is $\frac{70}{100} = 0.7$. This is indeed the maximum likelihood estimate!

From Bayes' theorem, the *posterior distribution* of p given x , $P(p | x)$, is:

$$\overbrace{P(p | x)}^{\text{posterior distribution}} = \frac{\underbrace{P(x | p) \times P(p)}_{\text{likelihood}} \times \overbrace{P(p)}^{\text{prior}}}{\underbrace{P(x)}_{\text{marginal likelihood}}}$$

The take-home message here is that, if we're interested in doing Bayesian inference for the coin flipping model, we need to specify a *likelihood function* and a *prior distribution* for p . In virtually all practical cases, we cannot compute the posterior distribution directly and instead use numerical procedures, such as a Markov chain Monte Carlo (MCMC) algorithm. Therefore, we will also have to write an MCMC algorithm that samples parameter values in the frequency of their posterior probability.

We'll use a simple beta distribution as a prior on the parameter of the model, p . The beta distribution has two parameters, α and β (Figure 4.1). Different choices for α and β represent different prior beliefs.

Figure 4.2 shows the graphical model for the binomial model. This nicely visualizes the dependency structure in the model. We see that the two parameters α and β are drawn in solid squares, representing that these variables are constant. From these two variables, we see arrows going into the variable p . That simply means that p depends on α and β . More specifically, p is a stochastic variable (shown as a solid circle) and drawn from a beta distribution with parameters α and β . Then, we have another constant variable, n . Finally, we have the observed data x which is drawn from a Binomial distribution with parameters p and n , as can be seen by the arrows going into x . Furthermore, the solid circle of x is shaded which means that the variable has data attached to it.

Writing an MCMC from Scratch

- Make yourself familiar with the example script called *Binomial_MH_algorithm.Rev* which shows the code for the following sections. Then, start a new and empty script and follow each step provided in the

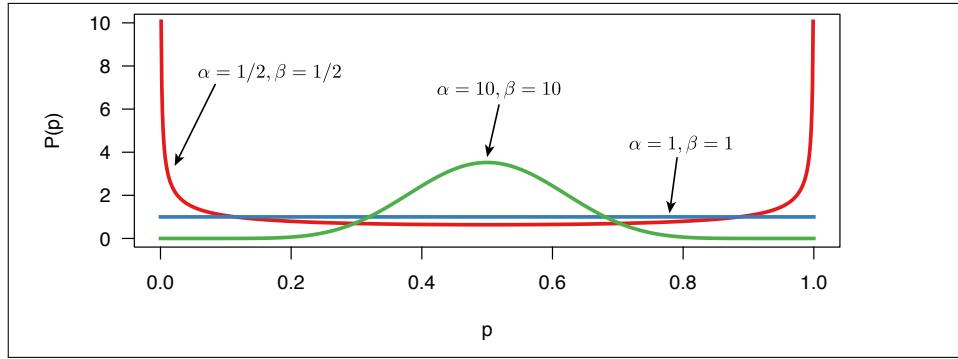


Figure 4.1: A beta distribution with two parameters, α and β . This distribution is used as a prior distribution on the probability parameter p of observing a head. Here we show different curves for the beta distribution when using different parameters.

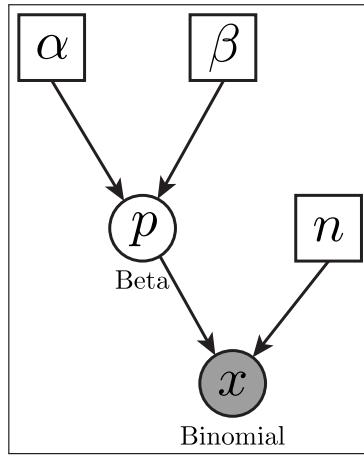


Figure 4.2: A graphical model for the binomial model.

blue boxes .

The Metropolis-Hastings Algorithm

Though RevBayes implements efficient and easy-to-use Markov chain Monte Carlo algorithms, we'll begin by writing one ourselves to gain a better understanding of the moving parts. The Metropolis-Hastings MCMC algorithm (Metropolis et al. 1953; Hastings 1970) proceeds as follows:

1. Generate initial values for the parameters of the model (in this case, p).
2. Propose a new value (which we'll call p') for some parameters of the model, (possibly) based on their current values
3. Calculate the acceptance probability, R , according to:

$$R = \min \left\{ 1, \frac{P(x | p')}{P(x | p)} \times \frac{P(p')}{P(p)} \times \frac{q(p)}{q(p')} \right\}$$

4. Generate a uniform random number between 1 and 0. If it is less than R , accept the move (set $p = p'$). Otherwise, keep the current value of p .

5. Record the values of the parameters.
6. Return to step 2 many many times, keeping track of the value of p .

Reading in the data

Actually, in this case, we're just going to make up some data on the spot. Feel free to alter these values to see how they influence the posterior distribution

```
# Make up some coin flips!
# Feel free to change these numbers
n <- 100 # the number of flips
x <- 63 # the number of heads
```

Initializing the Markov chain

We have to start the MCMC off with some initial parameter values. One way to do this is to randomly draw values of the parameters (just p , in this case) from the prior distribution. We'll assume a "flat" beta prior distribution; that is, one with parameters $\alpha = 1$ and $\beta = 1$.

```
# Initialize the chain with starting values
alpha <- 1
beta <- 1
p <- rbeta(n=1,alpha,beta)[1]
```

Likelihood function

We also need to specify the likelihood function. We use the binomial probability for the likelihood function:

```
# specify the likelihood function
function RealPos likelihood(p) {
    l = dbinomial(x,p,n,log=false)
    return l
}
```

Prior distribution

Similarly, we need to specify a function for the prior distribution. Here, we use the beta probability distribution for the prior on p :

```
# specify the prior function
function RealPos prior(p) {
```

```

    pp = dbeta(p,alpha,beta,log=false)
    return pp
}

```

Monitoring parameter values

Additionally, we are going to monitor, *i.e.*, store, parameter values into a file during the MCMC simulation. For this file we need to write the column headers:

```

# Prepare a file to log our samples
write("iteration","p","\n",file="binomial_MH.log")
write(0,p,"\n",file="binomial_MH.log",append=TRUE)

```

(You may have to change the newline characters to "`\r\n`" if you're using a Windows operating system.)

Writing the MH Algorithm

At long last, we can write our MCMC algorithm. First, let us define the frequency how often we print to file (*i.e.*, monitor), which is also often called thinning. If we set the variable `printgen` to 1, then we will store the parameter values every single iteration; if we choose `printgen=10` instead, then only every 10th iteration.

```
printgen = 10
```

We will repeat this resampling procedure many times (here, 10000), and iterate the MCMC using a `for` loop:

```

# Write the MH algorithm
reps = 10000
for(rep in 1:reps){

```

(remember to close your `for` loop at the end).

The first thing we do in the first generation is generate a new value of p' to evaluate. We'll propose a new value of p from a uniform distribution between 0 and 1. Note that in this first example we do not condition new parameter values on the current value.

```

# Propose a new value of p
p_prime <- runif(n=1,0.0,1.0)[1]

```

Next, we compute the proposed likelihood and prior probabilities, as well as the acceptance probability, R :

```
# Compute the acceptance probability
R <- ( likelihood(p_prime) / likelihood(p) ) * ( prior(p_prime) / prior(p) )
```

Then, we accept the proposal with probability R and reject otherwise:

```
# Accept or reject the proposal
u <- runif(1,0,1)[1]
if(u < R){
    # Accept the proposal
    p <- p_prime
}
```

Finally, we store the current value of p in our log file. Here, we actually check if we want to store the value during this iteration.

```
if ( (rep % printgen) == 0 ) {
    # Write the samples to a file
    write(rep,p,"\\n",file="binomial_MH.log",append=TRUE)
}
} # end MCMC
```

Exercise 1

Step 1) Write and execute this script (there is also an example file called *Binomial_MH_algorithm.Rev*).

Step 2) The `.log` file will contain samples from the posterior distribution of the model! Open the file in **Tracer** to learn about various features of the posterior distribution, for example: the posterior mean or the 95% credible interval.

Pretty awesome, right?

Below we show an example of the obtained output in **Tracer**. Specifically, Figure 4.3 shows the sample trace (left) and the estimated posterior distribution of p (right). There are other parameters, such as the posterior mean and the 95% HPD (highest posterior density) interval, that you can obtain from **Tracer**.

More on Moves: Tuning and weights

In the previous example we hard coded a single move updating the variable p by drawing a new value from a uniform(0,1) distribution. There are actually many other ways how to propose new values; some of which are more efficient than others.

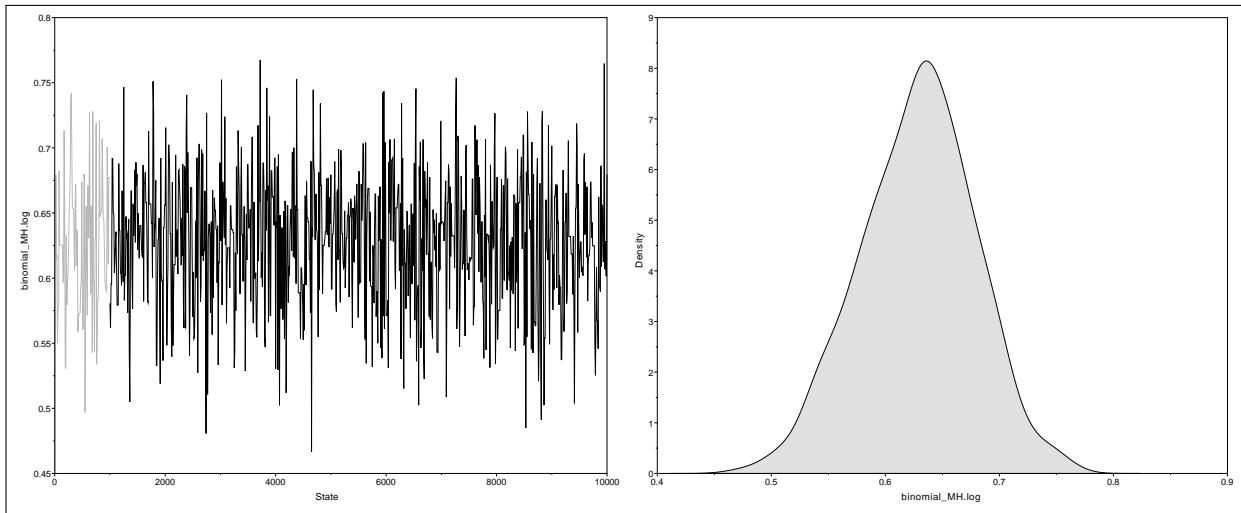


Figure 4.3: Left: The *Trace* of sample from an MCMC simulation. Right: The approximated posterior probability distribution for p .

First, let us rewrite the MCMC loop so that we use instead a function, which we call `move_uniform` for simplicity, that performs the move:

```
for (rep in 1:reps){

    # call uniform move
    move_uniform(1)

    if ( (rep % printgen) == 0 ) {
        # Write the samples to a file
        write(rep,p,"\n",file="binomial_MH.log",append=TRUE)
    }

} # end MCMC
```

This loop looks already much cleaner.

Uniform move

Now we need to actually write the `move_uniform` function. We mostly just copy the code we had before into a dedicated function

```
function move_uniform( Natural weight) {

    for (i in 1:weight) {
        # Propose a new value of p
        p_prime <- runif(n=1,0.0,1.0)[1]
```

```

# Compute the acceptance probability
R <- ( likelihood(p_prime) / likelihood(p) ) * ( prior(p_prime) / prior(p) )

# Accept or reject the proposal
u <- runif(1,0,1)[1]
if (u < R){
    # Accept the proposal
    p <- p_prime
} else {
    # Reject the proposal
    # (we don't have to do anything here)
}
}

}

```

There are a few things to consider in the function `move_uniform`. First, we do not have a return value because the move simply changes the variable p if the move is accepted. Second, we expect an argument called `weight` which will tell us how often we want to use this move. Otherwise, this function does exactly the same what was inside the for loop previously.

(Note that you need to define this function before the for loop in your script).

Sliding-window move

As a second move we will write a sliding-window move. The sliding-window moves propose an update by drawing a random number from a normal distribution and then adding this random number to the current value (*i.e.*, centered on the previous value).

```

function move_slide( RealPos delta, Natural weight) {

for (i in 1:weight) {
    # Propose a new value of p
    p_prime <- p + rnormal(n=1,0.0,delta)[1]

    # Compute the acceptance probability
    R <- ( likelihood(p_prime) / likelihood(p) ) * ( prior(p_prime) / prior(p) )

    # Accept or reject the proposal
    u <- runif(1,0,1)[1]
    if (u < R) {
        # Accept the proposal
        p <- p_prime
    } else {
        # Reject the proposal
        # (we don't have to do anything here)
    }
}

```

```

    }
}

}

```

This move has another argument, **delta**, additionally to the weight of the move. The argument **delta** defines the standard deviation of the normal distribution from which we draw new values. Thus, if **delta** is large, then the proposed values are more likely to be very different from the current value of p . Conversely, if **delta** is small, then the proposed values are more likely to be very close to the current value of p .

- Experiment with different values for **delta** and check how the effective sample size (ESS) changes.

There is, a priori, no good method for knowing what values of **delta** are most efficient. However, there are some algorithms implemented in RevBayes, called *auto-tuning*, that will estimate good values for **delta**.

Note that you could, in principle, use other distributions instead of the normal distribution too. For example, a uniform(- δ , δ) distribution would be similarly applicable.

Reflecting proposal

One thing we have to be careful about is making sure p (being a probability) stays between 0 and 1. We can do that by reflecting any proposals outside of that region back into the valid region. (One of the nice things about the real MCMC algorithms implemented in RevBayes is that it takes care of this for us).

```

if (p_prime < 0) {
    p_prime <- abs(p_prime)
} else if (p_prime > 1) {
    p_prime <- 2 - p_prime
}

```

However, this reflection procedure would have to be repeated until the values fall inside the acceptable range. Another alternative, as implicitly used here, is that the prior and/or likelihood will return *nan* if the parameter values are outside the allowed range. There are just more rejected, and thus wasted, proposals.

Scaling move

As a third and final move we will write a scaling move. The scaling move proposes an update by drawing a random number from a uniform(-0.5,0.5) distribution, exponentiating the random number, and then multiplying this scaling factor by the current value. An interesting feature of this move is that it is not symmetrical and thus needs a Hastings ratio. The Hastings ratio is rather trivial in this case, and one only needs to multiply the acceptance rate by the scaling factor.

```

function move_scale( RealPos lambda, Natural weight) {

    for (i in 1:weight) {
        # Propose a new value of p
        sf <- exp( lambda * ( runif(n=1,0,1) - 0.5 ) )
        p_prime <- p * sf

        # Compute the acceptance probability
        R <- ( likelihood(p_prime) / likelihood(p) ) * ( prior(p_prime) / prior(p) ) *
            sf

        # Accept or reject the proposal
        u <- runif(1,0,1)[1]
        if (u < R){
            # Accept the proposal
            p <- p_prime
        } else {
            # Reject the proposal
            # (we don't have to do anything here)
        }
    }

}

```

As before, this move has a tuning parameter called *lambda*.

The sliding-window and scaling moves are very common and popular moves in RevBayes. The code examples here are actually showing the exact same equation as implemented internally, except the reflection at the boundaries. It will be very useful for you to understand these moves.

Exercise 2

- Step 1) Rewrite your previous script to include these three different moves now.
- Step 2) Then, run the script to estimate the posterior distribution of *p* again.
- Step 3) Look at the output in Tracer.
- Step 4) Are the distributions, mean and credible interval the same?
- Step 5) Use only a single move and set **printgen=1**. Which move has the best ESS?
- Step 6) How does the ESS change if you use a **delta=10** for the sliding-window move?
- Step 7) Add to each move a counter variable that counts how often the move was accepted. For example:

```

if (u < R){
    # Accept the proposal
    p <- p_prime
    ++num_sliding_move_accepted
}

```

Step 8) Have a look at how the acceptance rate changes for different values of the tuning parameters.

However, this MCMC algorithm is *very* specific to our binomial model and thus hard to extend (also it's pretty inefficient!).

The Metropolis-Hastings Algorithm with the *Real RevBayes*

We'll now specify the exact same model in `Rev` using the built-in modeling functionality. It turns out that the `Rev` code to specify the above model is extremely simple and similar to the one we used before. Again, we start by “reading in” (*i.e.*, making up) our data.

```

# Make up some coin flips!
# Feel free to change these numbers
n <- 100 # the number of flips
x <- 63 # the number of heads

```

Now we specify our prior model.

```

# Specify the prior distribution
alpha <- 1
beta <- 1
p ~ dnBeta(alpha,beta)

```

One difference between `RevBayes` and the MH algorithm that we wrote above is that many MCMC proposals are already built-in, but we have to specify them *before* we run the MCMC. We usually define (at least) one move per parameter immediately after we specify the prior distribution for that parameter.

```

# Define a move for our parameter, p
moves[1] = mvSlide(p,delta=0.1,weight=1)

```

Next, our likelihood model.

```
# Specify the likelihood model
k ~ dnBinomial(p, n)
k.clamp(x)
```

We wrap our full Bayesian model into one model object (this is a convenience to keep the entire model in a single object, and is more useful when we have very large models):

```
# Construct the full model
my_model = model(p)
```

We use “monitors” to keep track of parameters throughout the MCMC. The two kinds of monitors we use here are the `mnModel`, which writes parameters to a specified file, and the `mnScreen`, which simply outputs some parts of the model to screen (as a sort of progress bar).

```
# Make the monitors to keep track of the MCMC
monitors[1] = mnModel(filename="binomial_MCMC.log", printgen=10, separator = TAB)
monitors[2] = mnScreen(printgen=100, p)
```

Finally, we assemble the analysis object (which contains the model, the monitors, and the moves) and execute the run using the `.run` command:

```
# Make the analysis object
analysis = mcmc(my_model, monitors, moves)

# Run the MCMC
analysis.run(100000)

# Show how the moves performed
analysis.operatorSummary()
```

- Open the resulting `binomial_MCMC.log` file in Tracer. Do the posterior distributions for the parameter p look the same as the ones we got from our first analysis?

Hopefully, you’ll note that this `Rev` model is substantially simpler and easier to read than the MH algorithm script we began with. Perhaps more importantly, this `Rev` analysis is *orders of magnitude* faster than our own script, because it makes use of extremely efficient probability calculations built-in to `RevBayes` (rather than the ones we hacked together in our own algorithm).

Exercise 3

- Step 1) Run the built-in MCMC and compare the results to your own MCMC. Are the posterior estimates the same? Are the acceptance rates of the moves similar?

- Step 2) Next, add a second move `moves[2] = mvScale(p,lambda=0.1,tune=true,weight=1.0)` just after the first one.
- Step 3) Run the analysis again and compare the output.
- Step 4) Finally, run a pre-burnin using `analysis.burnin(generations=10000,tuningInterval=200)` just before you call `analysis.run(100000)`. This will auto-tune the tuning parameters (*e.g.*, `delta` and `lambda`) so that the acceptance ratio is between 0.4 and 0.5.
- Step 5) What are the tuned values for `delta` and `lambda`? Did the auto-tuning increase the ESS?

Exercise 4

Play around with various parts of the model to develop an intuition for both the Bayesian model and the MCMC algorithm. For example, how does the posterior distribution change as you increase the number of coin flips (say, increase both the number of flips and the number of heads by an order of magnitude)? How does the estimated posterior distribution change if you change the prior model parameters, α and β (*i.e.*, is the model prior sensitive)? Does the prior sensitivity depend on the sample size? Are the posterior estimates sensitive to the length of the MCMC? Do you think this MCMC has been run sufficiently long, or should you run it longer?

Bibliography

- Hastings, W. K. 1970. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika* 57:97–109.
- Metropolis, N., A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. 1953. Equation of state calculations by fast computing machines. *Journal of Chemical Physics* 21:1087–1092.

Part III

Basic Phylogeny Estimation

Chapter 5

Continuous Time Markov Model for Discrete Character Evolution

Overview

This tutorial provides the first protocol from our recent publication ([Höhna et al. 2017](#)). The second protocol is described in the [Partitioned data analysis tutorial](#) and the third protocol is described in the [Bayes factor tutorial](#).

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

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

where μ_{ij} represents the instantaneous rate of substitution from state i to state j . The diagonal elements μ_i are the rates of *not* changing out of state i , equal to the sum of the elements in the corresponding row. Given the instantaneous-rate matrix, Q , we can compute the corresponding transition probabilities for a branch of length t , $P(t)$, by exponentiating the rate matrix:

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

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

In this tutorial you will perform phylogeny inference under common models of DNA sequence evolution: JC, F81, HKY85, GTR, GTR+Gamma and GTR+Gamma+I. For all of these substitution models, you will perform an MCMC analysis to estimate phylogeny and other model parameters. The estimated trees will be unrooted trees with independent branch-length parameters. We will provide comments on how to modify the tutorial if you wish to estimate rooted, clock-like trees. All the assumptions will be covered more in detail later in this tutorial.

Requirements

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

- [Getting started](#)
- [Rev basics](#)

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

Data and files

We provide the data file(s) which we will use in this tutorial. You may want to use your own data instead. In the **data** folder, you will find the following files

- **primates_and_galeopterus_cytb.nex**: Alignment of the *cytochrome b* subunit from 23 primates representing 14 of the 16 families (*Indriidae* and *Callitrichidae* are missing). Note that there is one outgroup species included: *Galeopterus variegatus*.

Example: Character Evolution under the Jukes-Cantor Substitution Model

Getting Started

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

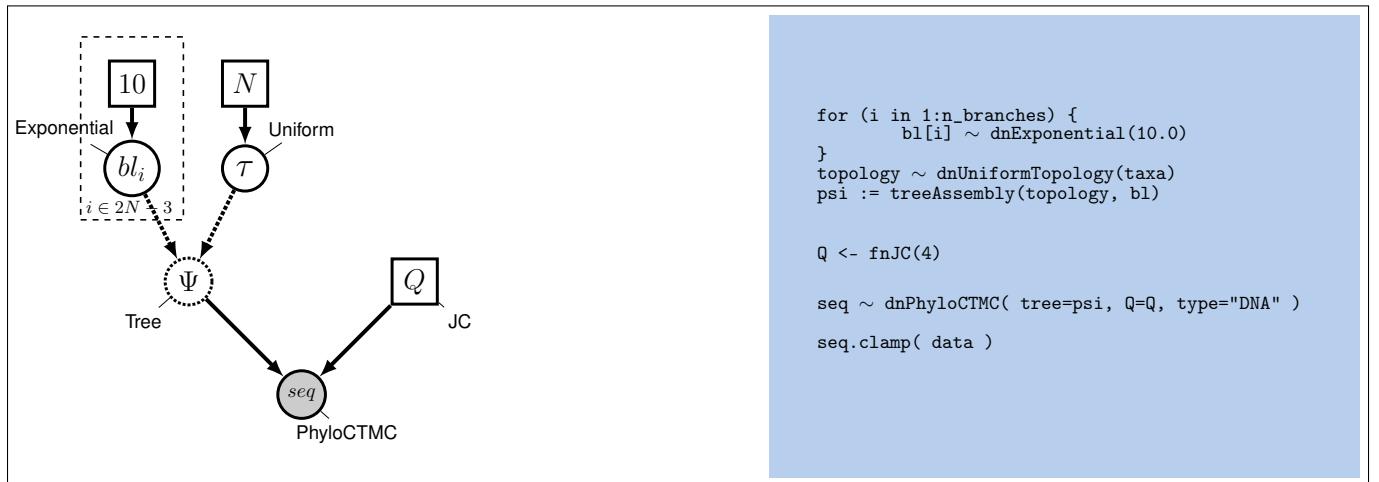


Figure 5.1: Graphical model representation of a simple phylogenetic model. The graphical model shows the dependencies between the parameters. Here, the rate matrix Q is a constant variable because it is fixed and does not depend on any parameters. The only free parameters of this model, the Jukes-Cantor model, are the tree Ψ including the node ages.

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

$$Q_{JC69} = \begin{pmatrix} * & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & * & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{3} & * & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & * \end{pmatrix},$$

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

$$P_{JC69} = \begin{pmatrix} \frac{1}{4} + \frac{3}{4}e^{-rt} & \frac{1}{4} - \frac{1}{4}e^{-rt} & \frac{1}{4} - \frac{1}{4}e^{-rt} & \frac{1}{4} - \frac{1}{4}e^{-rt} \\ \frac{1}{4} - \frac{1}{4}e^{-rt} & \frac{1}{4} + \frac{3}{4}e^{-rt} & \frac{1}{4} - \frac{1}{4}e^{-rt} & \frac{1}{4} - \frac{1}{4}e^{-rt} \\ \frac{1}{4} - \frac{1}{4}e^{-rt} & \frac{1}{4} - \frac{1}{4}e^{-rt} & \frac{1}{4} + \frac{3}{4}e^{-rt} & \frac{1}{4} - \frac{1}{4}e^{-rt} \\ \frac{1}{4} - \frac{1}{4}e^{-rt} & \frac{1}{4} - \frac{1}{4}e^{-rt} & \frac{1}{4} - \frac{1}{4}e^{-rt} & \frac{1}{4} + \frac{3}{4}e^{-rt} \end{pmatrix},$$

where t is the branch length in units of time, and r is the rate (clock) for the process. In the later exercises you will be asked to specify more complex substitution models. **Don't be scared by the math!** RevBayes will take care of all the computations for you. Here we only provide some of the equations for the models in case you might be interested in the details. You will be able to complete the exercises without understanding the underlying math.

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

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

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

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

Loading the Data

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

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

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

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

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

```
Number of characters:          1141
Number of included characters: 1141
Datatype:                      DNA
```

Next we will specify some useful variables based on our dataset. The variable `data` has *member functions* that we can use to retrieve information about the dataset. These include, for example, the number of species and the taxa. We will need that taxon information for setting up different parts of our model.

```
n_species <- data.ntaxa()
n_branches <- 2 * n_species - 3
taxa <- data.taxon()
```

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

```
mvi = 0
mni = 0
```

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

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

Jukes-Cantor Substitution Model

A given substitution model is defined by its corresponding instantaneous-rate matrix, Q . The Jukes-Cantor substitution model does not have any free parameters (as the substitution rates are all assumed to be equal, and there is a separate parameter that scales their overall magnitude), so we can define it as a constant variable. The function `fnJC(n)` will create an instantaneous-rate matrix for a character with n states. Since we use DNA data here, we create a 4x4 instantaneous-rate matrix:

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

You can see the rates of the Q matrix by typing

```

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

```

As you can see, all substitution rates are equal.

Tree Topology and Branch Lengths

The tree topology and branch lengths are stochastic nodes in our phylogenetic model. In Figure 5.1, the tree topology is denoted Ψ and the length of the branch leading to node i is bl_i .

We will assume that all possible labeled, unrooted tree topologies have equal probability. This is the `dnUniformTopology()` distribution in RevBayes. Note that in RevBayes it is advisable to specify the outgroup for your study system if you use an unrooted tree prior, whereas other software, *e.g.*, MrBayes uses the first taxon in the data matrix file as the outgroup. Specify the `topology` stochastic node by passing in the tip labels `names` to the `dnUniformTopology()` distribution:

```

out_group = clade("Galeopterus_variegatus")
topology ~ dnUniformTopology(taxa, outgroup=out_group)

```

Some types of stochastic nodes can be updated by a number of alternative moves. Different moves may explore parameter space in different ways, and it is possible to use multiple different moves for a given parameter to improve mixing (the efficiency of the MCMC simulation). In the case of our unrooted tree topology, for example, 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 proposal `weight`.

```

moves[++mvi] = mvNNI(topology, weight=1.0)
moves[++mvi] = mvSPR(topology, weight=1.0)

```

The weight specifies how often the move will be applied either on average per iteration or relative to all other moves. Have a look at the [MCMC Diagnosis tutorial](#) for more details about moves and MCMC strategies (found on the [RevBayes Tutorials Website](#)).

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

```

for (i in 1:n_branches) {
  br_lens[i] ~ dnExponential(10.0)
}

```

```

    moves[++mvi] = mvScale(br_lens[i])
}

```

It is convenient for monitoring purposes to add the tree length as deterministic variable. The tree length is simply the sum of all branch lengths. Accordingly, the tree length can be computed using the `sum()` function, which calculates the sum of any vector of values.

```
TL := sum(br_lens)
```

Alternative branch-length priors

Some studies, *e.g.*, [Brown et al. \(2010\)](#); [Rannala et al. \(2012\)](#), have criticized the exponential prior distribution for branch lengths because it induces a gamma-dsitributed tree-length and the mean of this gamma distribution grows with the number of taxa. For example, we can use instead a specific gamma prior distribution (or any other distribution defined on a positive real variable) for the tree length, and then use a Dirichlet prior distribution to break the tree length into the corresponding branch lengths ([Zhang et al. 2012](#)).

```

# specify a prior distribution on the tree length with your desired mean
TL ~ dnGamma(2,4)
moves[++mvi] = mvScale(TL)

# now create a random variable for the relative branch lengths
rel_branch_lengths ~ dnDirichlet( rep(1.0,n_branches) )
moves[++mvi] = mvBetaSimplex(rel_branch_lengths, weight=n_branches)
moves[++mvi] = mvDirichletSimplex(rel_branch_lengths, weight=n_branches/10.0)

# finally, transform the relative branch lengths into actual branch lengths
br_lens := rel_branch_lengths * TL

```

Finally, we can create a *phylogram* (a phylogeny in which the branch lengths are proportional to the expected number of substitutions/site) by combining the tree topology and branch lengths. We do this 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 `psi` variable is a deterministic node:

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

Alternative Analysis

Prior on Time-Trees: Tree Topology and Node Ages

Alternatively, you may want to specify a prior on time-trees. Here we will briefly indicate how to specify such an prior which will lead to inference of time trees.

The tree (the topology and node ages) is a stochastic node in our phylogenetic model. For simplicity, we will assume a uniform prior on both topologies and node ages. The distribution in `RevBayes` is `dnUniformTimeTree()`.

- For more information on tree priors, such as birth-death processes, please read the [RB_DiversificationRate_Tutorial](#).

First, we need to specify the age of the tree:

```
root_age <- 10.0
```

Here we simply assumed that the tree is 10.0 time units old. We could also specify a prior on the root age if we have fossil calibrations (see [Divergence Time and Calibration Tutorial](#)). Next, we specify the `tree` stochastic variable by passing in the taxon information `taxa` to the `dnUniformTimeTree()` distribution:

```
psi ~ dnUniformTimeTree(rootAge=root_age, taxa=taxa)
```

Some types of stochastic nodes can be updated by a number of alternative moves. Different moves may explore parameter space in different ways, and it is possible to use multiple different moves for a given parameter to improve mixing (the efficiency of the MCMC simulation). In the case of our rooted tree, for example, we can use both a nearest-neighbor interchange move without and with changing the node ages (`mvNarrow` and `mvNNI`) and a fixed-nodeheight subtree-prune and regrafting move (`mvFNPR`) and its Metropolized-Gibbs variant (`mvGPR`) ([Höhna et al. 2008](#); [Höhna and Drummond 2012](#)). We also need moves that change the ages of the internal nodes, for example, `mvSubtreeScale` and `mvNodeTimeSlideUniform`. These moves do not have tuning parameters associated with them, thus you only need to pass in the `psi` node and proposal `weight`.

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

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

Molecular clock

Additionally, in the case of time-calibrated trees, we need to add a molecular clock rate parameter. For example, we know from empirical estimates that the molecular clock rate is about 0.01 (=1%) per million years per site. Nevertheless, we can estimate it here because we fixed the root age. We use a uniform prior on the log-transform clock rate. This specifies our lack of prior knowledge on the magnitude of the clock rate.

```
log_clock_rate ~ dnUniform(-6,1)
moves[++mvi] = mvSlide(log_clock_rate, weight=2.0)
clock_rate := 10^log_clock_rate
```

- Instead, you could also fix the clock rate and estimate the root age. For more information on molecular clocks please read the [RB_DivergenceTime_Tutorial](#).

Putting it All Together

We have fully specified all of the parameters of our phylogenetic model—the tree topology with branch lengths, and the substitution model that describes how the sequence data evolved over the tree with branch lengths. Collectively, these parameters comprise a distribution called the *phylogenetic continuous-time Markov chain*, and we use the `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`; (3) the `type` of character data.

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

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

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

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

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

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

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

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

```
mymodel
```

Performing an MCMC Analysis Under the Jukes-Cantor Model

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

Specifying Monitors

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

```
monitors[++mni] = mnModel(filename="output/primates_cytb_JC_posterior.log",printgen  
=10, separator = TAB)
```

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

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

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

```
monitors[++mni] = mnScreen(printgen=1000, TL)
```

This monitor mostly helps us to see the progress of the MCMC run.

Initializing and Running the MCMC Simulation

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

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

Notice that we also specified `nruns=2` which means that `RevBayes` will automatically run 2 independent MCMC runs. You will find that the output is created in two files with extension `_run_1` and `_run_2` for each replicate and additionally the samples from both runs are combined into one file for more convenient post-processing.

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

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

Now, run the MCMC:

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

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

Methods for visualizing the marginal densities of parameter values are not currently available in `RevBayes` itself. Thus, it is important to use programs like `Tracer` ([Rambaut and Drummond 2011](#)) to evaluate mixing and non-convergence.

- Look at the files called `output/primates_cytb_JC_posterior_run_1.log` and `output/primates_cytb_JC_posterior_run_2.log` in `Tracer`. There you see the posterior distributions of the continuous parameters, *e.g.*, the tree length variable `TL`.

Exercise 1

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

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

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

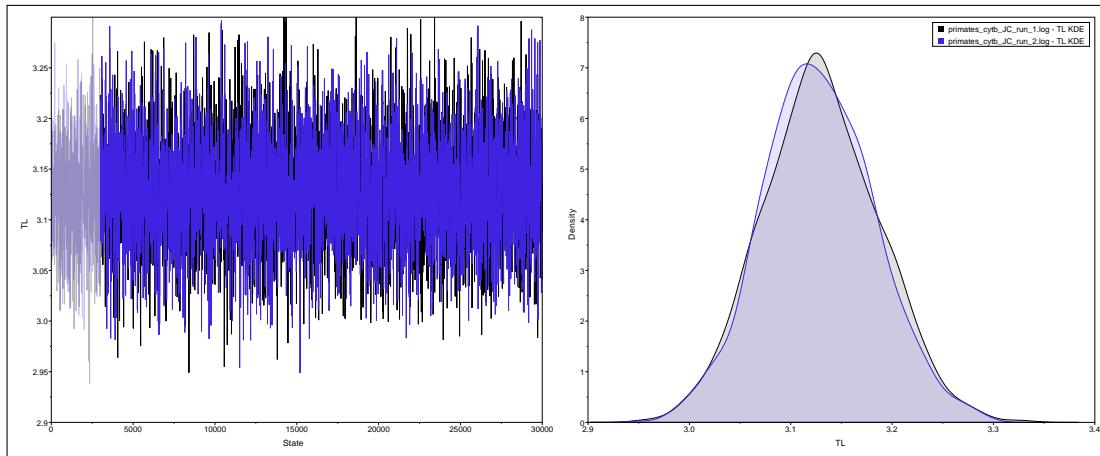


Figure 5.2: Left: Trace of tree-length samples for the two replicated MCMC runs. The caterpillar-like look is a good sign as well as that the two runs look virtually identical. You will also see that the effective sample size is comparably large, *i.e.*, much larger than 200. Right: Posterior distribution of the tree length of the primate phylogeny under a Jukes-Cantor substitution model. Here we show the two independent replicated MCMC runs which yield the same posterior estimate. This is a superficial test that the MCMC has converged to the same estimate.

```
map_tree = mapTree(treetrace,"output/primates_cytb_JC_MAP.tree")
```

→ Look at the file called `output/primates_cytb_JC_MAP.tree` in FigTree. We show it in Figure 5.3.

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

Note, you can query the posterior probability of a clade being monophyletic using the following command:

```
Lemuroidea <- clade("Cheirogaleus_major",
                      "Daubentonia_madagascariensis",
                      "Lemur_catta",
                      "Lepilemur_hubbardorum",
                      "Microcebus_murinus",
                      "Propithecus_coquereli",
                      "Varecia_variegata_variegata")

treetrace.cladeProbability( Lemuroidea )
```

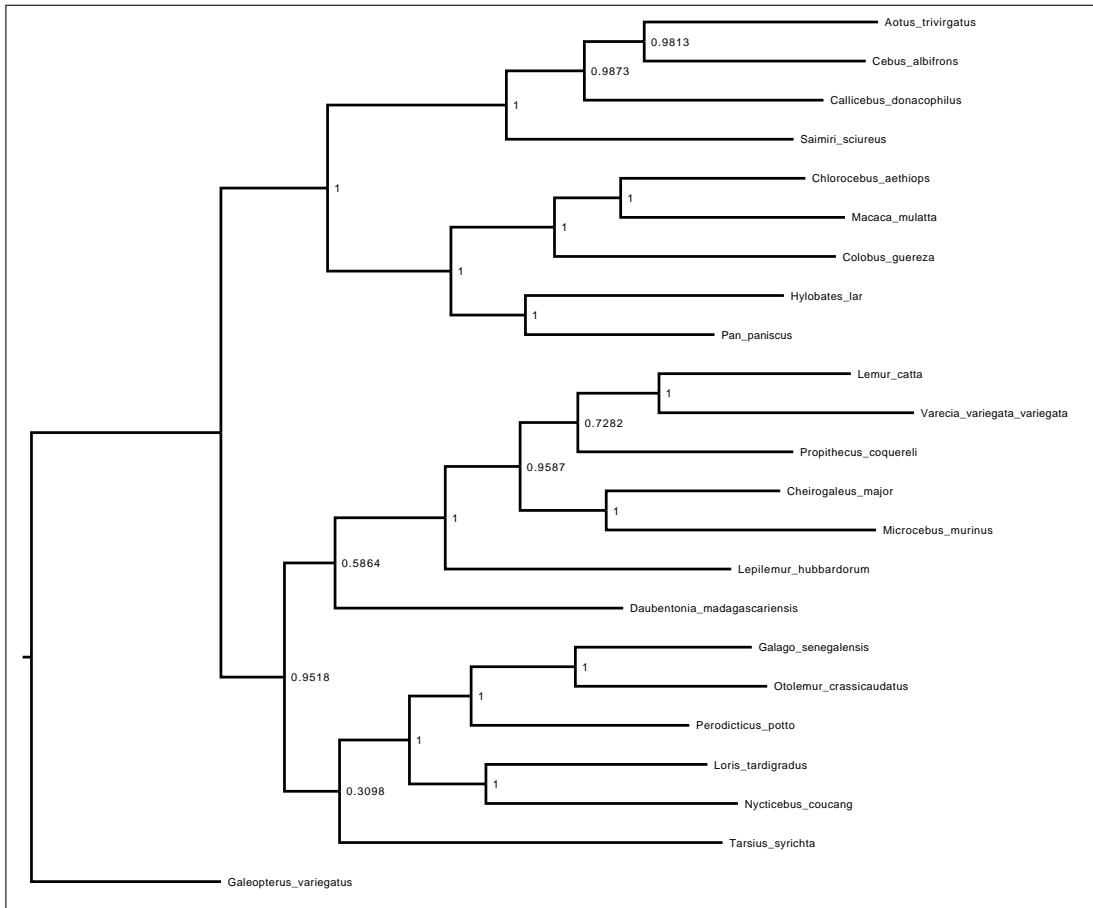


Figure 5.3: Maximum a posteriori estimate of the primate phylogeny under a Jukes-Cantor substitution model. The numbers at the nodes show the posterior probabilities for the clades. We have rooted the tree at the outgroup *Galeopterus variegatus*

Table 5.1: Posterior probabilities of primate phylogenetic relationships*.

Model	Lemuroidea	Lorisoidae	Platyrrhini	Catarrhini
Jukes-Cantor				
HKY85				
F81				
GTR				
GTR+Γ				
GTR+Γ+I				

*you can edit this table

Table 5.2: Primate species and famaly relationships.

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

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

The Jukes-Cantor model assumes that all substitution rates are equal, which also implies that the stationary frequencies of the four nucleotide bases are equal. These assumptions are not very biologically reasonable, so we might wish to consider a more realistic substitution model that relaxes some of these assumptions.

For example, we might allow stationary frequencies, π , to be unequal, and allow rates of transition and transversion substitutions to differ, κ . This corresponds to the substitution model proposed by Hasegawa et al. (1985; HKY), which is specified with the following instantaneous-rate matrix:

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

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

- Use the file `mcmc_JC.Rev` as a starting point for the HKY analysis.

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

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

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

```
moves[++mvi] = mvBetaSimplex(pi, weight=2)
moves[++mvi] = mvDirichletSimplex(pi, weight=1)
```

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

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

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

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

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

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

This should be all for the HKY model.

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

Exercise 2

- With figure 5.1 as your guide, draw the probabilistic graphical model of the HKY model.
- Copy the file called `mcmc_JC.Rev` and modify it by including the necessary parameters to specify the HKY substitution model.
- Run an MCMC analysis to estimate the posterior distribution under the HKY substitution model.
- Are the resulting estimates of the base frequencies equal? If not, how much do they differ? Are the estimated base frequencies similar to the empirical base frequencies? The empirical base frequencies are the frequencies of the characters in the alignment, which can be computed with RevBayes by `data.getEmpiricalBaseFrequencies()`.
- Is the inferred rate of transition substitutions higher than the rate of transversion substitutions? If so, by how much?
- Like the HKY model, the Felsenstein 1981 (F81) substitution model has unequal stationary frequencies, but it assumes equal transition-transversion rates ([Felsenstein 1981](#)). Can you set up the F81 model and run an analysis?
- Complete the Table 5.1 by reporting the posterior probabilities of phylogenetic relationships.

The General Time-Reversible (GTR) Substitution Model

The HKY substitution model can accommodate unequal base frequencies and different rates of transition and transversion substitutions. Despite these extensions, the HKY model may still be too simplistic for many real datasets. Here, we extend the HKY model to specify the General Time Reversible (GTR) substitution model (Tavaré 1986), which allows all six exchangeability rates to differ (Figure 5.4).

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

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

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

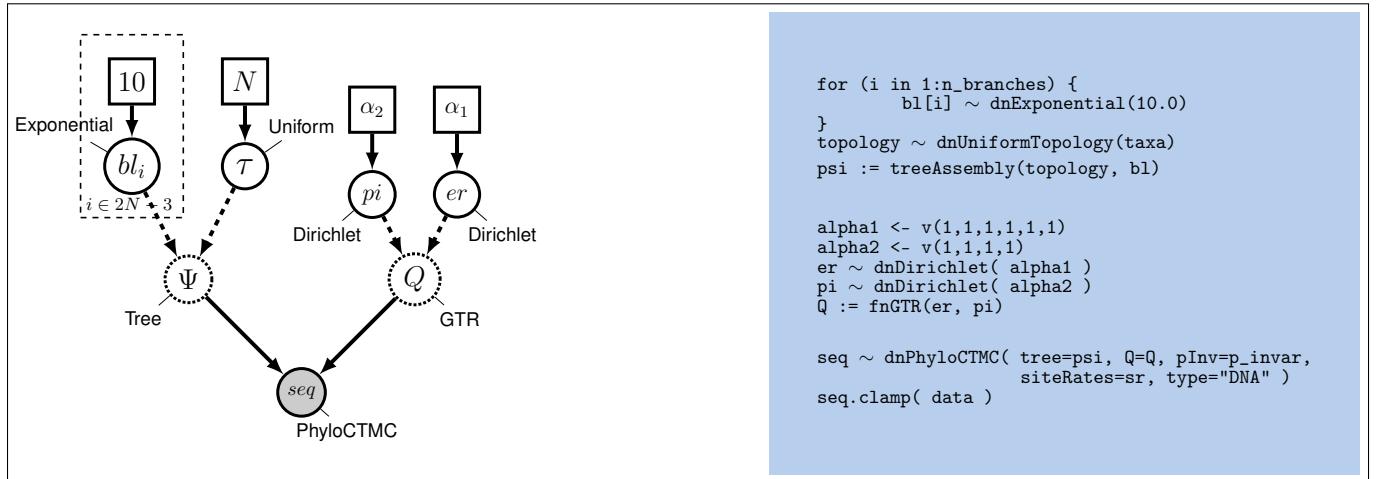


Figure 5.4: Graphical model representation of the General Time Reversible (GTR) phylogenetic model.

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

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

This node defines the concentration-parameter values of the Dirichlet prior distribution on the exchangeability rates. Now, we can create a stochastic node for the exchangeability rates using the `dnDirichlet()` function, which takes the vector of concentration-parameter values as an argument and the `~` operator. Together, these create a stochastic node named `er` (θ in Figure 5.4):

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

The Dirichlet distribution assigns probability densities to a group of parameters: e.g., those that measure proportions and must sum to 1. Here, we have specified a six-parameter Dirichlet prior, where each value describes one of the six relative rates of the GTR model: (1) $A \leq C$; (2) $A \leq G$; (3) $A \leq T$; (4) $C \leq G$; (5) $C \leq T$; (6) $G \leq T$. The input parameters of a Dirichlet distribution are called shape (or concentration) parameters. The expectation and variance for each variable are related to the sum of the shape parameters. The prior we specified above is a ‘flat’ or symmetric Dirichlet distribution; all of the shape parameters are equal (1,1,1,1,1,1). This describes a model that allows for equal rates of change between nucleotides, such that the expected rate for each is equal to $\frac{1}{6}$ (Figure 5.5a). We might also parameterize the Dirichlet distribution such that all of the shape parameters were equal to 100, which would also specify a prior with an expectation of equal exchangeability rates (Figure 5.5b). However, by increasing the values of the shape parameters, `er_prior <- v(100,100,100,100,100,100)`, the Dirichlet distribution will more strongly favor equal exchangeability rates; (*i.e.*, a relatively *informative* prior). Alternatively, we might consider an asymmetric Dirichlet parameterization that could reflect a strong prior belief that transition and transversion substitutions occur at different rates. For example, we might specify the prior density `er_prior <- v(4,8,4,4,8,4)`. Under this model, the expected rate for transversions would be $\frac{4}{32}$ and that for transitions would be $\frac{8}{32}$, and there would be greater prior probability on sets of GTR rates that matched this configuration (Figure 5.5c). Yet another asymmetric prior could specify that each of the six GTR rates had a different value conforming to a Dirichlet(2,4,6,8,10,12). This would lead to a different prior probability density for each rate parameter (Figure 5.5d). Without strong prior knowledge about the pattern of relative rates, however, we can better reflect our uncertainty by using a vague prior on the GTR rates. Notably, all patterns of relative rates have the same probability density under `er_prior <- v(1,1,1,1,1,1)`.

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

```
moves[++mvi] = mvBetaSimplex(er, weight=3)
moves[++mvi] = mvDirichletSimplex(er, weight=1)
```

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

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

The node `pi` represents the π node in Figure 5.4. Now add the simplex scale move on the stationary frequencies to the moves vector:

```
moves[++mvi] = mvBetaSimplex(pi, weight=2)
moves[++mvi] = mvDirichletSimplex(pi, weight=1)
```

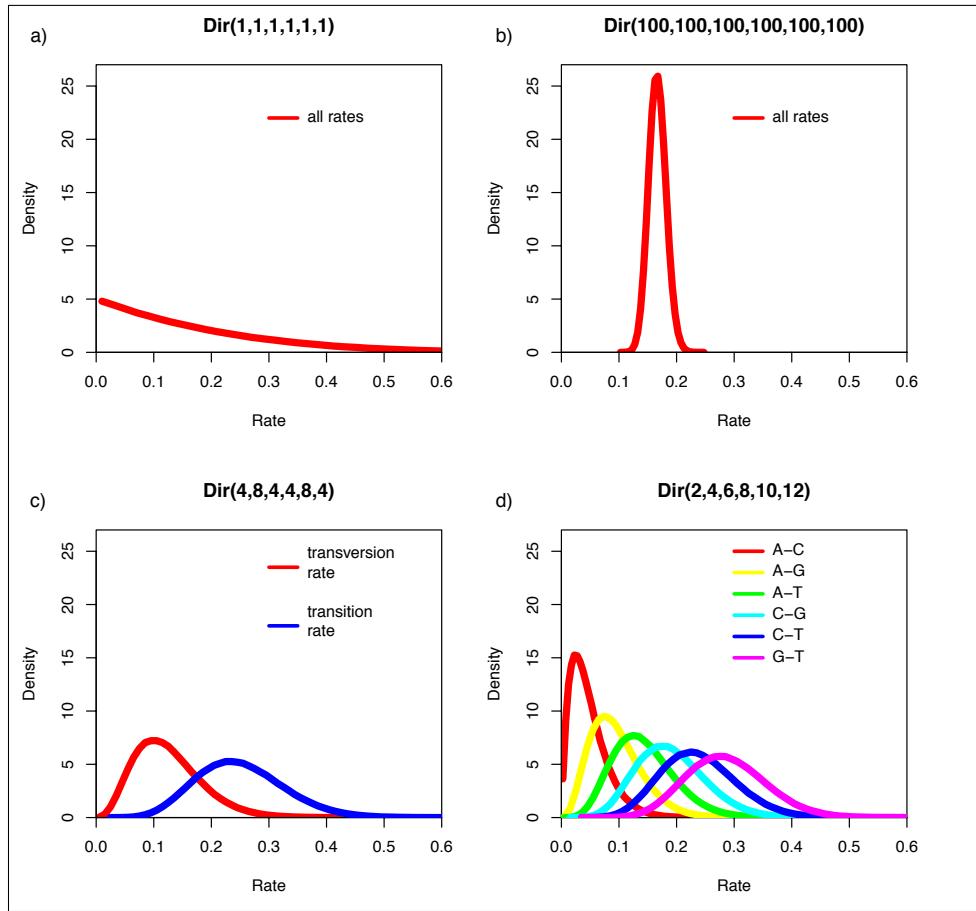


Figure 5.5: Four different examples of Dirichlet priors on exchangeability rates.

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

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

Exercise 3

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

The Discrete Gamma Model of Among Site Rate Variation

Members of the GTR family of substitution models assume that rates are homogeneous across sites, an assumption that is often violated by real data. We can accommodate variation in substitution rate among sites (ASRV) by adopting the discrete-gamma model (Yang 1994). This model assumes that the substitution rate at each site is a random variable that is described by a discretized gamma distribution, which has two parameters: the shape parameter, α , and the rate parameter, β . In order that we can interpret the branch lengths as the expected number of substitutions per site, this model assumes that the mean site rate is equal to 1. The mean of the gamma is equal to α/β , so a mean-one gamma is specified by setting the two parameters to be equal, $\alpha = \beta$. This means that we can fully describe the gamma distribution with the single shape parameter, α . The degree of among-site substitution rate variation is inversely proportional to the value of the α -shape parameter. As the value of the α -shape increases, the gamma distribution increasingly resembles a normal distribution with decreasing variance, which therefore corresponds to decreasing levels of ASRV (Figure 5.6). By contrast, when the value of the α -shape parameter is < 1 , the gamma distribution assumes a concave distribution that concentrates most of the prior density on low rates, but retains some prior mass on sites with very high rates, which therefore corresponds to high levels of ASRV (Figure 5.6). Note that, when $\alpha = 1$, the gamma distribution collapses to an exponential distribution with a rate parameter equal to β .

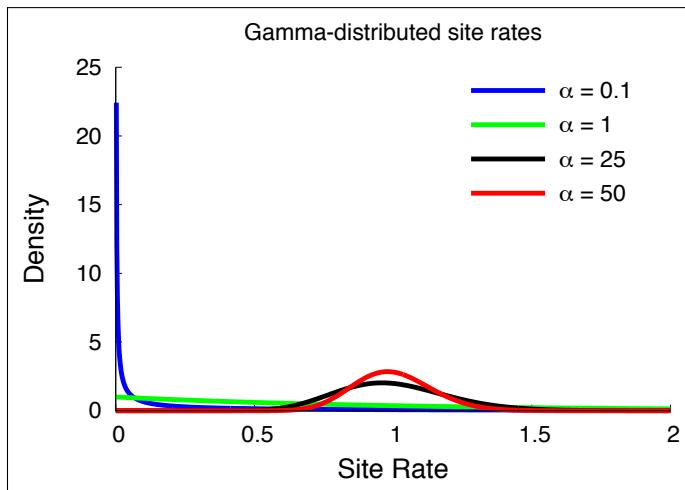


Figure 5.6: The probability density of mean-one gamma-distributed rates for different values of the α -shape parameter.

We typically lack prior knowledge regarding the degree of ASRV for a given alignment. Accordingly, rather than specifying a precise value of α , we can instead estimate the value of the α -shape parameter from the data. This requires that we specify a diffuse (relatively ‘uninformative’) prior on the α -shape parameter. For this analysis, we will use a lognormal distribution with a mean parameter, `alpha_prior_mean`, equal to 5.0, and standard deviation, `alpha_prior_sd`, equal to 0.587405 (thus, 95% of the prior density spans exactly one order of magnitude).

This approach for accommodating ASRV is another example of a hierarchical model (Figure 5.7). That is, variation in substitution rates across sites is addressed by applying a site-specific rate multiplier to each of the j sites, r_j . These rate-multipliers are drawn from a discrete, mean-one gamma distribution; the shape of this prior distribution (and the corresponding degree of ASRV) is governed by the α -shape parameter. The α -shape parameter, in turn, is treated as a lognormal distributed random variable. Finally, the shape of the lognormal prior is governed by the mean and standard deviation parameters, which are set to fixed

values.

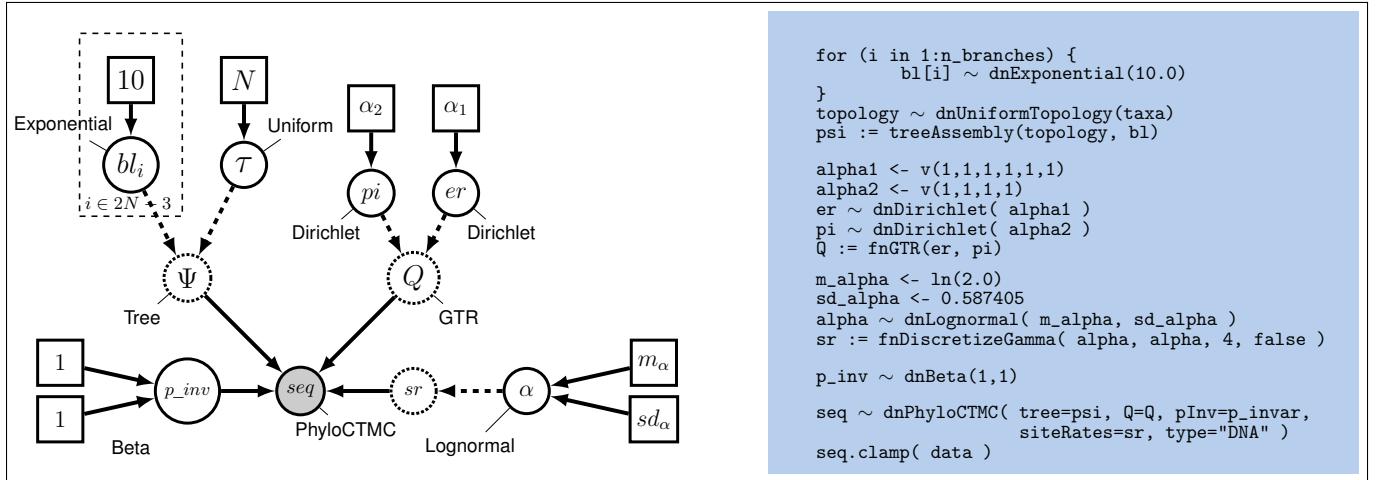


Figure 5.7: Graphical model representation of the General Time Reversible (GTR) + Gamma phylogenetic model with invariable sites.

Setting up the Gamma Model in RevBayes

Create a constant node called **alpha_prior_mean** for the mean parameter and a constant node called **alpha_prior_sd** for the standard deviation of the lognormal prior on the gamma-shape parameter (this is represented as the constant m_α and sd_α parameters in Figure 5.7):

```

alpha_prior_mean <- ln(5.0)
alpha_prior_sd <- 0.587405

```

Then create a stochastic node called **alpha** with a lognormal prior (this represents the stochastic node for the α -shape parameter in Figure 5.7):

```

alpha ~ dnLognormal(alpha_prior_mean, alpha_prior_sd)

```

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

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

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

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

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

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

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

Exercise 4

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

- Is there an impact on the estimated phylogeny compared with the previous analyses? Look at the MAP tree and the posterior probabilities of the clades.
- Complete the table of the phylogenetic relationship of primates.

Modeling Invariable Sites

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

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

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

```
data.getNumInvariantSites()
```

There seem to be a substantial number of invariant sites.

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

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

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

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

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

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

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

Exercise 5

- Extend the GTR model to account for invariable sites and run an analysis.
- What is the estimated probability of invariable sites and how does it relate to the ratio of invariant sites to the total number of sites?

- Extend the GTR+ Γ model to account for invariable sites and run an analysis.
- What is the estimated probability of invariable sites now?
- Complete the table of the phylogenetic relationship of primates.

Bibliography

- Brown, J. M., S. M. Hедтke, A. R. Lemmon, and E. M. Lemmon. 2010. When trees grow too long: Investigating the causes of highly inaccurate Bayesian branch-length estimates. *Systematic Biology* 59:145–161.
- Felsenstein, J. 1981. Evolutionary trees from DNA sequences: a maximum likelihood approach. *Journal of Molecular Evolution* 17:368–376.
- Hasegawa, M., H. Kishino, and T. Yano. 1985. Dating of the human-ape splitting by a molecular clock of mitochondrial DNA. *Journal of Molecular Evolution* 22:160–174.
- Höhna, S., M. Defoin-Platel, and A. Drummond. 2008. Clock-constrained tree proposal operators in Bayesian phylogenetic inference. Pages 1–7 in 8th IEEE International Conference on BioInformatics and BioEngineering, 2008. BIBE 2008.
- Höhna, S. and A. J. Drummond. 2012. Guided tree topology proposals for Bayesian phylogenetic inference. *Systematic Biology* 61:1–11.
- Höhna, S., M. J. Landis, and T. A. Heath. 2017. Phylogenetic inference using RevBayes. *Current Protocols in Bioinformatics*.
- Jukes, T. and C. Cantor. 1969. Evolution of protein molecules. *Mammalian Protein Metabolism* 3:21–132.
- Rambaut, A. and A. J. Drummond. 2011. Tracer v1.5. <http://tree.bio.ed.ac.uk/software/tracer/>.
- Rannala, B., T. Zhu, and Z. Yang. 2012. Tail paradox, partial identifiability, and influential priors in Bayesian branch length inference. *Molecular Biology and Evolution* 29:325–335.
- Tavaré, S. 1986. Some probabilistic and statistical problems in the analysis of DNA sequences. *Some Mathematical Questions in Biology: DNA Sequence Analysis* 17:57–86.
- Yang, Z. 1994. Maximum likelihood phylogenetic estimation from DNA sequences with variable rates over sites: Approximate methods. *Journal of Molecular Evolution* 39:306–314.
- Zhang, C., B. Rannala, and Z. Yang. 2012. Robustness of compound Dirichlet priors for Bayesian inference of branch lengths. *Systematic Biology* 61:779–784.

Part IV

Model Selection, Model Averaging, and Model Testing

Chapter 6

Model Selection and Bayes Factors

Overview

This tutorial provides the third protocol from our recent publication ([Höhna et al. 2017](#)). The first protocol is described in the [Substitution model tutorial](#) and the second protocol is described in the [Partitioned data analysis tutorial](#).

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

Requirements

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

- [Getting started](#)
- [Rev basics](#)
- [Substitution model](#)

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

Data and files

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

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

Introduction

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

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

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

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

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

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](#)):

$$BF(M_0, M_1) = \frac{\mathbb{P}(M_0 | \mathbf{X}, \theta_0)}{\mathbb{P}(M_1 | \mathbf{X}, \theta_1)} \div \frac{\mathbb{P}(M_0)}{\mathbb{P}(M_1)}. \quad (6.1)$$

Note that interpreting Bayes factors involves some subjectivity. That is, it is up to you to decide the degree of your belief in M_0 relative to M_1 . Despite the absence of an absolutely objective model-selection threshold, we can refer to the scale (outlined by [Jeffreys 1961](#)) that provides a “rule-of-thumb” for interpreting these measures (Table 6.1).

Table 6.1: The scale for interpreting Bayes factors by Harold [Jeffreys \(1961\)](#).

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

For a detailed description of Bayes factors see [Kass and Raftery \(1995\)](#)

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

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

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

$$\mathbb{P}(\mathbf{X} | M_i) = \int \mathbb{P}(\mathbf{X} | \theta_i) \mathbb{P}(\theta_i) dt. \quad (6.2)$$

This makes it clear that more complex (parameter-rich) models are penalized by virtue of the associated prior: each additional parameter entails integration of the likelihood over the corresponding prior density. If you refer back to equation 6.1, you can see that, with very little algebra, the ratio of marginal likelihoods is equal to the Bayes factor:

$$BF(M_0, M_1) = \frac{\mathbb{P}(\mathbf{X} | M_0)}{\mathbb{P}(\mathbf{X} | M_1)} = \frac{\mathbb{P}(M_0 | \mathbf{X}, \theta_0)}{\mathbb{P}(M_1 | \mathbf{X}, \theta_1)} \div \frac{\mathbb{P}(M_0)}{\mathbb{P}(M_1)}. \quad (6.3)$$

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 6.2), thus we must resort to numerical integration methods to estimate or approximate these values. In this exercise, we will estimate the marginal likelihood for each partition scheme using both the stepping-stone (Xie et al. 2011; Fan et al. 2011) and path sampling estimators (Lartillot 2006; Baele et al. 2012).

Substitution Models

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

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

Estimating the Marginal Likelihood

We will estimate the marginal likelihood of a given model using a ‘stepping-stone’ (or ‘path-sampling’) algorithm. These algorithms are similar to the familiar MCMC algorithms, which are intended to sample from (and estimate) the joint posterior probability of the model parameters. Stepping-stone algorithms are like a series of MCMC simulations that iteratively sample from a specified number of distributions that are discrete steps between the posterior and the prior probability distributions. The basic idea is to estimate the probability of the data for all points between the posterior and the prior—effectively summing the probability of the data over the prior probability of the parameters to estimate the marginal likelihood. Technically, the steps correspond to a series of `powerPosterior()`, where the likelihood is iteratively raised to a series of numbers between 1 and 0 (Figure 6.1). When the likelihood is raised to the power of 1 (typically the first stepping stone), samples are drawn from the (untransformed) posterior. By contrast, when the likelihood is raised to the power of 0 (typically the last stepping stone), samples are drawn from the prior. To perform a stepping-stone simulation, we need to specify (1) the number of stepping stones (power posteriors) that we will use to traverse the path between the posterior and the prior (*e.g.*, we specify 50 or 100 stones), (2) the spacing of the stones between the posterior and prior (*e.g.*, we may specify that the stones are distributed according to a beta distribution), (3) the number of samples (and their thinning) to be drawn from each stepping stone, and (4) the direction we will take (*i.e.*, from the posterior to the prior or vice versa).

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

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

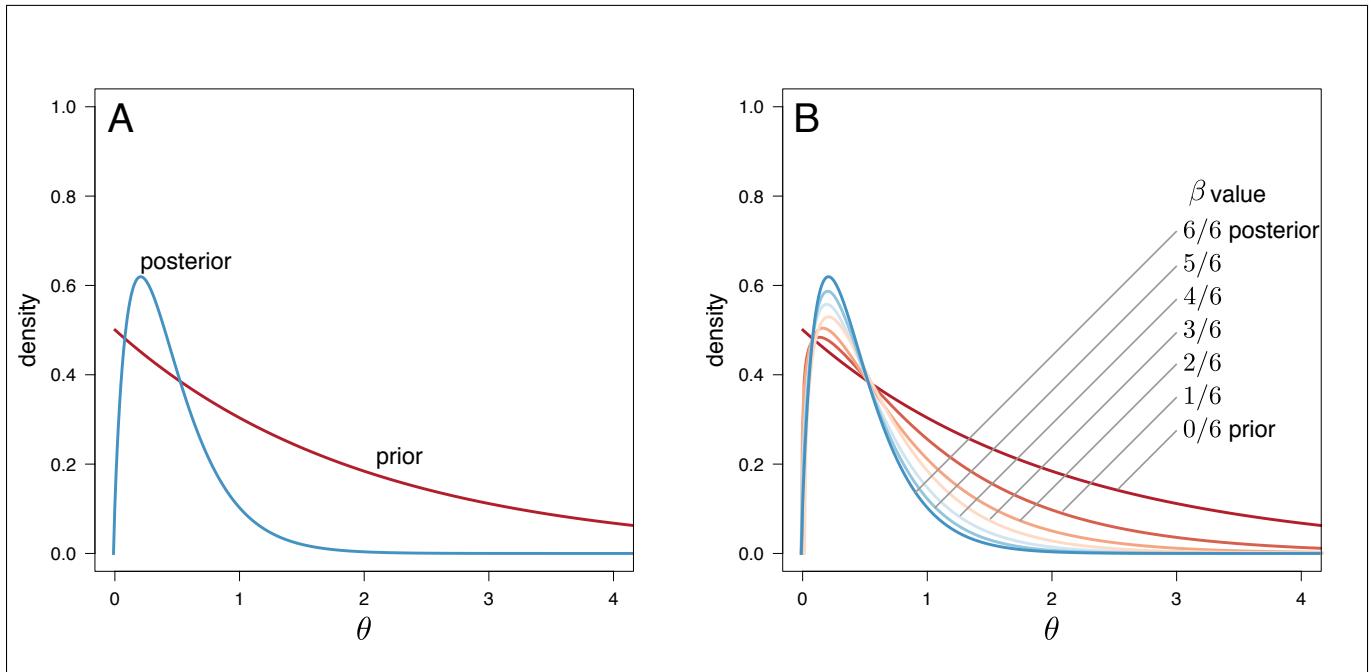


Figure 6.1: Estimating marginal likelihoods using stepping-stone simulation. Estimating the marginal likelihood involves integrating the likelihood of the data over the entire prior probability density for the model parameters. MCMC algorithms target the posterior probability density, which is typically concentrated in a small region of the prior probability density (A). Accordingly, standard MCMC simulation cannot provide unbiased estimates of the marginal likelihood because it will typically fail to explore most of the prior density. (B) Stepping-stone algorithms estimate the marginal likelihood by means of a series of MCMC-like simulations, where the likelihood is iteratively raised to a series of powers, effectively forcing the simulation to more fully explore the prior density of the model parameters. Here, six uniformly spaced stones span the posterior, where the power posterior is $\beta = 6/6 = 1$, to the prior, where the power posterior is $\beta = 0/6 = 0$.

```
clear()
```

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

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

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

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

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

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

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

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

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

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

These commands will execute a stepping-stone simulation with 50 stepping stones, sampling 1000 states from each step. Compute the marginal likelihood under stepping-stone sampling using the member function **marginal()** of the **ss** variable and record the value in Table 6.2.

```
ss.marginal()
```

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

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

Compute the marginal likelihood under stepping-stone sampling using the member function **marginal()** of the **ps** variable and record the value in Table 6.2.

```
ps.marginal()
```

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

We have kept this description of how to use stepping-stone-sampling and path-sampling very generic and did not provide the information about the model here. Our main motivation is to show that the marginal likelihood estimation algorithms are independent of the model. Thus, you can apply these algorithms to any model, *e.g.*, relaxed clock models and birth-death models, as well.

Exercises

- Compute the marginal likelihoods of the *cytb* alignment for the following substitution models:
 - Jukes-Cantor (JC) substitution model
 - Hasegawa-Kishino-Yano (HKY) substitution model
 - General-Time-Reversible (GTR) substitution model
 - GTR with gamma distributed-rate model (GTR+G)
 - GTR with invariable-sites model (GTR+I)
 - GTR+I+G model
- Enter the marginal likelihood estimate for each model in the corresponding cell of Table 6.2.
- Which is the best fitting substitution model?

Table 6.2: Estimated marginal likelihoods for different substitution models for the *cytb* alignment*.

Substitution Model	Marginal lnL estimates	
	Stepping-stone	Path sampling
JC (M_1)		
HKY (M_2)		
GTR (M_3)		
GTR+Γ (M_4)		
GTR+I (M_5)		
GTR+Γ+I (M_6)		

*you can edit this table

Computing Bayes Factors and Model Selection

Now that we have estimates of the marginal likelihood for each of our the candidate substitution models, we can evaluate their relative fit to the datasets using Bayes factors. Phylogenetic programs log-transform the likelihood values to avoid [underflow](#): multiplying likelihoods (numbers < 1) generates numbers that are too small to be held in computer memory. Accordingly, we need to use a different form of equation 6.3 to calculate the ln-Bayes factor (we will denote this value \mathcal{K}):

$$\mathcal{K} = \ln[BF(M_0, M_1)] = \ln[\mathbb{P}(\mathbf{X} | M_0)] - \ln[\mathbb{P}(\mathbf{X} | M_1)], \quad (6.4)$$

where $\ln[\mathbb{P}(\mathbf{X} | M_0)]$ is the *marginal lnL* estimate for model M_0 . The value resulting from equation 6.4 can be converted to a raw Bayes factor by simply taking the exponent of \mathcal{K}

$$BF(M_0, M_1) = e^{\mathcal{K}}. \quad (6.5)$$

Alternatively, you can directly interpret the strength of evidence in favor of M_0 in log space by comparing the values of \mathcal{K} to the appropriate scale (Table 6.1, second column). In this case, we evaluate \mathcal{K} in favor of model M_0 against model M_1 so that:

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

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

Using the values you entered in Table 6.2 and equation 6.4, calculate the ln-Bayes factors (using \mathcal{K}) for each model comparison. Enter your answers in Table 6.3 using the stepping-stone and the path-sampling estimates of the marginal log-likelihoods.

Table 6.3: Bayes factor calculation*.

Model comparison	M_1	M_2	M_3	M_4	M_5	M_6
M_1	-					
M_2		-				
M_3			-			
M_4				-		
M_5					-	
M_6						-

*you can edit this table

For your consideration...

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

Accommodating Model Uncertainty

In some or many situations the number of possible models to compare is large, *e.g.*, choosing all possible combinations of substitution models (Huelsenbeck et al. 2004). Furthermore, imagine, for example, that there are several (possibly many) alternative models that provide a similarly good fit to our given dataset. In such scenarios, conditioning inference on *any* single model (even the ‘best’) ignores uncertainty in the chosen model, which can cause estimates to be biased. This is the issue of *model uncertainty*. The Bayesian framework provides a natural approach for accommodating model uncertainty by means of *model averaging*; we simply adopt the perspective that models (like standard parameters) are random variables, and integrate the inference over the distribution of candidate models. We will demonstrate how to accommodate model uncertainty using **RevBayes** in a separate tutorial, RB_ModelAveraging_Tutorial.

Assessing Model Adequacy

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

Bibliography

- Baele, G., P. Lemey, T. Bedford, A. Rambaut, M. Suchard, and A. Alekseyenko. 2012. Improving the accuracy of demographic and molecular clock model comparison while accommodating phylogenetic uncertainty. *Molecular Biology and Evolution* 29:2157–2167.
- Baele, G., W. Li, A. Drummond, M. Suchard, and P. Lemey. 2013. Accurate model selection of relaxed molecular clocks in bayesian phylogenetics. *Molecular Biology and Evolution* 30:239–243.
- Fan, Y., R. Wu, M.-H. Chen, L. Kuo, and P. O. Lewis. 2011. Choosing among partition models in Bayesian phylogenetics. *Molecular Biology and Evolution* 28:523–532.
- Hasegawa, M., H. Kishino, and T. Yano. 1985. Dating of the human-ape splitting by a molecular clock of mitochondrial DNA. *Journal of Molecular Evolution* 22:160–174.

- Höhna, S., M. J. Landis, and T. A. Heath. 2017. Phylogenetic inference using RevBayes. *Current Protocols in Bioinformatics* .
- Huelsenbeck, J., B. Larget, and M. Alfaro. 2004. Bayesian phylogenetic model selection using reversible jump Markov chain Monte Carlo. *Molecular Biology and Evolution* 21:1123.
- Jeffreys, H. 1961. *The Theory of Probability*. Oxford University Press.
- Jukes, T. and C. Cantor. 1969. Evolution of protein molecules. *Mammalian Protein Metabolism* 3:21–132.
- Kass, R. and A. Raftery. 1995. Bayes factors. *Journal of the American Statistical Association* 90:773–795.
- Lartillot, N. 2006. Conjugate Gibbs sampling for Bayesian phylogenetic models. *Journal of Computational Biology* 13:1701–1722.
- Lavine, M. and M. J. Schervish. 1999. Bayes factors: What they are and what they are not. *The American Statistician* 53:119–122.
- Suchard, M. A., R. E. Weiss, and J. S. Sinsheimer. 2001. Bayesian selection of continuous-time Markov chain evolutionary models. *Molecular Biology and Evolution* 18:1001–1013.
- Tavaré, S. 1986. Some probabilistic and statistical problems in the analysis of DNA sequences. *Some Mathematical Questions in Biology: DNA Sequence Analysis* 17:57–86.
- Xie, W., P. Lewis, Y. Fan, L. Kuo, and M. Chen. 2011. Improving marginal likelihood estimation for Bayesian phylogenetic model selection. *Systematic Biology* 60:150–160.
- Yang, Z. 1994. Maximum likelihood phylogenetic estimation from DNA sequences with variable rates over sites: Approximate methods. *Journal of Molecular Evolution* 39:306–314.

Part V

More on Substitution Models

Chapter 7

Partitioned Data Analysis

Overview

This tutorial provides the second protocol from our recent publication ([Höhna et al. 2017](#)). The first protocol is described in the [Substitution model tutorial](#) and the third protocol is described in the [Bayes factor tutorial](#).

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

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

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

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

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

Requirements

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

- [Getting started](#)
- [Rev basics](#)
- [Substitution model](#)
- [Model selection using Bayes factors](#)

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

Data and files

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

- **primates_and_galeopterus_cytb.nex**: Alignment of the *cytochrome b* subunit from 23 primates representing 14 of the 16 families (*Indriidae* and *Callitrichidae* are missing).
- **primates_and_galeopterus_cox2.nex**: Alignment of the *COX-II* gene from the same 23 primates species.

Introduction & Background

Variation in the evolutionary process across the sites of nucleotide sequence alignments is well established, and is an increasingly pervasive feature of datasets composed of gene regions sampled from multiple loci and/or different genomes. Inference of phylogeny from these data demands that we adequately model the underlying process heterogeneity; failure to do so can lead to biased estimates of phylogeny and other parameters (Brown and Lemmon 2007).

Accounting for process heterogeneity involves adopting a partitioned data approach (sometimes also called a ‘mixed-model’ approach (Ronquist and Huelsenbeck 2003)), in which the sequence alignment is first parsed into a number of data subsets that are intended to capture plausible process heterogeneity within the data. The determination of the partitioning scheme is guided by biological considerations regarding the dataset at hand. For example, we might wish to evaluate possible variation in the evolutionary process within a single gene region (*e.g.*, between stem and loop regions of ribosomal sequences), or among gene regions in a concatenated alignment (*e.g.*, comprising multiple nuclear loci and/or gene regions sampled from different genomes). The choice of partitioning scheme is up to the investigator and many possible partitions might be considered for a typical dataset.

In this exercise, we assume that each data subset 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 rooted tree (Ψ) and branch lengths. When we assume different GTR+ Γ models for each data subset, this results in a composite model, in which all sites are assumed to share a common, rooted tree topology and proportional branch lengths, but subsets of sites are assumed to have independent substitution model parameters. Finally, we perform a separate MCMC simulation to approximate the joint posterior probability density of the phylogeny and other parameters.

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, partition-model selection is based on *Bayes factors* (*e.g.*, Suchard et al. 2001), which involves first calculating the marginal likelihood under each candidate partition scheme and then comparing the ratio of the marginal likelihoods for the set of candidate partition schemes (Brändley et al. 2005; Nylander et al. 2004; McGuire et al. 2007). The analysis pipeline that we will use in this tutorial is depicted in Figure 7.1.

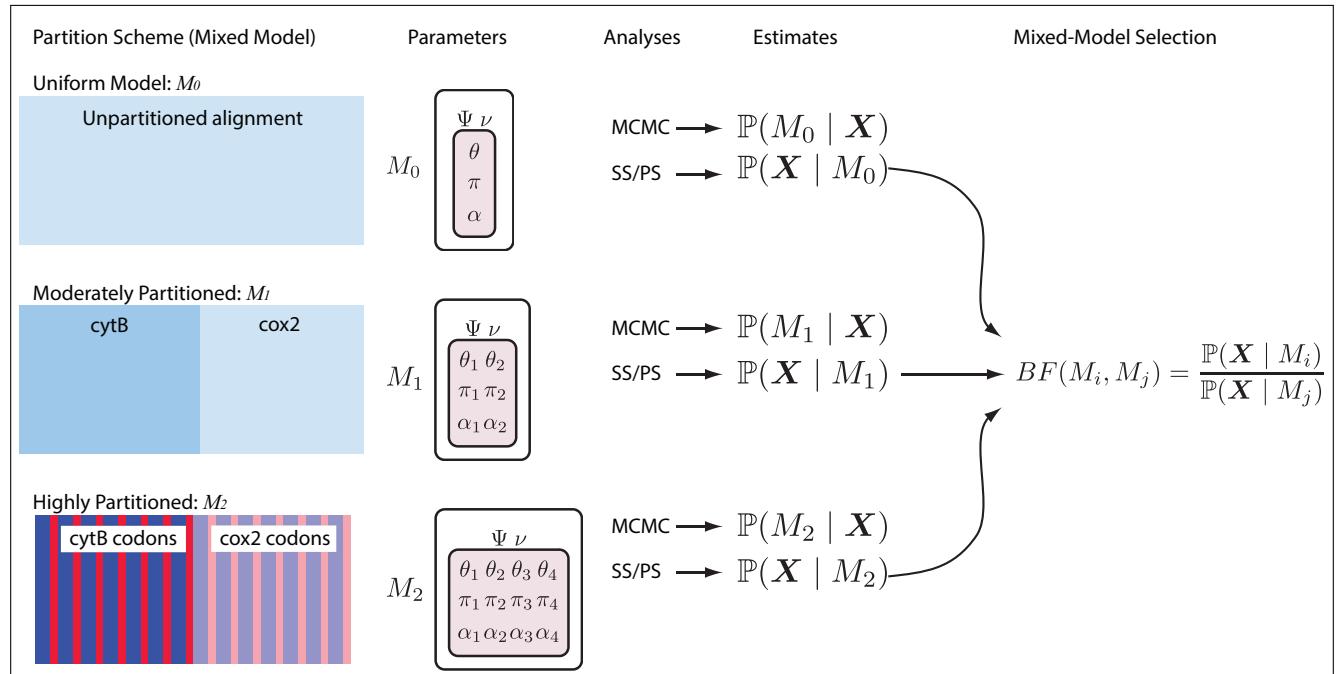


Figure 7.1: The analysis pipeline for Exercise 1. We will explore three partition schemes for the primates dataset. The first model (the ‘uniform model’, M_0) assumes that all sites evolved under a common GTR+ Γ substitution model. The second model (the ‘moderately partitioned’ model, M_1) invokes two data subsets corresponding to the two gene regions (cytB and cox2), and assumes each subset of sites evolved under an independent GTR+ Γ model. The final partition model (the ‘highly partitioned’ model, M_2) invokes four data subsets—the first two subsets corresponds to the cytB gene region, where the first and second codon position sites are combined into one subset distinct from the third codon position sites, and the cox2 gene has two subsets of its own, partitioned by codon positions in the same way—and each data subset 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, for each of the candidate partition schemes. We perform two separate sets of analyses for each partition model—a MCMC simulation to approximate the joint posterior probability density of the partition-model parameters, and a ‘power-posterior’ 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 partition models.

Concatenated, Non-partitioned

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

Setting up the model

Loading and preparing the data

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

```
data_cox2 = readDiscreteCharacterData("data/primates_and_galeopterus_cox2.nex")
data_cytb = readDiscreteCharacterData("data/primates_and_galeopterus_cytb.nex")
```

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

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

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

```
DNA character matrix with 23 taxa and 1852 characters
=====
Origionation:           primates_and_galeopterus_cox2.nex
Number of taxa:         23
Number of included taxa: 23
Number of characters:   1852
Number of included characters: 1852
Datatype:               DNA
```

For later use, we will store the taxon information (`taxa`) and the number of taxa and branches.

```
n_species <- data.ntaxa()
n_branches <- 2 * n_species - 3
taxa <- data.taxon()
```

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

```
mvi = 0
mni = 0
```

Substitution model

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

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

and assign its moves

```
moves[++mvi] = mvBetaSimplex(er, alpha=10, tune=true, weight=3)
moves[++mvi] = mvDirichletSimplex(er, alpha=10.0, tune=true, weight=1.0)
```

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

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

Now add the simplex scale move on the stationary frequencies to the moves vector

```
moves[++mvi] = mvBetaSimplex(pi, alpha=10, tune=true, weight=2)
moves[++mvi] = mvDirichletSimplex(pi, alpha=10.0, tune=true, weight=1.0)
```

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

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

Among site rate variation

We will also assume that the substitution rates vary among sites according to an one-parametric gamma distribution, *i.e.*, where the shape equals the rate ($\alpha = \beta$) and thus with mean 1.0 (Yang 1994). Since we do not have good prior knowledge about the variance in site rates, thus we can place a relative diffuse lognormal prior on the shape parameter. Create a constant node called **alpha_prior_mean** for the mean parameter and a constant node called **alpha_prior_sd** for the standard deviation of the lognormal prior on the gamma-shape parameter:

```
alpha_prior_mean <- ln(2.0)
alpha_prior_sd <- 0.587405
```

Then create a stochastic node called **alpha** with a lognormal prior:

```
alpha ~ dnLognormal(alpha_prior_mean, alpha_prior_sd)
```

The way the ASRV model is implemented involves discretizing the mean-one gamma distribution into a set number of rate categories. Thus, we can analytically marginalize over the uncertainty in the rate

at each site. To do this, we need a deterministic node that is a vector of rates calculated from the gamma distribution and the number of rate categories. The `fnDiscretizeGamma()` function returns this deterministic node and takes three arguments: the shape and rate of the gamma distribution and the number of categories. Since we want to discretize a mean-one gamma distribution, we can pass in `alpha` for both the shape and rate.

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

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

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

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

Invariant sites

Invariant sites (sites that remain fixed throughout their evolutionary history) may be seen as an extreme case of among-site rate variation. In contrast to $+\Gamma$ models, the $+I$ model allows site some probability of having substitution rate equal to zero. Here, we give the probability of a site being invariant with `pinvar`

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

Tree prior

The tree topology and branch lengths are also stochastic nodes in our model.

For simplicity, we will use the same prior distribution on the tree topology, a uniform topology prior, and branch lengths, independent exponential prior distributions, as done in the [substitution model tutorial](#).

We will assume that all possible labeled, unrooted tree topologies have equal probability. This is the `dnUniformTopology()` distribution in `RevBayes`. Note that in `RevBayes` it is advisable to specify the outgroup for your study system if you use an unrooted tree prior, whereas other software, *e.g.*, `MrBayes` uses the first taxon in the data matrix file as the outgroup. Specify the `topology` stochastic node by passing in the tip labels `names` to the `dnUniformTopology()` distribution:

```
out_group = clade("Galeopterus_variegatus")
topology ~ dnUniformTopology(taxa, outgroup=out_group)
```

To update 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 proposal `weight`.

```
moves[++mvi] = mvNNI(topology, weight=1.0)
moves[++mvi] = mvSPR(topology, weight=1.0)
```

The weight specifies how often the move will be applied either on average per iteration or relative to all other moves. Have a look at the [MCMC Diagnosis tutorial](#) for more details about moves and MCMC strategies (found on the [RevBayes Tutorials Website](#)).

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

```
for (i in 1:n_branches) {
    br_lens[i] ~ dnExponential(10.0)
    moves[++mvi] = mvScale(br_lens[i])
}
```

It is convenient for monitoring purposes to add the tree length as deterministic variable. The tree length is simply the sum of all branch lengths. . Accordingly, the tree length can be computed using the `sum()` function, which calculates the sum of any vector of values.

```
TL := sum(br_lens)
```

Finally, we can create a *phylogram* (a phylogeny in which the branch lengths are proportional to the expected number of substitutions/site) by combining the tree topology and branch lengths. We do this 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 `psi` variable is a deterministic node:

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

→ For more information on tree priors please read the [RB_DiversificationRate_Tutorial](#).

Putting it All Together

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

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

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

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

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

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

Specifying 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.

```
monitors[++mni] = mnModel(filename="output/PS_uniform.log", printgen=10)
```

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

```
monitors[++mni] = mnFile(psi, filename="output/PS_uniform.trees", printgen=10)
```

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

```
monitors[++mni] = mnScreen(alpha, pinvar, TL, printgen=1000)
```

Initializing and Running the MCMC Simulation

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

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

Note that this will automatically run two independent replicated MCMC simulations because we specified `nruns=2`.

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

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

Now, run the MCMC:

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

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

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

```
treetrace = readTreeTrace("output/PS_uniform.trees", treetype="non-clock")
treetrace.summarize()
```

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

```
map_tree = mapTree(treetrace,"output/PS_uniform_map.tre")
```

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

Partitioning by Gene Region

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

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

```
clear()
```

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

```
filenames <- v("data/primates_and_galeopterus_cox2.nex", "data/
primates_and_galeopterus_cytb.nex")
```

Set a variable for the number of partitions:

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

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

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

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

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

```
mvi = 0
mni = 0
```

Specify the Parameters by Looping Over Partitions

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

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

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

```
for (i in 1:n_data_subsets) {
  er_prior[i] <- v(1,1,1,1,1,1)
  er[i] ~ dnDirichlet(er_prior[i])
  moves[++mvi] = mvBetaSimplex(er[i], alpha=10, tune=true, weight=3)
}
```

and stationary frequencies

```
for (i in 1:n_data_subsets) {
    pi_prior[i] <- v(1,1,1,1)
    pi[i] ~ dnDirichlet(pi_prior[i])
    moves[++mvi] = mvBetaSimplex(pi[i], alpha=10, tune=true, weight=2)
}
```

then passing those parameters into a rate matrix function

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

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

```
for (i in 1:n_data_subsets) {
    alpha_prior_mean[i] <- ln(2.0)
    alpha_prior_sd[i] <- 0.587405
    alpha[i] ~ dnLognormal(alpha_prior_mean[i], alpha_prior_sd[i])
    gamma_rates[i] := fnDiscretizeGamma(alpha[i], alpha[i], 4, false)

    moves[++mvi] = mvScale(alpha[i], weight=2)
}
```

the $+I$ invariant sites model

```
for (i in 1:n_data_subsets) {
    pinvar[i] ~ dnBeta(1,1)
    moves[++mvi] = mvScale(pinvar[i], lambda=0.1, tune=true, weight=2.0)
    moves[++mvi] = mvSlide(pinvar[i], delta=0.1, tune=true, weight=2.0)
}
```

and the per-partition substitution rate multipliers

```
# specify a rate multiplier for each partition
part_rate_mult ~ dnDirichlet(rep(1.0, n_data_subsets))
moves[++mvi] = mvBetaSimplex(part_rate_mult, alpha=1.0, tune=true, weight=n_data_subsets)
moves[++mvi] = mvDirichletSimplex(part_rate_mult, alpha=1.0, tune=true, weight=2.0)

# note that we use here a vector multiplication,
# i.e., multiplying each element of part_rate_mult by n_data_subsets
part_rate := part_rate_mult * n_data_subsets
```

Different Substitution Models for each Gene

Alternatively, we might be interested in applying different substitution models for each gene independently instead of assuming the same substitution albeit with different parameters for each gene. In this two gene case this is rather simple to do by specifying the substitution model for each gene independently. For many genes this might become lengthy and you might want to write a script to generate this section (note: we may provide such scripts soon).

For simplicity and sake of demonstration, we assume that the cytochrome b region evolves under a Jukes-Cantor substitution model and the COX-II gene under an HKY substitution model. We begin with the cytochrome b gene and the Jukes-Cantor substitution model:

```
# specify the JC rate matrix
Q[1] <- fnJC(4)
```

Second, we specify the HKY substitution model for the COX-II gene:

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

# specify a move to propose updates to on pi
moves[++mvi] = mvBetaSimplex(pi, weight=2)
moves[++mvi] = mvDirichletSimplex(pi, weight=1)

# specify a lognormal distribution as the prior distribution on kappa
kappa ~ dnLognormal(0.0,1.25)

# a simple scaling move to update kappa
moves[++mvi] = mvScale(kappa)

# Finally, create the HKY rate matrix
Q[2] := fnHKY(kappa,pi)
```

Note that we specified manually in this way our vector of rate matrices **Q**. We can thus specify any substitution model manually for a given gene. We hope that this brief example conveys the idea how to specify gene-specific substitution models. You can add rate-variation among sites and/or probabilities for a site being invariant for each gene too. Finally, you can then either loop over all genes to create the **dnPhyloCTMC** distribution (see below) if the structure of the model allows it (*i.e.*, if all models have a variable for site-rate-variation and probabilities for invariant site), or you efficiently set these variables to default values (*e.g.*, **pinvar[i]=0.0** if there is no probability for a site being invariant for this gene), or you create the **seq[i] ~ dnPhyloCTMC(...)** manually outside a loop as well.

Tree prior

We assume that both genes evolve along the same tree. Hence, we need to specify a random variable for our tree parameter which is the same as was specified for `mcmc_Partition_uniform.Rev`.

```

out_group = clade("Galeopterus_variegatus")
# Prior distribution on the tree topology
topology ~ dnUniformTopology(taxa, outgroup=out_group)
moves[++mvi] = mvNNI(topology, weight=5.0)
moves[++mvi] = mvSPR(topology, weight=1.0)

# Branch length prior
for (i in 1:n_branches) {
    bl[i] ~ dnExponential(10.0)
    moves[++mvi] = mvScale(bl[i])
}

TL := sum(bl)

psi := treeAssembly(topology, bl)

```

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 `psi` node and their log-likelihoods are added to get the likelihood of the whole DAG.

```

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

```

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

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

Create monitors

create the monitors

```
monitors[++mni] = mnModel(filename="output/PS_gene.log", printgen=10)
monitors[++mni] = mnFile(psi, filename="output/PS_gene.trees", printgen=100)
monitors[++mni] = mnScreen(TL, printgen=1000)
```

configure and run the MCMC analysis

```
mymcmc = mcmc(mymodel, moves, monitors, nruns=2)
mymcmc.burnin(10000, 100)
mymcmc.run(30000)
```

and summarize the posterior density of trees with a MAP tree

```
treetrace = readTreeTrace("output/PS_gene.trees", treetype="non-clock")
treetrace.summarize()
mapTree(treetrace, "output/PS_gene_MAP.tre")
```

Partitioning by Codon Position and by Gene

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

```
clear()
data_cox2 <- readDiscreteCharacterData("data/primates_and_galeopterus_cox2.nex")
data_cytb <- readDiscreteCharacterData("data/primates_and_galeopterus_cytb.nex")
```

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

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

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

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

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

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

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

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

Now we have a data vector containing each subset. We can then specify the independent substitution models per data subset. The remaining parts of the model are identical to the previous exercise where we partitioned by gene.

- Don't forget to rename the output files!

Exercises

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

Aside from the tree topology and branch lengths, each data subset is modeled to have its own set of parameters. However, the posterior estimates for some parameters appear quite similar between some pairs of subsets yet different between other pairs of subsets. For example, `part_rate` is the per-subset substitution rate. This clock is approximately one order of magnitude faster for partitions 2 and 4 (third codon position sites) than it is for subsets 1 and 3 (non-third codon position sites).

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

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

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

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

Instead, configure and run a power posterior analysis

```
pow_p = powerPosterior(mymodel, moves, monitors, "output/model_uniform.out", cats
                      =100)
pow_p.burnin(generations=5000,tuningInterval=200)
pow_p.run(generations=2000)
```

then compute the marginal likelihood using the stepping stone sampler

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

and again using the path sampler

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

Bibliography

- Brändley, M. C., A. Schmitz, and T. W. Reeder. 2005. Partitioned bayesian analyses, partition choice, and the phylogenetic relationships of scincid lizards. *Systematic Biology* 54:373–390.
- Brown, J. M. and A. R. Lemmon. 2007. The importance of data partitioning and the utility of Bayes factors in Bayesian phylogenetics. *Systematic Biology* 56:643–655.
- Höhna, S., M. J. Landis, and T. A. Heath. 2017. Phylogenetic inference using RevBayes. *Current Protocols in Bioinformatics* .
- McGuire, J. A., C. C. Witt, D. L. Altshuler, and J. Remsen. 2007. Phylogenetic systematics and biogeography of hummingbirds: Bayesian and maximum likelihood analyses of partitioned data and selection of an appropriate partitioning strategy. *Systematic Biology* 56:837–856.
- Nylander, J. A., F. Ronquist, J. P. Huelsenbeck, and J. Nieves-Aldrey. 2004. Bayesian phylogenetic analysis of combined data. *Systematic Biology* 53:47–67.

- Ronquist, F. and J. Huelsenbeck. 2003. MrBayes 3: Bayesian phylogenetic inference under mixed models. *Bioinformatics* 19:1572–1574.
- Suchard, M. A., R. E. Weiss, and J. S. Sinsheimer. 2001. Bayesian selection of continuous-time Markov chain evolutionary models. *Molecular Biology and Evolution* 18:1001–1013.
- Yang, Z. 1994. Maximum likelihood phylogenetic estimation from DNA sequences with variable rates over sites: Approximate methods. *Journal of Molecular Evolution* 39:306–314.

Part VI

Divergence Time Estimation

Chapter 8

Overview

Background

Estimating nodes ages (*i.e.*, branch lengths in proportion to time) is confounded by the fact that the rate of evolution and time are intrinsically linked when inferring genetic differences between species.

$$\text{branch length} = \text{rate} \times \text{time} \quad (8.1)$$

A model of substitution rate and divergence time must be applied to tease apart rate and time. When applied in methods for divergence time estimation, the resulting trees have branch lengths that are proportional to time. External node age estimates from the fossil record or other sources are necessary for inferring the real-time (or absolute) ages of lineage divergences (Figure ??).

Modeling lineage-specific substitution rates

Many factors can influence the rate of substitution in a population such as mutation rate, generation time, and selection. As a result, many models have been proposed that describe how substitution rate may vary across the Tree of Life. The simplest model, the molecular clock, assumes that the rate of substitution remains constant over time (Zuckerkandl and Pauling 1962). However, many studies have shown that molecular data (in general) violate the assumption of a molecular clock and that there exists considerable variation in the rates of substitution among lineages.

Several models have been developed and implemented for inferring divergence times without assuming a strict molecular clock and are commonly applied to empirical data sets. Some models assume that rates are heritable and autocorrelated over the tree, others model rate change as a step-wise process, and others assume that the rates on each branch are independently drawn from a single distribution. Many of these models have been applied as priors using Bayesian inference methods. The implementation of dating methods in a Bayesian framework provides a flexible way to model rate variation and obtain reliable estimates of speciation times, provided the assumptions of the models are adequate. When coupled with numerical methods, such as MCMC, for approximating the posterior probability distribution of parameters, Bayesian methods are extremely powerful for estimating the parameters of a statistical model and are widely used in phylogenetics.

Some models of lineage-specific rate variation:

Global molecular clock: a constant rate of substitution over time (Zuckerkandl and Pauling 1962).

Local molecular clocks: Closely related lineages share the same rate and rates are clustered by sub-clades (Kishino et al. 1990; Rambaut and Bromham 1998; Yang and Yoder 2003; Drummond and Suchard 2010)

Compound Poisson process: Rate changes occur along lineages according to a point process and at rate-change events, the new rate is a product of the old rate and a Γ -distributed multiplier (Huelsenbeck et al. 2000).

Autocorrelated rates: substitution rates evolve gradually over the tree

- *Log-normally distributed rates:* the rate at a node is drawn from a log-normal distribution with a mean equal to the parent rate (Thorne et al. 1998; Kishino et al. 2001; Thorne and Kishino 2002)
- *Cox-Ingersoll-Ross Process:* the rate of the daughter branch is determined by a non-central χ^2 distribution. This process includes a parameter that determines the intensity of the force that drives the process to its stationary distribution (Lepage et al. 2006).

Uncorrelated rates: The rate associated with each branch is drawn from a single underlying parametric distribution such as an exponential or log-normal (Drummond et al. 2006; Rannala and Yang 2007; Lepage et al. 2007).

Mixture model on branch rates: Branches are assigned to distinct rate categories according to a Dirichlet process (Heath et al. 2012).

Priors on node times

There are many components that make up a Bayesian analysis of divergence time. One that is often overlooked is the prior on node times, often called a tree prior. This model describes how speciation events are distributed over time. When this model is combined with a model for branch rate, Bayesian inference allows you to estimate relative divergence times. Furthermore, because the rate and time are confounded in the branch-length parameter, the prior describing the branching times may have a strong effect on divergence time estimation.

We can separate the priors on node ages into different categories:

Phenomenological: models that make no explicit assumptions about the biological processes that generated the tree. These priors are conditional on the age of the root.

- *Uniform distribution:* This simple model assumes that internal nodes are uniformly distributed between the root and tip nodes (Lepage et al. 2007; Ronquist et al. 2012b).
- *Dirichlet distribution:* A flat Dirichlet distribution describes the placement of internal nodes on every path between the root and tips (Kishino et al. 2001; Thorne and Kishino 2002).

Mechanistic: models that describe the biological processes responsible for generating the pattern of lineage divergences.

- *Population-level processes:* Coalescent processes describe the time, in generations, between coalescent events and allow for the estimation of population-level parameters (Kingman 1982). Furthermore, they describe demographic processes (suitable for describing differences among individuals in the same species/population).
- *Species-level processes:* Birth-death stochastic branching models describe lineage diversification (suitable for describing the timing of divergences between samples from different species) (Kendall 1948; Thompson 1975; Nee et al. 1994; Rannala and Yang 1996; Yang and Rannala 1997; Höhna 2015).

We cover the different process that can be used for tree and divergence time prior distributions in detail in the [RB_DiversificationRate_Tutorial](#) because of their interest in estimating diversification rates and patterns.

Calibration to absolute time

Without external information to calibrate the tree, divergence time estimation methods can only reliably provide estimates of relative divergence times and not absolute node ages; or if one would know the (average) rate of substitutions. Calibration information can come from a variety of sources including “known” substitution rates (often secondary calibrations estimated from a previous study), dated tip sequences

from serially sampled data (time-stamped virus data or ancient DNA), or geological date estimates (fossils or biogeographical data). Age estimates from fossil organisms are the most common form of divergence time calibration information. These data are used as age constraints on their putative ancestral nodes. There are numerous difficulties with incorporating node age estimates from fossil data including disparity in fossilization and sampling, uncertainty in dating, and correct phylogenetic placement of the fossil. Thus, it is critical that careful attention is paid to the paleontological data included in phylogenetic divergence time analyses. With an accurately dated and identified fossil in hand, further consideration is required to determine how to apply the node-age constraint. If the fossil is truly a descendant of the node it calibrates, then it provides a reliable minimum age bound on the ancestral node time. However, maximum bounds are far more difficult to come by.

Bayesian methods provide a way to account for uncertainty in fossil calibrations. Prior distributions reflecting our knowledge (or lack thereof) of the amount of elapsed time from the ancestral node to its calibrating fossil are easily incorporated into these methods. A nice review paper by [Ho and Phillips \(2009\)](#) outlines a number of different parametric distributions appropriate for use as priors on calibrated nodes.

Uniform distribution: Typically, you must have both maximum and minimum age bounds when applying a uniform calibration prior (though some methods are available for applying uniform constraints with soft bounds). The minimum bound is provided by the fossil member of the clade. The maximum bound may come from a bracketing method or other external source. This distribution places equal probability across all ages spanning the interval between the lower and upper bounds.

Normal distribution: When applying a biogeographical date (*e.g.*, the Isthmus of Panama) or a secondary calibration (a node age estimate from a previous study), the normal distribution can be a useful calibration prior. This distribution is always symmetrical and places the greatest prior weight on the mean. Its scale is determined by the standard deviation parameter.

Exponential distribution: The exponential distribution is characterized by a single rate parameter and is useful for calibration if the fossil age is very close to the age of its ancestral node. The expected (mean) age difference under this distribution is equal to the $1/\text{rate}$. Under the exponential distribution, the greatest prior weight is placed on node ages very close to the age of the fossil with diminishing probability to ∞ . As the rate parameter is increased, this prior density becomes strongly informative, whereas very low values of the rate result in a fairly non-informative prior (Figure 3a).

Log-normal distribution: An offset, log-normal prior on the calibrated node age places the highest probability on ages somewhat older than the fossil, with non-zero probability to ∞ .

Integrating Fossil Occurrence Times in the Speciation Model

Calibrating Bayesian divergence-time estimates using parametric densities (as described in the previous section: Sec. 8.1.3) are typically applied in a multiplicative manner such that the prior probability of a calibrated node age is the product of the probability coming from the tree-wide speciation model and the probability under the calibration density ([Heled and Drummond 2012; Warnock et al. 2012; 2015](#)). However, when using fossil information, it is also possible to account for the fact that the fossils are part of the same diversification process (*i.e.*, birth-death model) that generated the extant species using the fossilized birth-death (FBD) model described in [Stadler \(2010\)](#) and [Heath et al. \(2014\)](#). This model simply treats the fossil observations as part of the prior on node times. The fossilized birth-death process provides a model for the distribution of speciation times, tree topology, and distribution of lineage samples before

the present (*i.e.*, non-contemporaneous samples like fossils or viruses). Thus, it provides a reasonable prior distribution for analyses combining morphological or DNA data for both extant and fossil taxa—*i.e.*, the so-called ‘total-evidence’ or ‘tip-dating’ approaches described by Ronquist et al. (2012a) (also see Pyron (2011)). When matrices of discrete morphological characters for both living and fossil species are unavailable, the fossilized birth-death model imposes a time structure on the tree by marginalizing over all possible attachment points for the fossils on the extant tree (Heath et al. 2014), therefore, some prior knowledge of phylogenetic relationships is important, much like for calibration-density approaches.

Bibliography

- Drummond, A., S. Ho, M. Phillips, and A. Rambaut. 2006. Relaxed Phylogenetics and Dating with Confidence. PLoS Biology 4:e88.
- Drummond, A. and M. Suchard. 2010. Bayesian random local clocks, or one rate to rule them all. BMC Biology 8:114.
- Heath, T., M. Holder, and J. Huelsenbeck. 2012. A dirichlet process prior for estimating lineage-specific substitution rates. Molecular Biology and Evolution 29:939–955.
- Heath, T. A., J. P. Huelsenbeck, and T. Stadler. 2014. The fossilized birth-death process for coherent calibration of divergence-time estimates. Proceedings of the National Academy of Sciences 111:E2957–E2966.
- Heled, J. and A. J. Drummond. 2012. Calibrated tree priors for relaxed phylogenetics and divergence time estimation. Systematic Biology 61:138–149.
- Ho, S. Y. and M. J. Phillips. 2009. Accounting for calibration uncertainty in phylogenetic estimation of evolutionary divergence times. Systematic Biology Page sys035.
- Höhna, S. 2015. The time-dependent reconstructed evolutionary process with a key-role for mass-extinction events. Journal of Theoretical Biology 380:321–331.
- Huelsenbeck, J. P., B. Larget, and D. L. Swofford. 2000. A compound Poisson process for relaxing the molecular clock. Genetics 154:1879–1892.
- Kendall, D. G. 1948. On the generalized "birth-and-death" process. The Annals of Mathematical Statistics 19:1–15.
- Kingman, J. F. C. 1982. On the genealogy of large populations. Journal of Applied Probability 19:27–43.
- Kishino, H., T. Miyata, and M. Hasegawa. 1990. Maximum likelihood inference of protein phylogeny and the origin of chloroplasts. Journal of Molecular Evolution 31:151–160.
- Kishino, H., J. L. Thorne, and W. J. Bruno. 2001. Performance of a divergence time estimation method under a probabilistic model of rate evolution. Molecular Biology and Evolution 18:352–361.
- Lepage, T., D. Bryant, H. Philippe, and N. Lartillot. 2007. A general comparison of relaxed molecular clock models. Molecular Biology and Evolution 24:2669.
- Lepage, T., S. Lawi, P. Tupper, and D. Bryant. 2006. Continuous and tractable models for the variation of evolutionary rates. Mathematical biosciences 199:216–233.

- Nee, S., R. M. May, and P. H. Harvey. 1994. The Reconstructed Evolutionary Process. *Philosophical Transactions: Biological Sciences* 344:305–311.
- Pyron, R. A. 2011. Divergence time estimation using fossils as terminal taxa and the origins of Lissamphibia. *Systematic Biology* Page syr047.
- Rambaut, A. and L. Bromham. 1998. Estimating divergence dates from molecular sequences. *Molecular Biology and Evolution* 15:442–448.
- Rannala, B. and Z. Yang. 1996. Probability distribution of molecular evolutionary trees: A new method of phylogenetic inference. *Journal of Molecular Evolution* 43:304–311.
- Rannala, B. and Z. Yang. 2007. Inferring Speciation Times under an Episodic Molecular Clock. *Systematic Biology* 56:453–466.
- Ronquist, F., S. Klopfenstein, L. Vilhelmsen, S. Schulmeister, D. L. Murray, and A. P. Rasnitsyn. 2012a. A total-evidence approach to dating with fossils, applied to the early radiation of the Hymenoptera. *Systematic Biology* 61:973–999.
- Ronquist, F., M. Teslenko, P. van der Mark, D. L. Ayres, A. Darling, S. Höhna, B. Larget, L. Liu, M. A. Suchard, and J. P. Huelsenbeck. 2012b. MrBayes 3.2: efficient bayesian phylogenetic inference and model choice across a large model space. *Systematic Biology* 61:539–542.
- Stadler, T. 2010. Sampling-through-time in birth-death trees. *Journal of Theoretical Biology* 267:396–404.
- Thompson, E. 1975. Human evolutionary trees. Cambridge University Press Cambridge.
- Thorne, J., H. Kishino, and I. S. Painter. 1998. Estimating the rate of evolution of the rate of molecular evolution. *Molecular Biology and Evolution* 15:1647–1657.
- Thorne, J. L. and H. Kishino. 2002. Divergence time and evolutionary rate estimation with multilocus data. *Systematic Biology* 51:689–702.
- Warnock, R. C., J. F. Parham, W. G. Joyce, T. R. Lyson, and P. C. Donoghue. 2015. Calibration uncertainty in molecular dating analyses: there is no substitute for the prior evaluation of time priors. *Proceedings of the Royal Society B: Biological Sciences* 282:20141013.
- Warnock, R. C., Z. Yang, and P. C. Donoghue. 2012. Exploring uncertainty in the calibration of the molecular clock. *Biology letters* 8:156–159.
- Yang, Z. and B. Rannala. 1997. Bayesian phylogenetic inference using DNA sequences: a Markov Chain Monte Carlo Method. *Molecular Biology and Evolution* 14:717–724.
- Yang, Z. and A. D. Yoder. 2003. Comparison of likelihood and bayesian methods for estimating divergence times using multiple gene loci and calibration points, with application to a radiation of cute-looking mouse lemur species. *Systematic Biology* 52:705–716.
- Zuckerkandl, E. and L. Pauling. 1962. Molecular disease, evolution, and genetic heterogeneity. Pages 189–225 in *Horizons in Biochemistry* (M. Kasha and B. Pullman, eds.) Academic Press, New York.

Chapter 9

Node and Fossil Calibration

Overview

This tutorial covers how to estimate divergence times and time calibrated phylogenies. The key concepts of this tutorial are node- and fossil-calibrations (assuming a global molecular clock). A good overview about best practices for node- and fossil-calibrations is found in [Parham et al. \(2012\)](#), [Warnock et al. \(2012\)](#), [Joyce et al. \(2013\)](#) and [Warnock et al. \(2015\)](#).

In this tutorial you will perform a Bayesian inference to estimate a time-calibrated phylogeny. In the first part we will demonstrate you how to set up a basic model for time-calibrated phylogeny inference. Throughout we will use the global molecular clock rate. In a later tutorial we will cover other models for the molecular clock, *e.g.*, relaxed clock methods. The first analysis will use an informative prior distribution on the root age (crown age) to calibrate the phylogeny. In the second analysis you will use informative node- and fossil-calibrations instead of the informative prior on the root age. In the third analysis you will use minimum and maximum age constraint (both using hard and soft constraints) to calibrate the phylogeny. All the assumptions will be covered more in detail later in this tutorial.

Requirements

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

- RB_Getting_Started
- RB_Basics_Tutorial
- RB_CTMC_Tutorial

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

Data and files

We provide the data file of DNA sequences required for this tutorial. You may want to use your own data instead.

→ Create a folder called **data** and download the following files:

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

Below you will also find two tables with the calibration dates that we will use later in this tutorial (see Table 9.1 and Table 9.2). Table 9.1 will be used for calibrating clades/nodes with informative priors (*e.g.*, normal distributions). Table 9.2 will be used to calibrate clade/nodes by specifying minimum and maximum ages using hard and soft constraints.

Table 9.1: Node information used for calibrating divergence times in the primate tree from Perelman et al. (2011).

Clade	Age range (My)	Citation
<i>Simiiformes</i>	43 ± 4.5	Seiffert et al. (2003)
<i>Lorisiformes</i>	40 ± 3	Franzen et al. (2009); Poux and Douzery (2004)
<i>Catarrhini</i>	29 ± 6	Poux and Douzery (2004)
<i>Platyrrhini</i>	23.5 ± 3	Hodgson et al. (2009); Kay et al. (2008)
<i>Primates</i>	90 ± 6	Matsui et al. (2009); Steiper et al. (2004); Tavaré et al. (2002)

Table 9.2: Calibration intervals used in Springer et al. (2012) to calibrate nodes in the primate tree.

Clade	Min Age (My)	Max Age (My)	Citation
<i>Simiiformes</i>	28.3	56	Seiffert et al. (2003)
<i>Lorisiformes</i>	37.1	56	Franzen et al. (2009); Poux and Douzery (2004)
<i>Catarrhini</i>	20.55	37.3	Poux and Douzery (2004)
<i>Platyrrhini</i>	11.8	37.3	Hodgson et al. (2009); Kay et al. (2008)

Divergence time estimation using an informative prior on the root age

Getting Started

The first section of this exercise involves: (1) setting up a general time reversible (GTR) substitution model (Tavaré 1986) with gamma distributed rate variation among sites (Yang 1994) for an alignment of the cytochrome b subunit; (2) use an informative prior on the root age to date the phylogeny; (3) approximating the posterior probability of the tree topology and node ages (and all other parameters) using MCMC, and; (4) summarizing the MCMC output by computing the maximum *a posteriori* tree. This analysis is mostly equivalent to the analysis performed in the RB_CTMC_Tutorial.

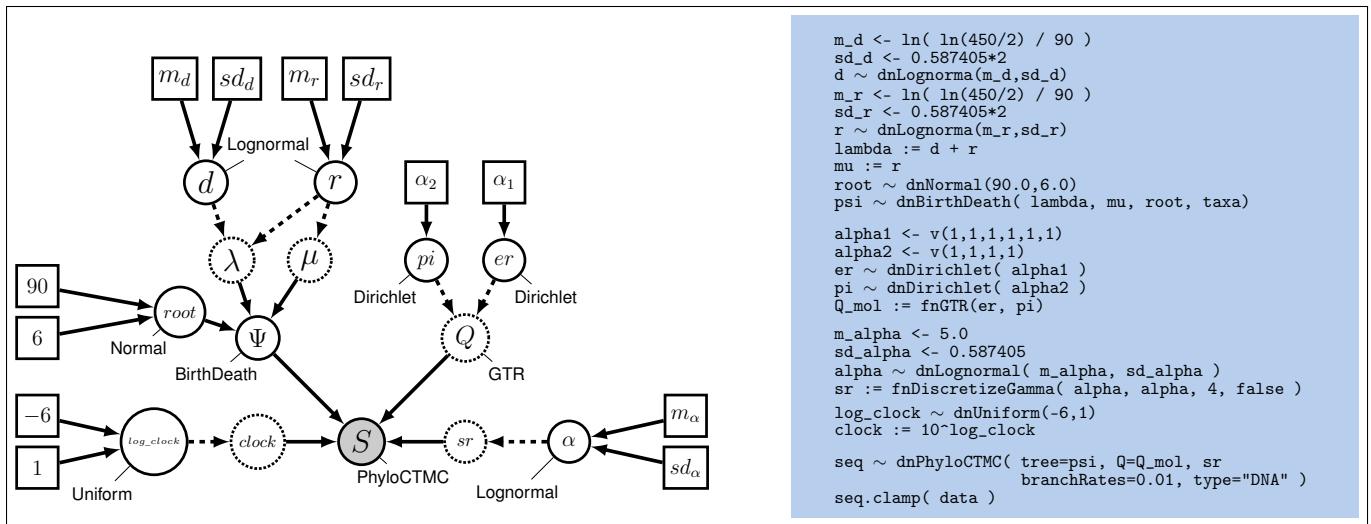


Figure 9.1: An example phylogenetic model depicted in graphical-model notation (left) and the corresponding specification in the Rev language (right). This example shows the basic outline of the root calibration that we will use in the first exercise.

The general structure of the model is represented in Figure 10.1. This figure shows the full model graph.

Loading the Data

- You should have already downloaded the data files in Section 10.2. Links to additional files, including the scripts to run these analyses can be found on the [RevBayes tutorials website](#). Remember that the data file should be in a directory called **data** that is in your current working directory.

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

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

Executing these lines initializes the data matrix as the respective Rev variable.

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

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

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

```
mvi = 0
```

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

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

General Time Reversible (GTR) Substitution Model

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

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

This node defines the concentration-parameter values of the Dirichlet prior distribution on the exchangeability rates. Now, we can create a stochastic node for the exchangeability rates using the `dnDirichlet()` function, which takes the vector of concentration-parameter values as an argument and the `~` operator. Together, these create a stochastic node named `er`, see Figure 10.1:

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

The Dirichlet prior on our parameter `er` creates a *simplex* of values that sum to 1.

For each stochastic node in our model, we must also specify a proposal mechanism if we wish to estimate that parameter.

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

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

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

The node `pi` represents the π node in Figure 10.1. Now add the simplex scale move on the stationary frequencies to the moves vector:

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

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

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

Setting up the Gamma Model

Create a constant node called `alpha_prior_mean` and a second constant node called `alpha_prior_sd` for the lognormal prior on the gamma-shape parameter (this is represented as the constant rate parameter in Figure 10.1):

```
alpha_prior_mean <- 5.0
alpha_prior_sd <- 0.587405
```

Then create a stochastic node called **alpha** with an lognormal prior (this represents the stochastic node for the α -shape parameter in Figure 10.1):

```
alpha ~ dnLognormal( alpha_prior_mean, alpha_prior_sd )
```

The way the ASRV model is implemented involves discretizing the mean-one gamma distribution into a set number of rate categories, k . To specify this, we need a deterministic node that is a vector that will hold the set of k rates drawn from the gamma distribution with k rate categories. The **fnDiscretizeGamma()** function returns this deterministic node and takes three arguments: the shape and rate of the gamma distribution and the number of categories. Since we want to discretize a mean-one gamma distribution, we can pass in **alpha** for both the shape and rate.

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

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

The random variable that controls the rate variation is the stochastic node **alpha**. We will apply a simple scale move to this parameter.

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

For more information on ASRV please read the [RB_CTMC_Tutorial](#).

Tree Prior: Tree Topology and Node Ages

The tree (the topology and node ages) is a stochastic node in our phylogenetic model. In Figure 10.1, the tree is denoted Ψ . We will assume a constant-rate birth-death process as the prior distribution on the tree. The distribution in RevBayes is **dnBDP()**. For more information on tree priors please read the [RB_DiversificationRate_Tutorial](#).

For the birth-death process we need a speciation rate and extinction rate parameter. Instead of prior distributions on these parameters directly, we will specify lognormal prior distributions on the diversification and turnover rates.

```
diversification_mean <- ln( ln(n_species/2.0) / 90 )
diversification_sd <- 0.587405*2
diversification ~ dnLognormal(mean=diversification_mean, sd=diversification_sd)
moves[++mvi] = mvScale(diversification, lambda=1.0, tune=true, weight=3.0)

turnover_mean <- ln( ln(n_species/2.0) / 90 )
turnover_sd <- 0.587405*2
turnover ~ dnLognormal(mean=turnover_mean, sd=turnover_sd)
```

```

moves[++mvi] = mvScale(turnover,lambda=1.0,tune=true,weight=3.0)

### Transform the parameters
birth_rate := diversification + turnover
death_rate := turnover

```

In our reference publication, [Perelman et al. \(2011\)](#) used a normal distribution with mean of 90.0 MYA with stdev = 6.0 as the prior distribution on the root age. The normal distribution itself is defined on the complete real line (*i.e.*, from $-\infty$ to $+\infty$), however, we know that the root age of primates is definitely larger than 0 (it happen before the present) and smaller than, say, 5000 MYA. Thus, we truncate the normal distribution. This also has the advantage that the type of the variable for the root age is a positive real number, instead of a real number.

```

root_time ~ dnNormal(mean=90.0,sd=6.0,min=0.0,max=5000.0)
moves[++mvi] = mvScale(root_time,weight=2.0)

```

Additionally, we know that we do not have all primate species included in this data set. We only have 23 out of the approximately 450 primate species. Thus, we use a sampling fraction to represent this incomplete taxon sampling ([Höhna et al. 2011](#); [Höhna 2014](#)).

```

rho <- n_species/450

```

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

```

psi ~ dnBDP(lambda=birth_rate, mu=death_rate, rho=rho, rootAge=root_time,
samplingStrategy="uniform", condition="survival", taxa=taxa)

```

Some types of stochastic nodes can be updated by a number of alternative moves. Different moves may explore parameter space in different ways, and it is possible to use multiple different moves for a given parameter to improve mixing (the efficiency of the MCMC simulation). In the case of our rooted tree, for example, we can use both a nearest-neighbor interchange move without and with changing the node ages (**mvNarrow** and **mvNNI**) and a fixed-nodeheight subtree-prune and regrafting move (**mvFNPR**). For overviews about moves on tree see [Lakner et al. \(2008\)](#), [Höhna et al. \(2008\)](#) and [Höhna and Drummond \(2012\)](#). We also need moves that change the ages of the internal nodes; which are for example the **mvSubtreeScale** and **mvNodeTimeSlideUniform**. These moves do not have tuning parameters associated with them, thus you only need to pass in the **psi** node and proposal **weight**.

```

moves[++mvi] = mvNarrow(psi, weight=5.0)
moves[++mvi] = mvNNI(psi, weight=1.0)
moves[++mvi] = mvFNPR(psi, weight=5.0)

```

```
moves[++mvi] = mvFNPR(psi, weight=2.0)
moves[++mvi] = mvSubtreeScale(psi, weight=3.0)
moves[++mvi] = mvNodeTimeSlideUniform(psi, weight=15.0)
```

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

Monitoring specific clade ages

The exercise in this tutorial involves looking at specific age estimates. There are two ways in RevBayes how to obtain age estimates. First, you can look into the generated *maximum a posteriori* tree in **FigTree**. Second, you can add deterministic variables for the ages that you are interested in and look at the values in **Tracer**. Both approaches are useful and could be used together. For the second approach to work we need to create these deterministic variables.

We start with a deterministic node monitoring the age of the *Catarrhini*. This involves create a clade object. A clade object simply contains a number of species names. Note, the names need to match **exactly**. Then, we use the **tmrca** function which will record the time of the most recent common ancestor of this clade.

```
clade_catarrhini = clade("Pan_paniscus", "Macaca_mulatta")
age_catarrhini := tmrca(psi, clade_catarrhini)
```

Next, a deterministic node monitoring the age of the *Platyrrhini*:

```
clade_platyrrhini = clade("Alouatta_palliata", "Callicebus_donacophilus")
age_platyrrhini := tmrca(psi, clade_platyrrhini)
```

Then, a deterministic node monitoring the age of the *Simiiformes*:

```
clade_simiiformes = clade("Cebus_albifrons", "Macaca_mulatta")
age_simiiformes := tmrca(psi, clade_simiiformes)
```

And finally, a deterministic node monitoring the age of the *Lorisiformes*:

```
clade_lorisiformes = clade("Loris_tardigradus", "Galago_senegalensis")
age_lorisiformes := tmrca(psi, clade_lorisiformes)
```

The Global Molecular Clock Model

The global molecular clock assumes that the rate of substitution is constant over the tree and over time (Fig. 10.1). Since we calibrated the tree with an informative distribution on the root age we will estimate the clock rate. Here we use a uniform distribution on the logarithm of the clock rate, which signifies our uncertainty of the magnitude of the clock rate. We will say that every magnitude or clock rates between 10^{-6} and 10^1 are equally probable a priori.

```
logClockRate ~ dnUniform(-6,1)
clockRate := 10^logClockRate

moves[++mvi] = mvSlide(logClockRate)
```

Putting it All Together

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

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

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

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

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

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

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

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

Performing an MCMC Analysis Under the Global Clock Model

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

Specifying Monitors

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

```
monitors[++mni] = mnModel(filename="output/primates_cytb_root_calibration.log",
    printgen=10, separator = TAB)
```

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

```
monitors[++mni] = mnFile(filename="output/primates_cytb_root_calibration.trees",
    printgen=10, separator = TAB, psi)
```

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

```
monitors[++mni] = mnScreen(printgen=1000, clockRate, root_time, age_simiiformes)
```

Initializing and Running the MCMC Simulation

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

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

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

```
mymcmc.burnin(generations=10000,tuningInterval=250)
```

Now, run the MCMC:

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

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

- Look at the file called `output/primates_cytb_root_calibration.log` in Tracer.

Exercise 1

We are interested in the divergence time estimate between *Simiiformes*, *Platyrrhini*, *Catarrhini*, *Lorisiformes* and all primates.

To obtain an estimate of the divergence time we read in the tree trace and build the annotated maximum *a posteriori* tree.

```
treetrace = readTreeTrace("output/primates_cytb_root_calibration.trees",
                           treetype="clock")
mapTree(treetrace,"output/primates_cytb_root_calibration.tree")
```

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

- Look at the file called `output/primates_cytb_root_calibration.tree` in FigTree.

Table 9.3: Estimated divergence times in our primates example*.

Clock Model	Primates		Simiiformes		Platyrrhini		Catarrhini		Lorisiformes	
	Mean Estimate	Credible interval								
9.3 Root Calibration										
9.4 Node Calibration										
9.5 Hard Bounds										
9.6 Soft Bounds										

*you can edit this table

Informative node calibration

In the previous section we calibrated the phylogeny using an informative prior on the root age only. This part of the exercise will involve specifying a informative prior on internal nodes in our tree: (1) *Semiiformes*, the split between *New World Monkeys* and *OldWorld Monkeys*; (2) *Platyrrhini*; (3) *Catarrhini*; and (4) *Lorisiformes*, the split between *galagids* and *lorisids*.

In **RevBayes**, calibrated internal nodes are treated differently than in many other programs for estimating species divergence times (*e.g.*, BEAST). This is because the graphical model structure used in **RevBayes** does not allow a stochastic node to be assigned more than one prior distribution. By contrast, the common approach to applying calibration densities as used in other dating softwares leads to incoherence in the calibration prior (for detailed explanations 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 ?? 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. ??).

Adding node calibrations

Additional to the model that we used in the previous exercise we will add calibrations to some clades (see Table 9.1).

First, we specify the calibration for the catarrhini split using a normal distribution with mean 29 and standard deviation of 6.

```
obs_age_catarrhini ~ dnNormal(age_catarrhini,6)
obs_age_catarrhini.clamp(29)
```

Next, we specify the calibration for the Platyrrhini split using a normal distribution with mean 23.5 and standard deviation of 3.

```
obs_age_platyrrhini ~ dnNormal(age_platyrrhini,3)
obs_age_platyrrhini.clamp(23.5)
```

Then, we specify the calibration for the split between *New World Monkeys* and *OldWorld Monkeys* using a normal distribution with mean 43 and standard deviation of 4.5.

```
obs_age_simiiformes ~ dnNormal(age_simiiformes,4.5)
obs_age_simiiformes.clamp(43)
```

Finally, we specify the calibration for the lorisiformes (the split between *galagids* and *lorisids*) using a normal distribution with mean 40 and standard deviation of 3.

```
obs_age_lorisiformes ~ dnNormal(age_lorisiformes,3)
obs_age_lorisiformes.clamp(40)
```

Exercise 2

- Copy the file `RootCalibration.Rev` and call it `NodeCalibration`.
- Add the node calibrations outlines above.
- Rename the output files (*i.e.*, the filenames for the monitors).
- Run the MCMC analysis.
- Look at the output in `Tracer`.
- Fill in the table.

Hard-bounded node calibrations

Additional to the model that we used in the previous exercise we will add calibrations to some clades (see Table 9.2).

First, we specify the calibration for the catarrhini split using a uniform distribution with minimum of 20.55 and maximum of 37.3.

```
min_age_catarrhini <- 20.55
max_age_catarrhini <- 37.3
width_age_prior_catarrhini <- (max_age_catarrhini-min_age_catarrhini)/2.0
mean_age_prior_catarrhini <- min_age_catarrhini + width_age_prior_catarrhini
obs_age_catarrhini ~ dnUniform(age_catarrhini - width_age_prior_catarrhini,
                                age_catarrhini + width_age_prior_catarrhini)
obs_age_catarrhini.clamp( mean_age_prior_catarrhini )
```

Next, we specify the calibration for the Platyrrhini split using a uniform distribution with minimum of 11.8 and maximum of 37.3.

```
min_age_platyrrhini <- 11.8
max_age_platyrrhini <- 37.3
width_age_prior_platyrrhini <- (max_age_platyrrhini-min_age_platyrrhini)/2.0
mean_age_prior_platyrrhini <- min_age_platyrrhini + width_age_prior_platyrrhini
obs_age_platyrrhini ~ dnUniform(age_platyrrhini - width_age_prior_platyrrhini,
                                 age_platyrrhini + width_age_prior_platyrrhini)
obs_age_platyrrhini.clamp( mean_age_prior_platyrrhini )
```

Then, we specify the calibration for the split between *New World Monkeys* and *OldWorld Monkeys* using a uniform distribution with minimum of 28.3 and maximum of 56.

```
min_age_simiiformes <- 28.3
max_age_simiiformes <- 56
width_age_prior_simiiformes <- (max_age_simiiformes-min_age_simiiformes)/2.0
mean_age_prior_simiiformes <- min_age_simiiformes + width_age_prior_simiiformes
obs_age_simiiformes ~ dnUniform(age_simiiformes - width_age_prior_simiiformes,
                                 age_simiiformes + width_age_prior_simiiformes)
obs_age_simiiformes.clamp( mean_age_prior_simiiformes )
```

Finally, we specify the calibration for the lorisiformes (the split between *galagids* and *lorisids*) using a uniform distribution with minimum of 37.1 and maximum of 56.

```
min_age_lorisiformes <- 37.1
max_age_lorisiformes <- 56
width_age_prior_lorisiformes <- (max_age_lorisiformes-min_age_lorisiformes)/2.0
mean_age_prior_lorisiformes <- min_age_lorisiformes + width_age_prior_lorisiformes
obs_age_lorisiformes ~ dnUniform(age_lorisiformes - width_age_prior_lorisiformes,
                                 age_lorisiformes + width_age_prior_lorisiformes)
obs_age_lorisiformes.clamp( mean_age_prior_lorisiformes )
```

Exercise 3

- Copy the file `RootCalibration.Rev` and call it `HardBoundsNodeCalibration.Rev`.
- Add the node calibrations outlines above.
- Rename the output files (*i.e.*, the filenames for the monitors).
- Run the MCMC analysis.
- Look at the output in `Tracer`.
- Fill in the table.

Soft-bounded node calibrations

Additional to the model that we used in the previous exercise we will add calibrations to some clades (see Table 9.2).

First, we specify the calibration for the catarrhini split using a uniform distribution with minimum of 20.55 and maximum of 37.3.

```
min_age_catarrhini <- 20.55
max_age_catarrhini <- 37.3
width_age_prior_catarrhini <- (max_age_catarrhini-min_age_catarrhini)/2.0
mean_age_prior_catarrhini <- min_age_catarrhini + width_age_prior_catarrhini
obs_age_catarrhini ~ dnSoftBoundUniformNormal(min=age_catarrhini -
    width_age_prior_catarrhini, max=age_catarrhini + width_age_prior_catarrhini, sd
    =2.5, p=0.95)
obs_age_catarrhini.clamp( mean_age_prior_catarrhini )
```

Next, we specify the calibration for the Platyrhini split using a uniform distribution with minimum of 11.8 and maximum of 37.3.

```
min_age_platyrhini <- 11.8
max_age_platyrhini <- 37.3
width_age_prior_platyrhini <- (max_age_platyrhini-min_age_platyrhini)/2.0
mean_age_prior_platyrhini <- min_age_platyrhini + width_age_prior_platyrhini
obs_age_platyrhini ~ dnSoftBoundUniformNormal(min=age_platyrhini -
    width_age_prior_platyrhini, max=age_platyrhini + width_age_prior_platyrhini, sd
    =2.5, p=0.95)
obs_age_platyrhini.clamp( mean_age_prior_platyrhini )
```

Then, we specify the calibration for the split between *New World Monkeys* and *OldWorld Monkeys* using a uniform distribution with minimum of 28.3 and maximum of 56.

```
min_age_simiiformes <- 28.3
max_age_simiiformes <- 56
width_age_prior_simiiformes <- (max_age_simiiformes-min_age_simiiformes)/2.0
mean_age_prior_simiiformes <- min_age_simiiformes + width_age_prior_simiiformes
obs_age_simiiformes ~ dnSoftBoundUniformNormal(min=age_simiiformes -
    width_age_prior_simiiformes, max=age_simiiformes + width_age_prior_simiiformes, sd
    =2.5, p=0.95)
width_age_prior_simiiformes)
obs_age_simiiformes.clamp( mean_age_prior_simiiformes )
```

Finally, we specify the calibration for the lorisiformes (the split between *galagids* and *lorisids*) using a uniform distribution with minimum of 37.1 and maximum of 56.

```

min_age_lorisiformes <- 37.1
max_age_lorisiformes <- 56
width_age_prior_lorisiformes <- (max_age_lorisiformes-min_age_lorisiformes)/2.0
mean_age_prior_lorisiformes <- min_age_lorisiformes + width_age_prior_lorisiformes
obs_age_lorisiformes ~ dnSoftBoundUniformNormal(min=age_lorisiformes -
    width_age_prior_lorisiformes, max=age_lorisiformes + width_age_prior_lorisiformes,
    sd=2.5, p=0.95)
width_age_prior_lorisiformes)
obs_age_lorisiformes.clamp( mean_age_prior_lorisiformes )

```

Exercise 4

- Copy the file `RootCalibration.Rev` and call it `SoftBoundsNodeCalibration.Rev`.
- Add the node calibrations outlines above.
- Rename the output files (*i.e.*, the filenames for the monitors).
- Run the MCMC analysis.
- Look at the output in `Tracer`.
- Fill in the table.

Bibliography

- Franzen, J. L., P. D. Gingerich, J. Habersetzer, J. H. Hurum, W. von Koenigswald, and B. H. Smith. 2009. Complete primate skeleton from the middle Eocene of Messel in Germany: morphology and paleobiology. *PLoS One* 4:e5723.
- Heath, T. A., J. P. Huelsenbeck, and T. Stadler. 2014. The fossilized birth-death process for coherent calibration of divergence-time estimates. *Proceedings of the National Academy of Sciences* 111:E2957–E2966.
- Heled, J. and A. J. Drummond. 2012. Calibrated tree priors for relaxed phylogenetics and divergence time estimation. *Systematic Biology* 61:138–149.
- Hodgson, J. A., K. N. Sterner, L. J. Matthews, A. S. Burrell, R. A. Jani, R. L. Raaum, C.-B. Stewart, and T. R. Disotell. 2009. Successive radiations, not stasis, in the South American primate fauna. *Proceedings of the National Academy of Sciences* 106:5534–5539.
- Höhna, S. 2014. Likelihood Inference of Non-Constant Diversification Rates with Incomplete Taxon Sampling. *PLoS One* 9:e84184.
- Höhna, S., M. Defoin-Platel, and A. Drummond. 2008. Clock-constrained tree proposal operators in Bayesian phylogenetic inference. Pages 1–7 in 8th IEEE International Conference on BioInformatics and BioEngineering, 2008. BIBE 2008.
- Höhna, S. and A. J. Drummond. 2012. Guided tree topology proposals for Bayesian phylogenetic inference. *Systematic Biology* 61:1–11.

- Höhna, S., T. Stadler, F. Ronquist, and T. Britton. 2011. Inferring speciation and extinction rates under different species sampling schemes. *Molecular Biology and Evolution* 28:2577–2589.
- Joyce, W. G., J. F. Parham, T. R. Lyson, R. C. Warnock, and P. C. Donoghue. 2013. A divergence dating analysis of turtles using fossil calibrations: an example of best practices. *Journal of Paleontology* 87:612–634.
- Kay, R. F., J. Fleagle, T. Mitchell, M. Colbert, T. Bown, and D. W. Powers. 2008. The anatomy of *Dolichocebus gaimanensis*, a stem platyrhine monkey from Argentina. *Journal of Human Evolution* 54:323–382.
- Lakner, C., P. van der Mark, J. P. Huelsenbeck, B. Larget, and F. Ronquist. 2008. Efficiency of Markov Chain Monte Carlo Tree Proposals in Bayesian Phylogenetics. *Systematic Biology* 57:86–103.
- Matsui, A., F. Rakotondraparany, I. Munechika, M. Hasegawa, and S. Horai. 2009. Molecular phylogeny and evolution of prosimians based on complete sequences of mitochondrial DNAs. *Gene* 441:53–66.
- Parham, J. F., P. C. Donoghue, C. J. Bell, T. D. Calway, J. J. Head, P. A. Holroyd, J. G. Inoue, R. B. Irmis, W. G. Joyce, D. T. Ksepka, et al. 2012. Best practices for justifying fossil calibrations. *Systematic Biology* 61:346–359.
- Perelman, P., W. E. Johnson, C. Roos, H. N. Seuánez, J. E. Horvath, M. A. Moreira, B. Kessing, J. Pontius, M. Roelke, Y. Rumpler, et al. 2011. A molecular phylogeny of living primates. *PLoS Genetics* 7:e1001342.
- Poux, C. and E. J. Douzery. 2004. Primate phylogeny, evolutionary rate variations, and divergence times: a contribution from the nuclear gene IRBP. *American Journal of Physical Anthropology* 124:01–16.
- Seiffert, E. R., E. L. Simons, and Y. Attia. 2003. Fossil evidence for an ancient divergence of lorises and galagos. *Nature* 422:421–424.
- Springer, M. S., R. W. Meredith, J. Gatesy, C. A. Emerling, J. Park, D. L. Rabosky, T. Stadler, C. Steiner, O. A. Ryder, J. E. Janečka, et al. 2012. Macroevolutionary dynamics and historical biogeography of primate diversification inferred from a species supermatrix. *PLoS One* 7:e49521.
- Steiper, M. E., N. M. Young, and T. Y. Sukarna. 2004. Genomic data support the hominoid slowdown and an Early Oligocene estimate for the hominoid–cercopithecoid divergence. *Proceedings of the National Academy of Sciences* 101:17021–17026.
- Tavaré, S. 1986. Some probabilistic and statistical problems in the analysis of DNA sequences. *Some Mathematical Questions in Biology: DNA Sequence Analysis* 17:57–86.
- Tavaré, S., C. R. Marshall, O. Will, C. Soligo, and R. D. Martin. 2002. Using the fossil record to estimate the age of the last common ancestor of extant primates. *Nature* 416:726–729.
- Warnock, R. C., J. F. Parham, W. G. Joyce, T. R. Lyson, and P. C. Donoghue. 2015. Calibration uncertainty in molecular dating analyses: there is no substitute for the prior evaluation of time priors. *Proceedings of the Royal Society B: Biological Sciences* 282:20141013.
- Warnock, R. C., Z. Yang, and P. C. Donoghue. 2012. Exploring uncertainty in the calibration of the molecular clock. *Biology letters* 8:156–159.
- Yang, Z. 1994. Maximum likelihood phylogenetic estimation from DNA sequences with variable rates over sites: Approximate methods. *Journal of Molecular Evolution* 39:306–314.

Yang, Z. and B. Rannala. 2006. Bayesian Estimation of Species Divergence Times Under a Molecular Clock Using Multiple Fossil Calibrations with Soft Bounds. *Molecular Biology and Evolution* 23:212–226.

Chapter 10

Relaxed Clocks

Overview

This tutorial extends on the node and fossil calibration tutorial and focuses on relaxed clock methods. In the first part we will use an uncorrelated lognormal (UCLN) relaxed clock model. In the second analysis you a random local clock to identify potential rate shifts. All the assumptions will be covered more in detail later in this tutorial.

Requirements

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

- RB_Getting_Started
- RB_Basics_Tutorial
- RB_CTMC_Tutorial
- RB_DivergenceTime_Tutorial

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

Data and files

We provide the data file of DNA sequences required for this tutorial. You may want to use your own data instead.

→ Create a folder called `data` and download the following files:

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

Additionally, we will use the same fossil calibration as in the previous tutorial.

Divergence time estimation using uncorrelated, lognormal distributed branch rates

Getting Started

The first section of this exercise involves: (1) setting up a general time reversible (GTR) substitution model (Tavaré 1986) with gamma distributed rate variation among sites (Yang 1994) for an alignment of the cytochrome b subunit; (2) use node calibrations to date the phylogeny; (3) use a relaxed clock model; (4) approximating the posterior probability of the tree topology and node ages (and all other parameters) using MCMC, and; (5) summarizing the MCMC output by computing the maximum *a posteriori* tree. This analysis is mostly equivalent to the analysis performed in the RB_CTMC_Tutorial.

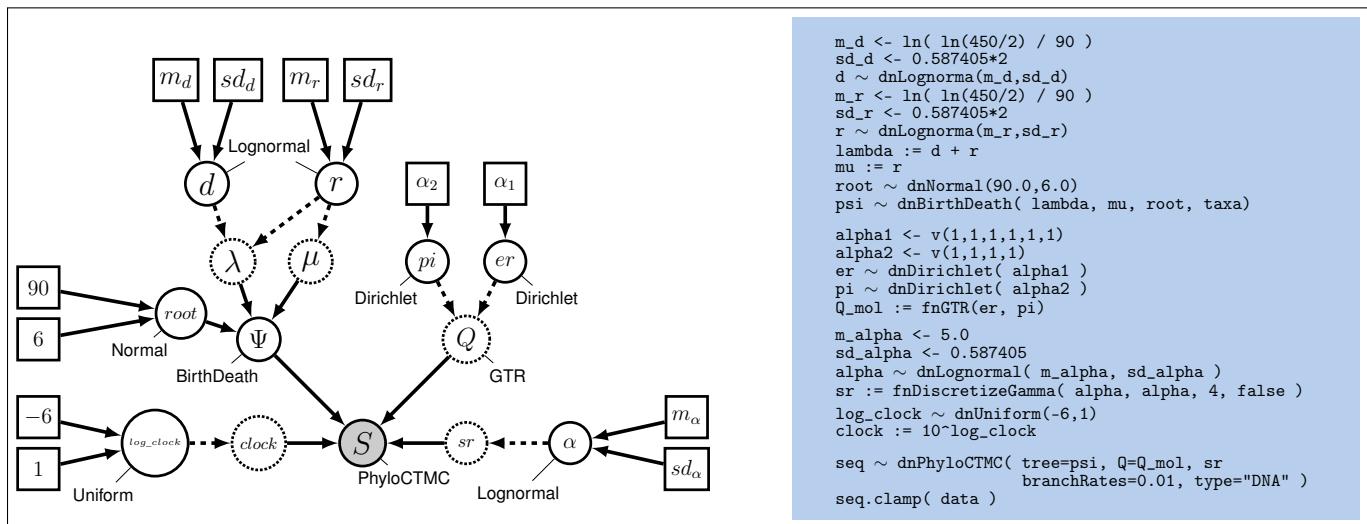


Figure 10.1: An example phylogenetic model depicted in graphical-model notation (left) and the corresponding specification in the Rev language (right). This example shows the basic outline of the root calibration that we will use in the first exercise.

The general structure of the model is represented in Figure 10.1. This figure shows the full model graph.

Loading the Data

- You should have already downloaded the data files in Section 10.2. Links to additional files, including the scripts to run these analyses can be found on the [RevBayes tutorials website](#). Remember that the data file should be in a directory called **data** that is in your current working directory.

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

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

Executing these lines initializes the data matrix as the respective Rev variable.

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

```
n_species <- data.nntaxa()
taxa <- data.taxon()
```

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

```
mvi = 0
```

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

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

General Time Reversible (GTR) Substitution Model

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

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

This node defines the concentration-parameter values of the Dirichlet prior distribution on the exchangeability rates. Now, we can create a stochastic node for the exchangeability rates using the **dnDirichlet()** function, which takes the vector of concentration-parameter values as an argument and the `~` operator. Together, these create a stochastic node named **er**, see Figure 10.1:

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

The Dirichlet prior on our parameter **er** creates a *simplex* of values that sum to 1.

For each stochastic node in our model, we must also specify a proposal mechanism if we wish to estimate that parameter.

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

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

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

The node **pi** represents the π node in Figure 10.1. Now add the simplex scale move on the stationary frequencies to the moves vector:

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

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

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

Setting up the Gamma Model

Create a constant node called **alpha_prior_mean** and a second constant node called **alpha_prior_sd** for the lognormal prior on the gamma-shape parameter (this is represented as the constant rate parameter in Figure 10.1):

```
alpha_prior_mean <- 5.0
alpha_prior_sd <- 0.587405
```

Then create a stochastic node called **alpha** with an lognormal prior (this represents the stochastic node for the α -shape parameter in Figure 10.1):

```
alpha ~ dnLognormal( alpha_prior_mean, alpha_prior_sd )
```

The way the ASRV model is implemented involves discretizing the mean-one gamma distribution into a set number of rate categories, k . To specify this, we need a deterministic node that is a vector that will hold the set of k rates drawn from the gamma distribution with k rate categories. The **fnDiscretizeGamma()** function returns this deterministic node and takes three arguments: the shape and rate of the gamma

distribution and the number of categories. Since we want to discretize a mean-one gamma distribution, we can pass in `alpha` for both the shape and rate.

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

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

The random variable that controls the rate variation is the stochastic node `alpha`. We will apply a simple scale move to this parameter.

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

For more information on ASRV please read the [RB_CTMC_Tutorial](#).

Tree Prior: Tree Topology and Node Ages

The tree (the topology and node ages) is a stochastic node in our phylogenetic model. In Figure 10.1, the tree is denoted Ψ . We will assume a constant-rate birth-death process as the prior distribution on the tree. The distribution in RevBayes is `dnBDP()`. For more information on tree priors please read the [RB_DiversificationRate_Tutorial](#).

For the birth-death process we need a speciation rate and extinction rate parameter. Instead of prior distributions on these parameters directly, we will specify lognormal prior distributions on the diversification and turnover rates.

```
diversification_mean <- ln( ln(n_species/2.0) / 90 )
diversification_sd <- 0.587405*2
diversification ~ dnLognormal(mean=diversification_mean, sd=diversification_sd)
moves[++mvi] = mvScale(diversification, lambda=1.0, tune=true, weight=3.0)

turnover_mean <- ln( ln(n_species/2.0) / 90 )
turnover_sd <- 0.587405*2
turnover ~ dnLognormal(mean=turnover_mean, sd=turnover_sd)
moves[++mvi] = mvScale(turnover, lambda=1.0, tune=true, weight=3.0)

### Transform the parameters
birth_rate := diversification + turnover
death_rate := turnover
```

In our reference publication, [Perelman et al. \(2011\)](#) used a normal distribution with mean of 90.0 MYA with stdev = 6.0 as the prior distribution on the root age. The normal distribution itself is defined on the complete real line (*i.e.*, from $-\infty$ to $+\infty$), however, we know that the root age of primates is definitely larger than 0 (it happen before the present) and smaller than, say, 5000 MYA. Thus, we truncate the

normal distribution. This also has the advantage that the type of the variable for the root age is a positive real number, instead of a real number.

```
root_time ~ dnNormal(mean=90.0,sd=6.0,min=0.0,max=5000.0)
moves[++mvi] = mvScale(root_time,weight=2.0)
```

Additionally, we know that we do not have all primate species included in this data set. We only have 23 out of the approximately 450 primate species. Thus, we use a sampling fraction to represent this incomplete taxon sampling (Höhna et al. 2011; Höhna 2014).

```
rho <- n_species/450
```

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

```
psi ~ dnBDP(lambda=birth_rate, mu=death_rate, rho=rho, rootAge=root_time,
            samplingStrategy="uniform", condition="survival", taxa=taxa)
```

Some types of stochastic nodes can be updated by a number of alternative moves. Different moves may explore parameter space in different ways, and it is possible to use multiple different moves for a given parameter to improve mixing (the efficiency of the MCMC simulation). In the case of our rooted tree, for example, we can use both a nearest-neighbor interchange move without and with changing the node ages (**mvNarrow** and **mvNNI**) and a fixed-nodeheight subtree-prune and regrafting move (**mvFNPR**). For overviews about moves on tree see Lakner et al. (2008), Höhna et al. (2008) and Höhna and Drummond (2012). We also need moves that change the ages of the internal nodes; which are for example the **mvSubtreeScale** and **mvNodeTimeSlideUniform**. These moves do not have tuning parameters associated with them, thus you only need to pass in the **psi** node and proposal **weight**.

```
moves[++mvi] = mvNarrow(psi, weight=5.0)
moves[++mvi] = mvNNI(psi, weight=1.0)
moves[++mvi] = mvFNPR(psi, weight=5.0)
moves[++mvi] = mvFNPR(psi, weight=2.0)
moves[++mvi] = mvSubtreeScale(psi, weight=3.0)
moves[++mvi] = mvNodeTimeSlideUniform(psi, weight=15.0)
```

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

Calibrating node ages

In this exercise we use soft-bounded node calibrations (Yang and Rannala 2006). You can also apply different node calibrations if you want.

We start with a deterministic node monitoring the age of the *Catarrhini*. This involves create a clade object. A clade object simply contains a number of species names. Note, the names need to match **exactly**. Then, we use the **tmrca** function which will record the time of the most recent common ancestor of this clade. Once we have this variable, we can use it for our calibration

```
clade_catarrhini = clade("Pan_paniscus", "Macaca_mulatta")
age_catarrhini := tmrca(psi, clade_catarrhini)
min_age_catarrhini <- 20.55
max_age_catarrhini <- 37.3
width_age_prior_catarrhini <- (max_age_catarrhini-min_age_catarrhini)/2.0
mean_age_prior_catarrhini <- min_age_catarrhini + width_age_prior_catarrhini
obs_age_catarrhini ~ dnSoftBoundUniformNormal(min=age_catarrhini -
    width_age_prior_catarrhini, max=age_catarrhini + width_age_prior_catarrhini, sd
    =2.5, p=0.95)
obs_age_catarrhini.clamp( mean_age_prior_catarrhini )
```

Next, a calibration for the *Platyrrhini*:

```
clade_platyrrhini = clade("Alouatta_palliata", "Callicebus_donacophilus")
age_platyrrhini := tmrca(psi, clade_platyrrhini)
min_age_platyrrhini <- 11.8
max_age_platyrrhini <- 37.3
width_age_prior_platyrrhini <- (max_age_platyrrhini-min_age_platyrrhini)/2.0
mean_age_prior_platyrrhini <- min_age_platyrrhini + width_age_prior_platyrrhini
obs_age_platyrrhini ~ dnSoftBoundUniformNormal(min=age_platyrrhini -
    width_age_prior_platyrrhini, max=age_platyrrhini + width_age_prior_platyrrhini, sd
    =2.5, p=0.95)
obs_age_platyrrhini.clamp( mean_age_prior_platyrrhini )
```

Then, a calibration for the *Simiiformes*:

```
clade_simiiformes = clade("Cebus_albifrons", "Macaca_mulatta")
age_simiiformes := tmrca(psi, clade_simiiformes)
min_age_simiiformes <- 28.3
max_age_simiiformes <- 56
width_age_prior_simiiformes <- (max_age_simiiformes-min_age_simiiformes)/2.0
mean_age_prior_simiiformes <- min_age_simiiformes + width_age_prior_simiiformes
obs_age_simiiformes ~ dnSoftBoundUniformNormal(min=age_simiiformes -
    width_age_prior_simiiformes, max=age_simiiformes + width_age_prior_simiiformes, sd
    =2.5, p=0.95)
width_age_prior_simiiformes)
obs_age_simiiformes.clamp( mean_age_prior_simiiformes )
```

And finally, a calibration for the *Lorisiformes*:

```
clade_lorisiformes = clade("Loris_tardigradus", "Galago_senegalensis")
age_lorisiformes := tmrca(psi, clade_lorisiformes)
min_age_lorisiformes <- 37.1
max_age_lorisiformes <- 56
width_age_prior_lorisiformes <- (max_age_lorisiformes-min_age_lorisiformes)/2.0
mean_age_prior_lorisiformes <- min_age_lorisiformes + width_age_prior_lorisiformes
obs_age_lorisiformes ~ dnSoftBoundUniformNormal(min=age_lorisiformes -
    width_age_prior_lorisiformes, max=age_lorisiformes + width_age_prior_lorisiformes,
    sd=2.5, p=0.95)
width_age_prior_lorisiformes
obs_age_lorisiformes.clamp( mean_age_prior_lorisiformes )
```

The Uncorrelated Lognormal Relaxed Clock Model

In theory each branch has its own clock rate drawn independently and identically (iid) from a lognormal distribution. However, independent draws from this continuous distribution are problematic because the prior density would converge to ∞ when all rates are equal and the variance is equal to zero. Instead, a discretized version of the chosen prior distribution behaves better and is computationally more efficient. We will thus specify (more or less arbitrarily) the number of rate categories to be 10.

```
NUM_RATE_CATEGORIES = 10
```

You can experiment with the number of rate categories if you want.

The lognormal distribution is parameterized by the log-transformed mean and the standard deviation. As usual, we will say that every magnitude or clock rates between 10^{-6} and 10^1 are equally probably a priori. Additionally, we use an exponential prior distribution on the standard deviation because we want to give the highest prior probability to small values which represent our preference to a more clock-like behavior.

```
ucln_mean ~ dnUnif(1E-6, 10)
ucln_log_mean := ln( ucln_mean )
ucln_sigma ~ dnExponential(10.0)
```

Since both variables are stochastic parameters of the model, and defined for positive real numbers, we apply scaling moves on them.

```
moves[++mvi] = mvScale(ucln_sigma, lambda=0.5, weight=5.0)
```

In the next step we use the function **fnDiscretizeDistribution** to compute the quantiles deterministically of the lognormal distribution. The **fnDiscretizeDistribution** function can be applied to discretize any distribution (*e.g.*, the exponential or gamma distribution) we

```
rate_categories := fnDiscretizeDistribution(dnLnorm(ucln_log_mean, ucln_sigma),
    NUM_RATE_CATEGORIES)
```

Next, we need a prior probability for each rate category. We will simply say that each rate category is equally probable *a priori*.

```
rate_probs <- simplex(rep(1, NUM_RATE_CATEGORIES))
```

Now, we can draw a random variable for each branch rate by using the `dnMixture` to draw a rate from one of the rate categories. We do this for each branch by using a for loop over all branches.

```
for (j in 1:n_branches) {
    # for each branch pull from one of the rate categories
    branch_rates[j] ~ dnMixture(rate_categories, rate_probs)
    moves[++mvi] = mvMixtureAllocation(branch_rates[j], weight=2.0)
}
```

Finally, we create a deterministic helper variable that holds the mean of all branch rates. We may or may not be interested in this later.

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

Putting it All Together

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

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

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

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

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

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

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

Performing an MCMC Analysis Under the Global Clock Model

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

Specifying Monitors

For our MCMC analysis, we set up a vector of *monitors* to record the states of our Markov chain. First, we will initialize the model monitor using the `mnModel` function. This creates a new monitor variable that will output the states for all model parameters when passed into a MCMC function.

```
monitors[++mni] = mnModel(filename="output/primates_cytb_UCLN.log", printgen=10,
    separator = TAB)
```

The `mnExtNewick` monitor will record the tree and node or branch parameters. These parameters can be summarized automatically in our

```
monitors[++mni] = mnExtNewick(filename="output/primates_cytb_UCLN.trees",
    isNodeParameter=FALSE, printgen=10, separator = TAB, tree=psi, branch_rates)
```

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

```
monitors[++mni] = mnScreen(printgen=1000, mean_rt, root_time, age_simiiformes)
```

Initializing and Running the MCMC Simulation

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

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

We may wish to run the `.burnin()` member function.

```
mymcmc.burnin(generations=10000,tuningInterval=250)
```

Now, run the MCMC:

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

When the analysis is complete, you will have the monitored files in your output directory. Hence, we can read in the tree trace and compile the *maximum a posteriori* (MAP) tree.

```
treetrace = readTreeTrace("output/primates_cytb_UCLN.trees", treetype="clock")
map_tree = mapTree(treetrace,"output/primates_cytb_UCLN.tree")
```

Now, run the MCMC:

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

- Look at the file called `output/primates_cytb_UCLN.log` in Tracer.
- Look at the file called `output/primates_cytb_UCLN.tree` in FigTree.

Exercise 1

Again, we are interested in the divergence time estimate between *Simiiformes*, *Platyrrhini*, *Catarrhini*, *Lorisiformes* and all primates.

- Specify your model using the UCLN relaxed clock as outlined above.
- Run the MCMC analysis.
- Look at the output in Tracer.
- Do the node-ages fall into the calibration intervals?
- Fill in the Table 10.1.
- * Repeat the analysis using an exponential distribution for the branch rates.

Table 10.1: Estimated divergence times in our primates example*.

Clock Model	Primates		Simiiformes		Platyrrhini		Catarrhini		Lorisiformes	
	Mean Estimate	Credible interval								
10.3 UCLN										
9.4 UCE										

*you can edit this table

Random local clock

In the previous example we used uncorrelated lognormal distributed clock rates. Often, one may be interested if there are discrete jumps in the clock rates but otherwise the clock rates remain constant. This can be modeled by a random local clock (Drummond and Suchard 2010).

We start by specifying a prior probability of a rate change on a branch.

```
rho <- 0.01
```

Next, we specify a prior on the root clock rate. Here we use the same prior as if we would assume a global strict clock.

```
log_clock_mean ~ dnUniform(-6,1)
moves[++mvi] = mvSlide(log_clock_mean, weight=2.0)
clock_mean := 10^log_clock_mean
```

The next part is the interesting part of setting up the random local clock. We loop again over all branches to create a branch specific clock rate. Then, we draw a clock rate multiplier for a branch from a **dnReversibleJumpMixture**. This is a reversible jump distribution. Either, the value is constant (*e.g.*, 1.0) or drawn from some distribution (*e.g.*, a gamma distribution) with some probability **rho**. Then we can simply compute the rate for a given branch by the product of the ancestral branch times the rate multiplier. Only for the descendent branches of the root node, which does not have an ancestral branch rate, we take the **clock_mean** instead.

```
for (i in n_branches:1) {
    rateChangeProbability[i] := Probability(1-rho)
    clockRateMultiplier[i] ~ dnReversibleJumpMixture(1, dnGamma(2,2),
        rateChangeProbability[i] )
    if ( psi.isRoot( psi.parent(i) ) ) {
        branch_rates[i] := clock_mean * clockRateMultiplier[i]
    } else {
        branch_rates[i] := branch_rates[psi.parent(i)] * clockRateMultiplier[i]
    }
    clockRateChange[i] := ifelse( clockRateMultiplier[i] == 1, 0, 1 )
    moves[++mvi] = mvRJSwitch(clockRateMultiplier[i], weight=1.0)
    moves[++mvi] = mvScale(clockRateMultiplier[i], lambda=0.1, tune=true, weight=2.0)
}
```

Finally, we create two deterministic helper variable that holds the mean of all branch rates and the number of rate shifts. We may or may not be interested in this later.

```
numRateChanges := sum( clockRateChange )
mean_rt := mean(branch_rates)
```

Bibliography

- Drummond, A. and M. Suchard. 2010. Bayesian random local clocks, or one rate to rule them all. *BMC Biology* 8:114.
- Höhna, S. 2014. Likelihood Inference of Non-Constant Diversification Rates with Incomplete Taxon Sampling. *PLoS One* 9:e84184.
- Höhna, S., M. Defoin-Platel, and A. Drummond. 2008. Clock-constrained tree proposal operators in Bayesian phylogenetic inference. Pages 1–7 in 8th IEEE International Conference on BioInformatics and BioEngineering, 2008. BIBE 2008.
- Höhna, S. and A. J. Drummond. 2012. Guided tree topology proposals for Bayesian phylogenetic inference. *Systematic Biology* 61:1–11.
- Höhna, S., T. Stadler, F. Ronquist, and T. Britton. 2011. Inferring speciation and extinction rates under different species sampling schemes. *Molecular Biology and Evolution* 28:2577–2589.
- Lakner, C., P. van der Mark, J. P. Huelsenbeck, B. Larget, and F. Ronquist. 2008. Efficiency of Markov Chain Monte Carlo Tree Proposals in Bayesian Phylogenetics. *Systematic Biology* 57:86–103.
- Perelman, P., W. E. Johnson, C. Roos, H. N. Seuánez, J. E. Horvath, M. A. Moreira, B. Kessing, J. Pontius, M. Roelke, Y. Rumpler, et al. 2011. A molecular phylogeny of living primates. *PLoS Genetics* 7:e1001342.
- Tavaré, S. 1986. Some probabilistic and statistical problems in the analysis of DNA sequences. Some Mathematical Questions in Biology: DNA Sequence Analysis 17:57–86.
- Yang, Z. 1994. Maximum likelihood phylogenetic estimation from DNA sequences with variable rates over sites: Approximate methods. *Journal of Molecular Evolution* 39:306–314.
- 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.

Part VII

Diversification Rate Estimation

Chapter 11

Overview

Overview: Diversification Rate Estimation

Models of speciation and extinction are fundamental to any phylogenetic analysis of macroevolutionary processes (*e.g.*, divergence time estimation, diversification rate estimation, continuous and discrete trait evolution, and historical biogeography). First, a prior model describing the distribution of speciation events over time is critical to estimating phylogenies with branch lengths proportional to time. Second, stochastic branching models allow for inference of speciation and extinction rates. These inferences allow us to investigate key questions in evolutionary biology.

Diversification-rate parameters may be included as nuisance parameters of other phylogenetic models—*i.e.*, where these diversification-rate parameters are not of direct interest. For example, many methods for estimating species divergence times—such as BEAST (Drummond et al. 2012), MrBayes (Ronquist et al. 2012), and RevBayes (Höhna et al. 2016)—implement ‘relaxed-clock models’ that include a constant-rate birth-death branching process as a prior model on the distribution of tree topologies and node ages. Although the parameters of these ‘tree priors’ are not typically of direct interest, they are nevertheless estimated as part of the joint posterior probability distribution of the relaxed-clock model, and so can be estimated simply by querying the corresponding marginal posterior probability densities. In fact, this may provide more robust estimates of the diversification-rate parameters, as they accommodate uncertainty in the other phylogenetic-model parameters (including the tree topology, divergence-time estimates, and the other relaxed-clock model parameters). More recent work, *e.g.*, Heath et al. (2014), uses macroevolutionary models (the fossilized birth-death process) to calibrate phylogenies and thus to infer dated trees.

In these tutorials we focus on the different types of macroevolutionary models to study diversification processes and thus the diversification-rate parameters themselves. Nevertheless, these macroevolutionary models should be used for other evolutionary questions, when an appropriate prior distribution on the tree and divergence times is needed.

Types of Hypotheses for Estimating Diversification Rates

Many evolutionary phenomena entail differential rates of diversification (speciation – extinction); *e.g.*, adaptive radiation, diversity-dependent diversification, key innovations, and mass extinction. The specific study questions regarding lineage diversification may be classified within three fundamental categories of inference problems. Admittedly, this classification scheme is somewhat arbitrary, but it is nevertheless useful, as it allows users to navigate the ever-increasing number of available phylogenetic methods. Below, we describe each of the fundamental questions regarding diversification rates.

(1) Diversification-rate through time estimation *What is the (constant) rate of diversification in my study group?* The most basic models estimate parameters of the stochastic-branching process (*i.e.*, rates of speciation and extinction, or composite parameters such as net-diversification and relative-extinction rates) under the assumption that rates have remained constant across lineages and through time; *i.e.*, under a constant-rate birth-death stochastic-branching process model (Nee et al. 1994). Extensions to the (basic) constant-rate models include diversification-rate variation through time (Stadler 2011; Höhna 2015). First, we might ask whether there is evidence of an episodic, tree-wide increase in diversification rates (associated with a sudden increase in speciation rate and/or decrease in extinction rate), as might occur during an episode of adaptive radiation. A second question asks whether there is evidence of a continuous/gradual decrease in diversification rates through time (associated with decreasing speciation rates and/or increasing extinction rates), as might occur because of diversity-dependent diversification (*i.e.*, where competitive ecological interactions among the species of a growing tree decrease the opportunities for

speciation and/or increase the probability of extinction, *e.g.*, Höhna (2014)). Third, we can ask whether changes in diversification rates are correlated with environmental factors, such as environmental CO₂ or temperature (Condamine et al. 2013). A final question in this category asks whether our study tree was impacted by a mass-extinction event (where a large fraction of the standing species diversity is suddenly lost, *e.g.*, May et al. (2016)). The common theme of these studies is that the diversification process is tree-wide, that is, all lineages of the study group have the exact same rates at a given time.

(2) Diversification-rate variation across branches estimation *Is there evidence that diversification rates have varied significantly across the branches of my study group?* Models have been developed to detect departures from rate constancy across lineages; these tests are analogous to methods that test for departures from a molecular clock—*i.e.*, to assess whether substitution rates vary significantly across lineages (Alfaro et al. 2009; Rabosky 2014). These models are important for assessing whether a given tree violates the assumptions of rate homogeneity among lineages. Furthermore, these models are important to answer questions such as: *What are the branch-specific diversification rates?;* and *Have there been significant diversification-rate shifts along branches in my study group, and if so, how many shifts, what magnitude of rate-shifts and along which branches?*

(3) Character-dependent diversification-rate estimation *Are diversification rates correlated with some variable in my study group?* Character-dependent diversification-rate models aim to identify overall correlations between diversification rates and organismal features (binary and multi-state discrete morphological traits, continuous morphological traits, geographic range, etc.). For example, one can hypothesize that a binary character, say if an organism is herbivorous/carnivorous or self-compatible/self-incompatible, impact the diversification rates. Then, if the organism is in state 0 (*e.g.*, is herbivorous) it has a lower (or higher) diversification rate than if the organism is in state 1 (*e.g.*, carnivorous) (Maddison et al. 2007).

Models

We begin this section with a general introduction to the stochastic birth-death branching process that underlies inference of diversification rates in RevBayes. This primer will provide some details on the relevant theory of stochastic-branching process models. We appreciate that some readers may want to skip this somewhat technical primer; however, we believe that a better understanding of the relevant theory provides a foundation for performing better inferences. We then discuss a variety of specific birth-death models, but emphasize that these examples represent only a tiny fraction of the possible diversification-rate models that can be specified in RevBayes.

The birth-death branching process

Our approach is based on the *reconstructed evolutionary process* described by Nee et al. (1994); a birth-death process in which only sampled, extant lineages are observed. Let $N(t)$ denote the number of species at time t . Assume the process starts at time t_1 (the ‘crown’ age of the most recent common ancestor of the study group, t_{MRCA}) when there are two species. Thus, the process is initiated with two species, $N(t_1) = 2$. We condition the process on sampling at least one descendant from each of these initial two lineages; otherwise t_1 would not correspond to the t_{MRCA} of our study group. Each lineage evolves independently of all other lineages, giving rise to exactly one new lineage with rate $b(t)$ and losing one existing lineage with rate $d(t)$ (Figure 11.1 and Figure 11.2). Note that although each lineage evolves independently, all lineages share both a common (tree-wide) speciation rate $b(t)$ and a common extinction rate $d(t)$ (Nee et al. 1994; Höhna 2015). Additionally, at certain times, t_M , a mass-extinction event occurs and each species existing at that time has the same probability, ρ , of survival. Finally, all extinct lineages

are pruned and only the reconstructed tree remains (Figure 11.1).

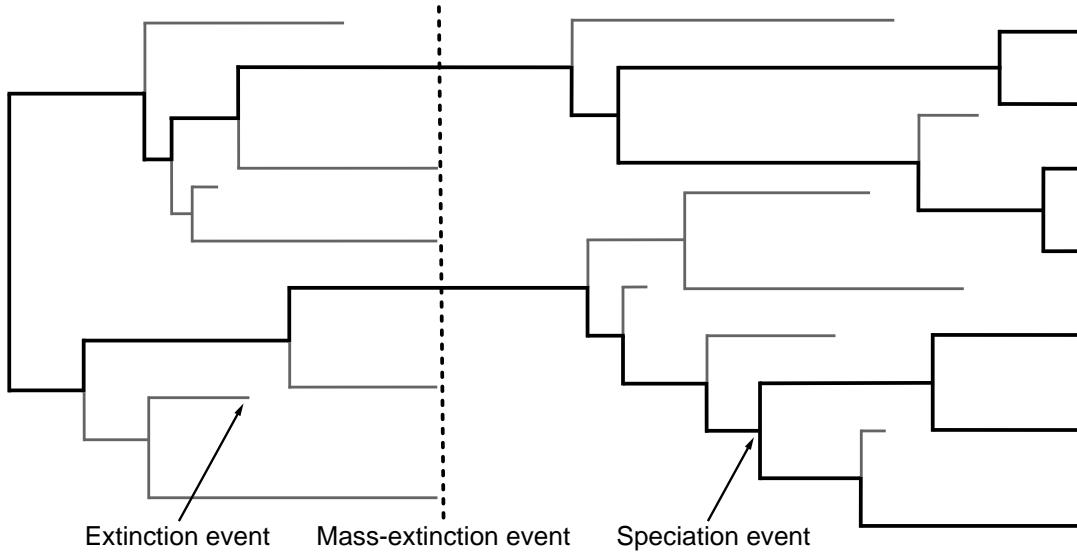


Figure 11.1: A realization of the birth-death process with mass extinction. Lineages that have no extant or sampled descendant are shown in gray and surviving lineages are shown in a thicker black line.

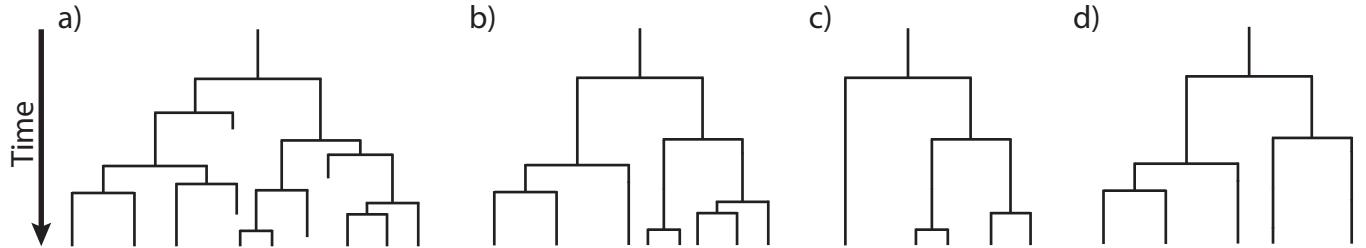


Figure 11.2: **Examples of trees produced under a birth-death process.** The process is initiated at the first speciation event (the ‘crown-age’ of the MRCA) when there are two initial lineages. At each speciation event the ancestral lineage is replaced by two descendant lineages. At an extinction event one lineage simply terminates. (A) A complete tree including extinct lineages. (B) The reconstructed tree of tree from A with extinct lineages pruned away. (C) A *uniform* subsample of the tree from B, where each species was sampled with equal probability, ρ . (D) A *diversified* subsample of the tree from B, where the species were selected so as to maximize diversity.

To condition the probability of observing the branching times on the survival of both lineages that descend from the root, we divide by $P(N(T) > 0 | N(0) = 1)^2$. Then, the probability density of the branching times, \mathbb{T} , becomes

$$P(\mathbb{T}) = \frac{\overbrace{P(N(T) = 1 | N(0) = 1)^2}^{\text{both initial lineages have one descendant}}}{\underbrace{P(N(T) > 0 | N(0) = 1)^2}_{\text{both initial lineages survive}}} \times \prod_{i=2}^{n-1} \overbrace{i \times b(t_i)}^{\text{speciation rate}} \times \overbrace{P(N(T) = 1 | N(t_i) = 1)}^{\text{lineage has one descendant}},$$

and the probability density of the reconstructed tree (topology and branching times) is then

$$\begin{aligned} P(\Psi) &= \frac{2^{n-1}}{n!(n-1)!} \times \left(\frac{P(N(T) = 1 \mid N(0) = 1)}{P(N(T) > 0 \mid N(0) = 1)} \right)^2 \\ &\quad \times \prod_{i=2}^{n-1} i \times b(t_i) \times P(N(T) = 1 \mid N(t_i) = 1) \end{aligned} \quad (11.1)$$

We can expand Equation (11.1) by substituting $P(N(T) > 0 \mid N(t) = 1)^2 \exp(r(t, T))$ for $P(N(T) = 1 \mid N(t) = 1)$, where $r(u, v) = \int_u^v d(t) - b(t) dt$; the above equation becomes

$$\begin{aligned} P(\Psi) &= \frac{2^{n-1}}{n!(n-1)!} \times \left(\frac{P(N(T) > 0 \mid N(0) = 1)^2 \exp(r(0, T))}{P(N(T) > 0 \mid N(0) = 1)} \right)^2 \\ &\quad \times \prod_{i=2}^{n-1} i \times b(t_i) \times P(N(T) > 0 \mid N(t_i) = 1)^2 \exp(r(t_i, T)) \\ &= \frac{2^{n-1}}{n!} \times \left(P(N(T) > 0 \mid N(0) = 1) \exp(r(0, T)) \right)^2 \\ &\quad \times \prod_{i=2}^{n-1} b(t_i) \times P(N(T) > 0 \mid N(t_i) = 1)^2 \exp(r(t_i, T)). \end{aligned} \quad (11.2)$$

For a detailed description of this substitution, see Höhna (2015). Additional information regarding the underlying birth-death process can be found in (Thompson 1975; Equation 3.4.6) and Nee et al. (1994) for constant rates and Höhna (2013; 2014; 2015) for arbitrary rate functions.

To compute the equation above we need to know the rate function, $r(t, s) = \int_t^s d(x) - b(x) dx$, and the probability of survival, $P(N(T) > 0 \mid N(t) = 1)$. Yule (1925) and later Kendall (1948) derived the probability that a process survives ($N(T) > 0$) and the probability of obtaining exactly n species at time T ($N(T) = n$) when the process started at time t with one species. Kendall's results were summarized in Equation (3) and Equation (24) in Nee et al. (1994)

$$P(N(T) > 0 \mid N(t) = 1) = \left(1 + \int_t^T \left(\mu(s) \exp(r(t, s)) \right) ds \right)^{-1} \quad (11.3)$$

$$\begin{aligned} P(N(T) = n \mid N(t) = 1) &= (1 - P(N(T) > 0 \mid N(t) = 1) \exp(r(t, T)))^{n-1} \\ &\quad \times P(N(T) > 0 \mid N(t) = 1)^2 \exp(r(t, T)) \end{aligned} \quad (11.4)$$

An overview for different diversification models is given in Höhna (2015).

Sidebar: Phylogenetic trees as observations

The branching processes used here describe probability distributions on phylogenetic trees. This probability distribution can be used to infer diversification rates given an “observed” phylogenetic tree. In reality we never observe a phylogenetic tree itself. Instead, phylogenetic trees themselves are estimated from actual observations, such as DNA sequences. These phylogenetic tree estimates, especially the divergence times, can have considerable uncertainty associated with them. Thus, the correct approach for estimating diversification rates is to include the uncertainty in the phylogeny by, for example, jointly

estimating the phylogeny and diversification rates. For the simplicity of the following tutorials, we take a shortcut and assume that we know the phylogeny without error. For publication quality analysis you should always estimate the diversification rates jointly with the phylogeny and divergence times.

Bibliography

- Alfaro, M., F. Santini, C. Brock, H. Alamillo, A. Dornburg, D. Rabosky, G. Carnevale, and L. Harmon. 2009. Nine exceptional radiations plus high turnover explain species diversity in jawed vertebrates. *Proceedings of the National Academy of Sciences* 106:13410–13414.
- Condamine, F. L., J. Rolland, and H. Morlon. 2013. Macroevolutionary perspectives to environmental change. *Ecology Letters* .
- Drummond, A., M. Suchard, D. Xie, and A. Rambaut. 2012. Bayesian phylogenetics with beauti and the beast 1.7. *Molecular Biology and Evolution* 29:1969–1973.
- Heath, T. A., J. P. Huelsenbeck, and T. Stadler. 2014. The fossilized birth-death process for coherent calibration of divergence-time estimates. *Proceedings of the National Academy of Sciences* 111:E2957–E2966.
- Höhna, S. 2013. Fast simulation of reconstructed phylogenies under global time-dependent birth-death processes. *Bioinformatics* 29:1367–1374.
- Höhna, S. 2014. Likelihood Inference of Non-Constant Diversification Rates with Incomplete Taxon Sampling. *PLoS One* 9:e84184.
- Höhna, S. 2015. The time-dependent reconstructed evolutionary process with a key-role for mass-extinction events. *Journal of Theoretical Biology* 380:321–331.
- Höhna, S., M. J. Landis, T. A. Heath, B. Boussau, N. Lartillot, B. R. Moore, J. P. Huelsenbeck, and F. Ronquist. 2016. RevBayes: Bayesian phylogenetic inference using graphical models and an interactive model-specification language. *Systematic Biology* 65:726–736.
- Kendall, D. G. 1948. On the generalized "birth-and-death" process. *The Annals of Mathematical Statistics* 19:1–15.
- Maddison, W., P. Midford, and S. Otto. 2007. Estimating a binary character's effect on speciation and extinction. *Systematic Biology* 56:701.
- May, M. R., S. Höhna, and B. R. Moore. 2016. A Bayesian Approach for Detecting the Impact of Mass-Extinction Events on Molecular Phylogenies When Rates of Lineage Diversification May Vary. *Methods in Ecology and Evolution* 7:947–959.
- Nee, S., R. M. May, and P. H. Harvey. 1994. The Reconstructed Evolutionary Process. *Philosophical Transactions: Biological Sciences* 344:305–311.
- Rabosky, D. L. 2014. Automatic detection of key innovations, rate shifts, and diversity-dependence on phylogenetic trees. *PLoS One* 9:e89543.

- Ronquist, F., M. Teslenko, P. van der Mark, D. L. Ayres, A. Darling, S. Höhna, B. Larget, L. Liu, M. A. Suchard, and J. P. Huelsenbeck. 2012. Mrbayes 3.2: efficient bayesian phylogenetic inference and model choice across a large model space. *Systematic Biology* 61:539–542.
- Stadler, T. 2011. Mammalian phylogeny reveals recent diversification rate shifts. *Proceedings of the National Academy of Sciences* 108:6187–6192.
- Thompson, E. 1975. Human evolutionary trees. Cambridge University Press Cambridge.
- Yule, G. 1925. A mathematical theory of evolution, based on the conclusions of dr. jc willis, frs. *Philosophical Transactions of the Royal Society of London. Series B, Containing Papers of a Biological Character* 213:21–87.

Chapter 12

Basic Diversification Rate Estimation

Estimating Constant Speciation & Extinction Rates

Outline

This tutorial describes how to specify basic branching-process models in `RevBayes`; two variants of the constant-rate birth-death process (Yule 1925; Kendall 1948; Thompson 1975; Nee et al. 1994; Rannala and Yang 1996; Yang and Rannala 1997; Höhna 2015). The probabilistic graphical model is given for each component of this tutorial. After each model is specified, you will estimate speciation and extinction rates using Markov chain Monte Carlo (MCMC). Finally, you will estimate the marginal likelihood of the model and evaluate the relative support using Bayes factors.

Requirements

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

- `RB_Getting_Started`
- `RB_Basics_Tutorial`
- `RB_BayesFactor_Tutorial`

Note that the `RB_Basics_Tutorial` introduces the basic syntax of `Rev` but does not cover any phylogenetic models. You may skip the `RB_Basics_Tutorial` if you have some familiarity with `R`. The `RB_BayesFactor_Tutorial` introduced Bayesian model selection by means of Bayes factors, which can be skipped by readers familiar with Bayesian model selection. We tried to keep this tutorial very basic and introduce all the language concepts and theory on the way. You may only need the `RB_Basics_Tutorial` for a more in-depth discussion of concepts in `Rev`.

Data and files

We provide the data file(s) which we will use in this tutorial. You may want to use your own data instead. In the `data` folder, you will find the following files

- `primates_tree.nex`: Dated primates phylogeny including 233 species from Magnuson-Ford and Otto (2012).
- Open the tree `data/primates_tree.nex` in FigTree.

Pure-Birth (Yule) Model

Before evaluating the relative support for different models, we must first specify them in `Rev`. In this section, we will walk through specifying a pure-birth process model and estimating the marginal likelihood. The section about the birth-death process will be less detailed because it will build up on this section.

The simplest branching model is the *pure-birth process* described by Yule (1925). 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 1925; Aldous 2001; Höhna 2014). The pure-birth branching model does not allow for lineage extinction (*i.e.*, the extinction rate $\mu = 0$). However, the model depends on a second parameter, ρ , which is the probability of sampling a species in the present time. It also depends on 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. 12.1).

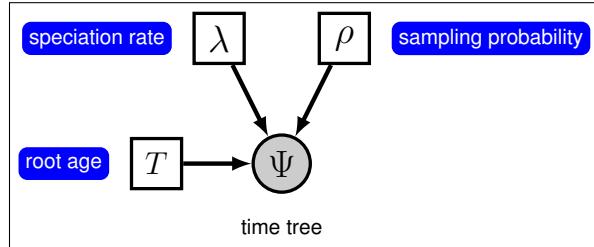


Figure 12.1: The graphical model representation of the pure-birth (Yule) process.

We can add hierarchical structure to this model and account for uncertainty in the value of the speciation rate by placing a hyperprior on λ (Fig. 12.2). The graphical models in Figures 12.1 and 12.2 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.

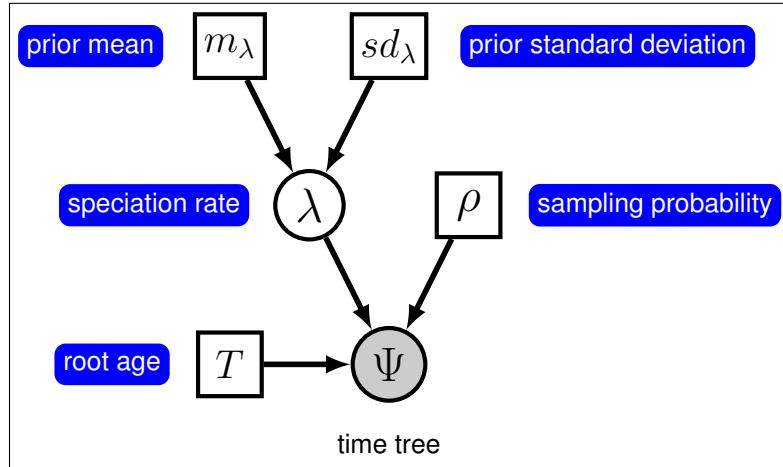


Figure 12.2: The graphical model representation of the pure-birth (Yule) process, where the speciation rate is treated as a random variable drawn from a lognormal distribution.

For this exercise, we will specify a Yule model, such that the speciation rate is a stochastic node, drawn from a lognormal distribution as in Figure 12.2. In a Bayesian framework, we are interested in estimating the posterior probability of λ given that we observe a time tree.

$$\mathbb{P}(\lambda | \Psi) = \frac{\mathbb{P}(\Psi | \lambda)\mathbb{P}(\lambda | \nu)}{\mathbb{P}(\Psi)} \quad (12.1)$$

In this example, we have a phylogeny of 233 primates. 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 [Magnuson-Ford and Otto \(2012\)](#). Furthermore, there are approximately 367 described primates species, so we will fix the parameter ρ to 233/367.

- The full Yule-model specification is in the file called `Yule.Rev` on the RevBayes tutorial repository.

Read the tree

Begin by reading in the observed tree.

```
T <- readTrees("data/primates_tree.nex") [1]
```

From this tree, we can get some helpful variables:

```
taxa <- T.taxa()
```

Additionally, we can initialize an iterator variable for our vector of moves:

```
mvi = 0
mni = 0
```

Specifying the model

Birth rate

The model we are specifying only has three nodes (Fig. 12.2). We can specify the birth rate λ , the mean and standard deviation of the lognormal hyperprior on λ , and the conditional dependency of the two parameters all in one line of Rev code.

```
birth_rate_mean <- ln( ln(367/2) / T.rootAge() )
birth_rate_sd <- 0.587405
birth_rate ~ dnLognormal(mean=birth_rate_mean,sd=birth_rate_sd)
```

Here, the stochastic node called `birth_rate` represents the speciation rate λ . `birth_rate_mean` and `birth_rate_sd` are the prior mean and prior standard deviation, respectively. We chose the prior mean so that it is centered around observed number of species (*i.e.*, the expected number of species under a Yule process will thus be equal to the observed number of species) and a prior standard deviation of 0.587405 which creates a lognormal distribution with 95% prior probability spanning exactly one order of magnitude. If you want to represent more prior uncertainty by, *e.g.*, allowing for two orders of magnitude in the 95% prior probability then you can simply multiply `birth_rate_sd` by a factor of 2.

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[++mvi] = mvScale(birth_rate,lambda=1,tune=true,weight=3)
```

Sampling probability

Our prior belief is that we have sampled 233 out of 367 living primate species. To account for this we can set the sampling parameter as a constant node with a value of 233/367

```
rho <- T.nTips()/367
```

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 [Magnuson-Ford and Otto \(2012\)](#) tree.

```
root_time <- T.rootAge()
```

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="survival", taxa=taxa)
```

If you refer back to Equation 12.1 and Figure 12.2, the time tree Ψ is the variable we observe, *i.e.*, the data. We can set this in Rev by using the `clamp()` function.

```
timetree.clamp(T)
```

Here we are fixing the value of the time tree to our observed tree from [Magnuson-Ford and Otto \(2012\)](#).

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 Rev 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 traverses all of the connections and finds all of the nodes we specified.

Running an MCMC analysis

Specifying Monitors

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

```
monitors[++mni] = mnModel(filename="output/primates_Yule.log", printgen=10, separator = TAB)
```

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

```
monitors[++mni] = mnScreen(printgen=1000, birth_rate)
```

Initializing and Running the MCMC Simulation

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

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

We may wish to run the `.burnin()` member function, *i.e.*, if we wish to pre-run the chain and discard the initial states. Recall that the `.burnin()` function specifies a *completely separate* preliminary MCMC analysis that is used to tune the scale of the moves to improve mixing of the MCMC analysis.

```
mymcmc.burnin(generations=10000, tuningInterval=200)
```

Now, run the MCMC:

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

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

- The Rev file for performing this analysis: [mcmc_Yule.Rev](#).

Exercise 1

- Run an MCMC simulation to estimate the posterior distribution of the speciation rate (`birth_rate`).
- Load the generated output file into Tracer: What is the mean posterior estimate of the `birth_rate` and what is the estimated HPD?
- Compare the prior mean with the posterior mean. (**Hint:** Use the optional argument `underPrior=TRUE` in the function `mymcmc.run()`) Are they different (e.g., Figure 12.3)? Is the posterior mean outside the prior 95% probability interval?
- Repeat the analysis and allow for two orders of magnitude of prior uncertainty.

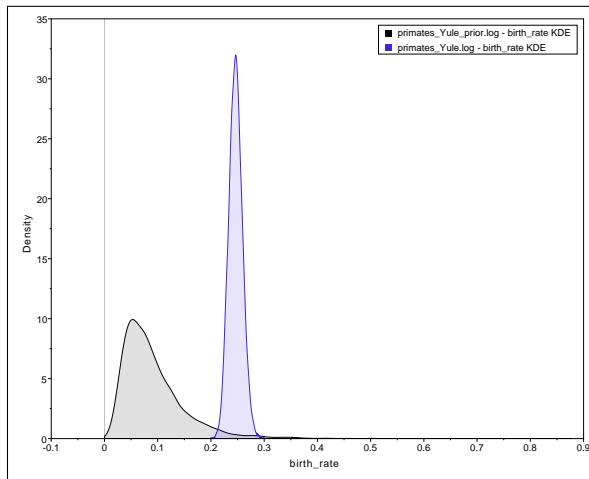


Figure 12.3: Estimates of the posterior and prior distribution of the `birth_rate` visualized in Tracer. The prior (black curve) shows the lognormal distribution that we chose as the prior distribution.

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. For more information on marginal likelihood estimation please read the [RB_BayesFactor_Tutorial](#).

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, monitors, "output/Yule_powp.out", cats=100,
    sampleFreq=10)
```

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

```
pow_p.burnin(generations=10000,tuningInterval=200)
```

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

```
pow_p.run(generations=10000)
```

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_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 12.1.

```
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_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 12.1.

```
ps.marginal()
```

→ The Rev file for performing this analysis: [ml_Yule.Rev](#).

Exercise 2

- Compute the marginal likelihood under the Yule model.
- Enter the estimate in the table below.

Table 12.1: Marginal likelihoods and Bayes factors*.

Estimate	Stepping-stone	Path sampling
Marginal likelihood Yule (M_0)		
Marginal likelihood birth-death (M_1)		
Supported model?		

*you can edit this table

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 et al. 1994](#); [Höhna 2015](#)).

[Yang and Rannala \(1997\)](#) derived the probability of time trees under an extension of the birth-death model that accounts for incomplete sampling of the tips (Fig. 12.4) (see also [Stadler \(2009\)](#) and [Höhna \(2014\)](#)). 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.

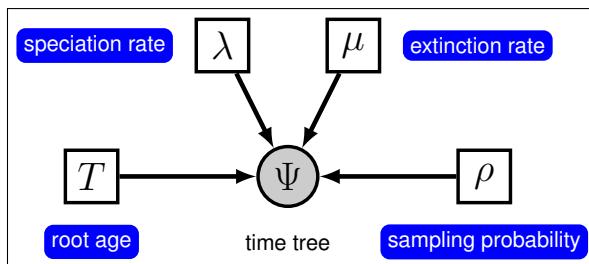


Figure 12.4: The graphical model representation of the birth-death process with uniform sampling and conditioned on the root age.

In principle, we can specify a model with prior distributions on speciation and extinction rates directly. One possibility is to specify an exponential, lognormal, or gamma distribution as the prior on either rate parameter. However, it is more common to specify prior distributions on a transformation of the speciation and extinction rate because, for example, we want to enforce that the speciation rate is always larger than the extinction rate.

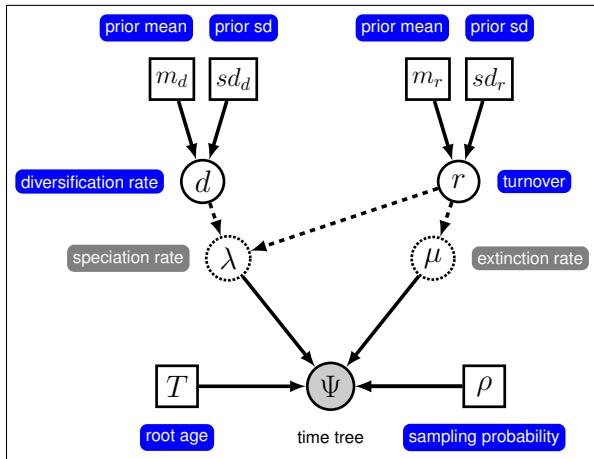


Figure 12.5: The graphical model representation of the birth-death process with uniform sampling parameterized using the diversification and turnover.

In the following subsections we will only provide the key command that are different for the constant-rate birth-death process. All other commands will be the same as in the previous exercise. You should copy the `mcmc_Yule.Rev` script and modify it accordingly. Don't forget to rename the filenames of the monitors to avoid overwriting of your previous results!

Diversification and turnover

We have some good prior information about the magnitude of the diversification. The diversification rate represent the rate at which the species diversity increases. Thus, we just use the same prior for the diversification rate as we used before for the birth rate.

```
diversification_mean <- ln( ln(367.0/2.0) / T.rootAge() )
diversification_sd <- 0.587405
diversification ~ dnLognormal(mean=diversification_mean,sd=diversification_sd)
moves[++mvi] = mvScale(diversification,lambda=1.0,tune=true,weight=3.0)
```

Unfortunately, we have less prior information about the turnover rate. The turnover rate is the rate at which one species is replaced by another species due to a birth plus death event. Hence, the turnover rate represent the longevity of a species. For simplicity we use the same prior on the turnover rate but with two orders of magnitude prior uncertainty.

```
turnover_mean <- ln( ln(367.0/2.0) / T.rootAge() )
turnover_sd <- 0.587405*2
turnover ~ dnLognormal(mean=turnover_mean,sd=turnover_sd)
moves[++mvi] = mvScale(turnover,lambda=1.0,tune=true,weight=3.0)
```

Birth rate and death rate

The birth and death rates are both deterministic nodes. We compute them by simple parameter transformation. Note that the death rate is in fact equal to the turnover rate.

```
birth_rate := diversification + turnover
death_rate := turnover
```

All other parameters, such as the sampling probability and the root age are kept the same as in the analysis above.

The time tree

Initialize the stochastic node representing the time tree. The main difference now is that we provide a stochastic parameter for the extinction rate μ .

```
timetree ~ dnBDP(lambda=birth_rate, mu=death_rate, rho=rho, rootAge=root_time,
    samplingStrategy="uniform", condition="survival", taxa=taxa)
```

Exercise 3

- Run an MCMC simulation to compute the posterior distribution of the diversification and turnover rate.
- Look at the parameter estimates in **Tracer**. What can you say about the diversification, turnover, speciation and extinction rates? How high is the extinction rate compared with the speciation rate?
- Compute the marginal likelihood under the BD model. Which model is supported by the data?
- Enter the estimate in the table above.
- Can you modify the script to use a prior on the birth drawn from a lognormal distribution and relative death rate drawn from a beta distribution so that the extinction rate is equal to the birth rate times the relative death rate?
 - Do the parameter estimates change?
 - What about the marginal likelihood estimates?

Bibliography

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

Höhna, S. 2014. Likelihood Inference of Non-Constant Diversification Rates with Incomplete Taxon Sampling. PLoS One 9:e84184.

- Höhna, S. 2015. The time-dependent reconstructed evolutionary process with a key-role for mass-extinction events. *Journal of Theoretical Biology* 380:321–331.
- Kendall, D. G. 1948. On the generalized "birth-and-death" process. *The Annals of Mathematical Statistics* 19:1–15.
- Magnuson-Ford, K. and S. P. Otto. 2012. Linking the investigations of character evolution and species diversification. *The American Naturalist* 180:225–245.
- Nee, S., R. M. May, and P. H. Harvey. 1994. The Reconstructed Evolutionary Process. *Philosophical Transactions: Biological Sciences* 344:305–311.
- Rannala, B. and Z. Yang. 1996. Probability distribution of molecular evolutionary trees: A new method of phylogenetic inference. *Journal of Molecular Evolution* 43:304–311.
- 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, E. 1975. Human evolutionary trees. Cambridge University Press Cambridge.
- Yang, Z. and B. Rannala. 1997. Bayesian phylogenetic inference using DNA sequences: a Markov Chain Monte Carlo Method. *Molecular Biology and Evolution* 14:717–724.
- Yule, G. 1925. A mathematical theory of evolution, based on the conclusions of dr. jc willis, frs. *Philosophical Transactions of the Royal Society of London. Series B, Containing Papers of a Biological Character* 213:21–87.

Chapter 13

Episodic Diversification Rate Estimation

Estimating Speciation & Extinction Rates Through Time

Outline

This tutorial describes how to specify an episodic branching-process model in RevBayes; a birth-death process where diversification rates vary episodically through time modeled as piecewise-constant rates (Stadler 2011; Höhna 2015). The probabilistic graphical model is given once at the beginning of this tutorial. Your goal is to estimate speciation and extinction rates through-time using Markov chain Monte Carlo (MCMC).

Requirements

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

- [Getting started](#)
- [Rev basics](#)
- [Basic Diversification Rate Estimation](#)

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

Data and files

We provide the data file(s) which we will use in this tutorial. You may want to use your own data instead. In the `data` folder, you will find the following files

- `primates_tree.nex`: Dated primates phylogeny including 233 out of 367 species from [Magnuson-Ford and Otto \(2012\)](#).
- Open the tree `data/primates_tree.nex` in FigTree.

Episodic Birth-Death Model

The basic idea behind the model is that speciation and extinction rates are constant within time intervals but can be different between time intervals. Thus, we will divide time into equally sized intervals, with the only exception that the first 20% of the time does not have any rate changes. Our only reason to do so is because there are too few lineages in the reconstructed tree at that time to obtain reliable parameter estimates. An overview of the underlying theory of the specific model and implementation is given in [Höhna \(2015\)](#). Figure 15.1 shows an example of a constant rate birth-death process and an episodic birth-death process.

We assume that the log-transformed rates are drawn from a normal distribution. Furthermore, we will assume that rates are autocorrelated, that is, rates in the current time interval will be centered around the

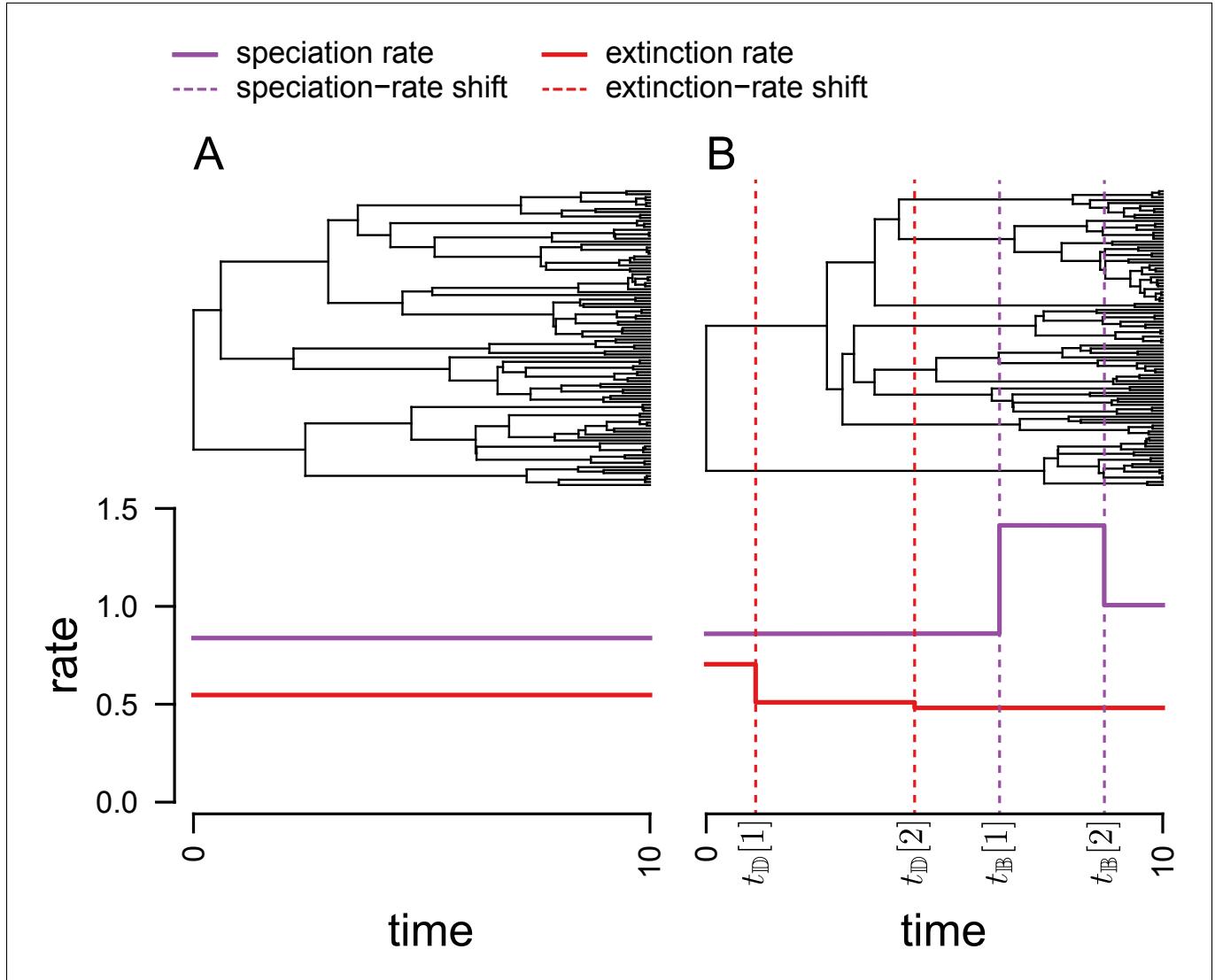


Figure 13.1: Two scenarios of birth-death models. On the left we show constant diversification. On the right we show an example of an episodic birth-death process where rates are constant in each time interval (epoch). The top panel of this figure shows an example realization under the given rates.

rates in the previous time interval. Thus, we model (log-transformed) diversification rates by a Brownian motion. The assumption of autocorrelated rates does not only makes sense biologically but also improves our ability to estimate parameters. We show a graphical model of the episodic birth-death process with autocorrelated rates in Figure 15.2. This graphical model shows you which variables are included in the model, and the dependency between the variables. Thus, it makes the structure and assumption of the model clear and visible instead of a black-box (Höhna et al. 2014). For example, you see how the speciation and extinction rates in each time interval depend on the rates of the previous interval, and that we use a hyperprior for the standard deviation of rates between time intervals.

Read the tree

Begin by reading in the “observed” tree.

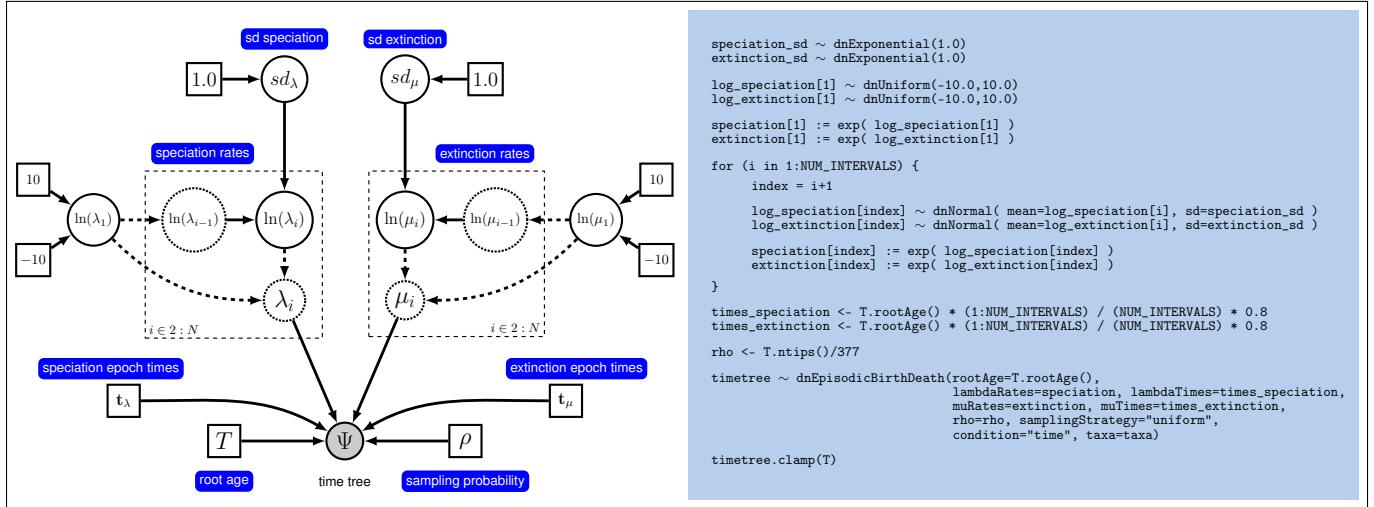


Figure 13.2: A graphical model with the outline of the Rev code. On the left we see the graphical model describing the correlated (Brownian motion) model for rate-variation through time. On the right we show the corresponding Rev commands to instantiate this model. This figure gives a complete overview of the model that we use here in this analysis.

```
T <- readTrees("data/primates_tree.nex") [1]
```

From this tree, we get some helpful variables, such as the taxon information which we need to instantiate the birth-death process.

```
taxa <- T.taxa()
```

Additionally, we initialize an iterator variable for our vector of moves and monitors.

```
mvi = 0
mni = 0
```

Finally, we create a helper variable that specifies the number of intervals.

```
NUM_INTERVALS = 10
```

Using this variable we can easily change our script to break-up time into many (*e.g.*, `NUM_INTERVALS = 100`) or few (*e.g.*, `NUM_INTERVALS = 4`) intervals.

Specifying the model

Priors on amount of rate variation

We start by specifying prior distributions on the rates. Each interval-specific speciation- and extinction-rate will be drawn from a normal distribution. Thus, we need a parameter for the standard deviation of those normal distributions. We use an exponential hyperprior with rate 1.0 to estimate the standard deviation, but assume that all speciation rates and all extinction rates share the same standard deviation. The motivation for an exponential hyperprior is that it has the highest probability density at 0 which would make the variance of rates between consecutive time intervals 0 and thus represent a constant-rate process. The data will tell us if there should be much variation in rates through time. (You may want to experiment with this hyperprior if you are interested.)

```
speciation_sd ~ dnExponential(1.0)
extinction_sd ~ dnExponential(1.0)
```

We apply a simple scaling move on each prior parameter.

```
moves[++mvi] = mvScale(speciation_sd, weight=5.0)
moves[++mvi] = mvScale(extinction_sd, weight=5.0)
```

Specifying episodic rates

As we mentioned before, we will apply normal distributions as priors for each log-transformed rate. We begin with the rate at the present which is our initial rate parameter. The rates at the present will be specified slightly differently because they are not correlated to any previous rates. This is because we are actually modeling rate-changes backwards in time and there is no previous rate for the rate at the present. Modeling rates backwards in time makes it easier for us if we had some prior information about some event affected diversification sometime before the present, *e.g.*, 25 million years ago.

We use a uniform distribution between -10 and 10 because of our lack of prior knowledge on the diversification rate. This actually means that we allow speciation and extinction rates between e^{-10} and e^{10} , so we should clearly cover the true values. (Note that for diversification rate estimates, e^{-10} is virtually 0 since the rate is so slow).

```
log_speciation[1] ~ dnUniform(-10.0,10.0)
log_extinction[1] ~ dnUniform(-10.0,10.0)
```

Notice that we store the diversification rate variables in vectors. Storing the rate parameters in vectors will be useful and important later when we pass the rates into the birth-death process.

We apply simple sliding window moves for the rates. Normally we would use scaling moves but in this case we work on the log-transformed parameters and thus sliding moves perform better. (If you are keen you can test the differences.)

```
moves[++mvi] = mvSlide(log_speciation[1], weight=2)
moves[++mvi] = mvSlide(log_extinction[1], weight=2)
```

Now we transform the diversification rate parameters into actual rates using an exponential parameter transformation.

```
speciation[1] := exp( log_speciation[1] )
extinction[1] := exp( log_extinction[1] )
```

Next, we specify the speciation and extinction rates for each time interval (*i.e.*, epoch). This can be done efficiently using a **for**-loop. We will use a specific index variable so that we can more easily refer to the rate at the previous interval. Remember that we want to model the rates as a Brownian motion, which we achieve by specifying a normal distribution as the prior distribution on the rates centered around the previous rate (*i.e.*, the mean of the normal distribution is equal to the previous rate).

```
for (i in 1:NUM_INTERVALS) {
    index = i+1

    log_speciation[index] ~ dnNormal( mean=log_speciation[i], sd=speciation_sd )
    log_extinction[index] ~ dnNormal( mean=log_extinction[i], sd=extinction_sd )

    moves[++mvi] = mvSlide(log_speciation[index], weight=2)
    moves[++mvi] = mvSlide(log_extinction[index], weight=2)

    speciation[index] := exp( log_speciation[index] )
    extinction[index] := exp( log_extinction[index] )

}
```

Finally, we apply moves that slide all values in the rate vectors, *i.e.*, all speciation or extinction rates. We will use an **mvVectorSlide** move.

```
moves[++mvi] = mvVectorSlide(log_speciation, weight=10)
moves[++mvi] = mvVectorSlide(log_extinction, weight=10)
```

Additionally, we apply a **mvShrinkExpand** move which changes the spread of several variables around their mean.

```
moves[++mvi] = mvShrinkExpand( log_speciation, sd=speciation_sd, weight=10 )
moves[++mvi] = mvShrinkExpand( log_extinction, sd=extinction_sd, weight=10 )
```

Both moves considerably improve the efficiency of our MCMC analysis.

Setting up the time intervals

In `RevBayes` you actually have the possibility to specify unequal time intervals or even different intervals for the speciation and extinction rate. This is achieved by providing a vector of times when each interval ends. Here we simply break-up the most recent 80% of time since the root in equal-length intervals.

```
interval_times <- T.rootAge() * (1:NUM_INTERVALS) / (NUM_INTERVALS) * 0.8
```

This vector of times will be used for both the speciation and extinction rates. Also, remember that the times of the intervals represent ages going backwards in time.

Incomplete Taxon Sampling

We know that we have sampled 233 out of 367 living primate species. To account for this we can set the sampling parameter as a constant node with a value of 233/367. For simplicity, and since almost all species have been sampled, we assume *uniform* taxon sampling ([Höhna et al. 2011](#); [Höhna 2014](#)),

```
rho <- T.ntips()/367
```

Root age

The birth-death process requires a parameter for the root age. In this exercise we use a fixed tree and thus we know the age of the tree. Hence, we can get the value for the root from the [Magnuson-Ford and Otto \(2012\)](#) tree.

```
root_time <- T.rootAge()
```

The time tree

Now we have all of the parameters we need to specify the full episodic birth-death model. We initialize the stochastic node representing the time tree.

```
timetree ~ dnEpisodicBirthDeath(rootAge=T.rootAge(), lambdaRates=speciation,
lambdaTimes=interval_times, muRates=extinction, muTimes=interval_times, rho=rho,
samplingStrategy="uniform", condition="survival", taxa=taxa)
```

You may notice that we explicitly specify that we want to condition on survival. It is possible to change this condition to the *time of the process* or *the number of sampled taxa* too.

Then we attach data to the **timetree** variable.

```
timetree.clamp(T)
```

Finally, we create a workspace object of our whole model using the **model()** function.

```
mymodel = model(speciation)
```

The **model()** function traversed all of the connections and found all of the nodes we specified.

Running an MCMC analysis

Specifying Monitors

For our MCMC analysis, we need to set up a vector of *monitors* to record the states of our Markov chain. First, we will initialize the model monitor using the **mnModel** function. This creates a new monitor variable that will output the states for all model parameters when passed into a MCMC function.

```
monitors[++mni] = mnModel(filename="output/primates_EBD.log",printgen=10, separator =
    TAB)
```

Additionally, we create four separate file monitors, one for each vector of speciation and extinction rates and for each speciation and extinction rate epoch (*i.e.*, the times when the interval ends). We want to have the speciation and extinction rates stored separately so that we can plot them nicely afterwards.

```
monitors[++mni] = mnFile(filename="output/primates_EBD_speciation_rates.log",printgen
    =10, separator = TAB, speciation)
monitors[++mni] = mnFile(filename="output/primates_EBD_speciation_times.log",printgen
    =10, separator = TAB, interval_times)
monitors[++mni] = mnFile(filename="output/primates_EBD_extinction_rates.log",printgen
    =10, separator = TAB, extinction)
monitors[++mni] = mnFile(filename="output/primates_EBD_extinction_times.log",printgen
    =10, separator = TAB, interval_times)
```

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

```
monitors[++mni] = mnScreen(printgen=1000, extinction_sd, speciation_sd)
```

Initializing and Running the MCMC Simulation

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

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

First, we will run a pre-burnin to tune the moves and to obtain starting values from the posterior distribution.

```
mymcmc.burnin(generations=10000,tuningInterval=200)
```

Now, run the MCMC:

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

When the analysis is complete, you will have the monitored files in your output directory. You can then visualize the rates through time using R using our package `RevGadgets`. If you don't have the R-package `RevGadgets` installed, or if you have trouble with the package, then please read the separate tutorial about the package.

Just start R in the main directory for this analysis and then type the following commands:

```
library(RevGadgets)

tree <- read.nexus("data/primates_tree.nex")

rev_out <- rev.process.div.rates(speciation_times_file = "output/
  primates_EBD_speciation_times.log", speciation_rates_file = "output/
  primates_EBD_speciation_rates.log", extinction_times_file = "output/
  primates_EBD_extinction_times.log", extinction_rates_file = "output/
  primates_EBD_extinction_rates.log", tree, burnin=0.25,numIntervals=100)

pdf("EBD.pdf")
par(mfrow=c(2,2))
rev.plot.output(rev_out,use.geoscale=FALSE)
dev.off()
```

(Note, you may want to add a nice geological timescale to the plot by setting `use.geoscale=TRUE` but then you can only plot one figure per page.)

- The Rev file for performing this analysis: [mcmc_EBD.Rev](#).

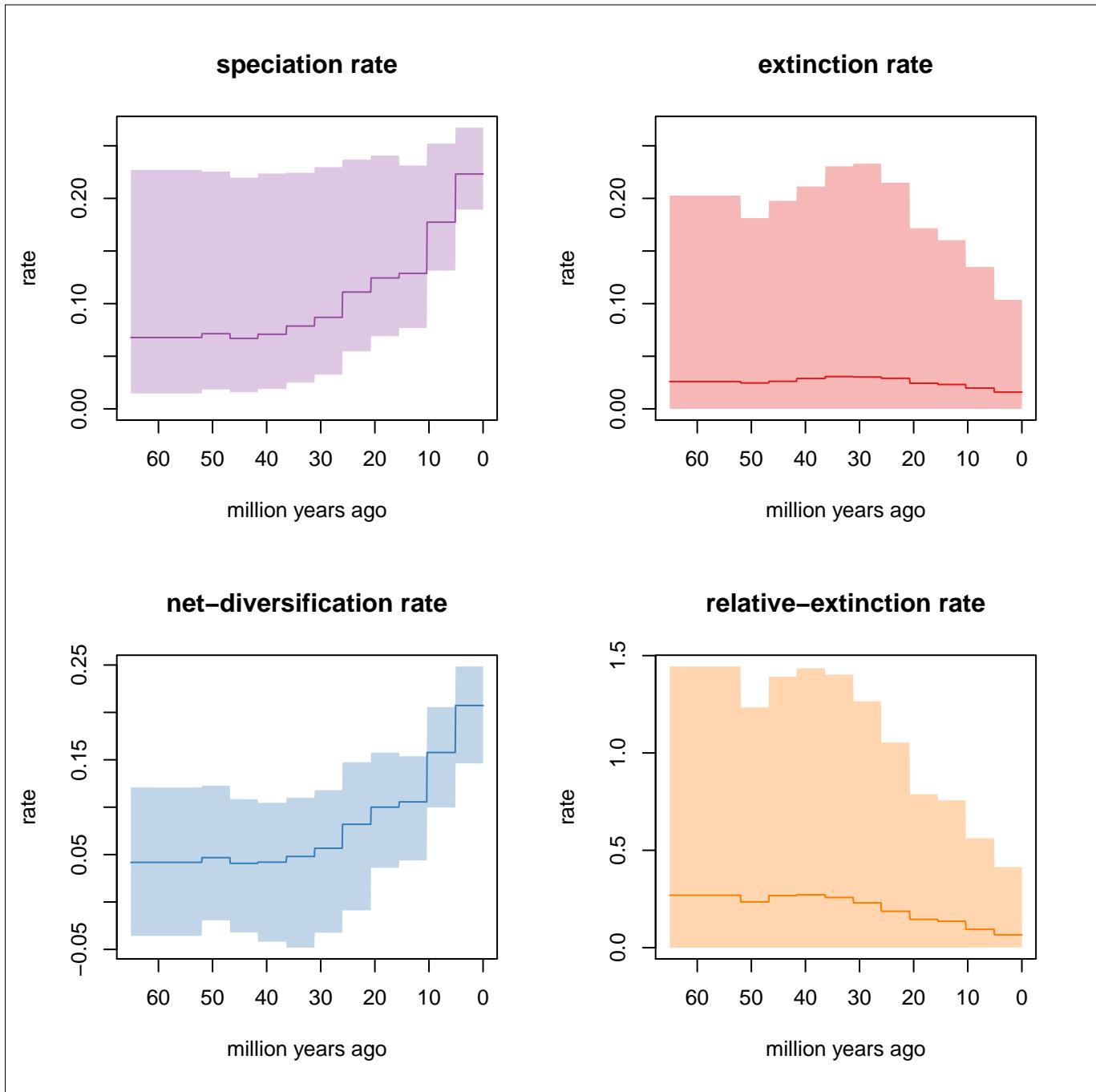


Figure 13.3: Resulting diversification rate estimations when using 10 intervals. You should obtain similar results when you use 10 intervals. The estimated rates might change when you use a different resolution, *i.e.*, a different number of intervals.

Exercise 1

- Run an MCMC simulation to estimate the posterior distribution of the speciation rate and extinction rate.
- Visualize the rate through time using R.
- Do you see evidence for rate decreases or increases? What is the general trend?
- Is there evidence for rate variation? Look at the estimates of `speciation_sd` and `extinction_sd`: Is there information in the data to change the estimates from the prior?
- Run the analysis using a different number of intervals, *e.g.*, 5 or 50. How do the rates change?

Exercise 2

- In our results we see that the extinction rate is fairly constant. Modify the model by using a constant-rate for the extinction rate parameter but still let the speciation rate vary through time.
 - Remove all previous occurrences of the extinction rates (*i.e.*, priors, parameters and moves).
 - Specify a lognormal prior distribution on the constant extinction rate
`(extinction ~ dnLognormal(-5, sd=2*0.587405))`
 - Add a move for the new extinction rate parameter
`moves[++mvi] = mvScale(extinction, weight=5.0)`.
 - Remove the argument `muTimes=interval_times` from the birth-death process.
- How does this influence your estimated rates?

Bibliography

- Höhna, S. 2014. Likelihood Inference of Non-Constant Diversification Rates with Incomplete Taxon Sampling. PLoS One 9:e84184.
- Höhna, S. 2015. The time-dependent reconstructed evolutionary process with a key-role for mass-extinction events. Journal of Theoretical Biology 380:321–331.
- Höhna, S., T. A. Heath, B. Boussau, M. J. Landis, F. Ronquist, and J. P. Huelsenbeck. 2014. Probabilistic graphical model representation in phylogenetics. Systematic Biology 63:753–771.
- Höhna, S., T. Stadler, F. Ronquist, and T. Britton. 2011. Inferring speciation and extinction rates under different species sampling schemes. Molecular Biology and Evolution 28:2577–2589.
- Magnuson-Ford, K. and S. P. Otto. 2012. Linking the investigations of character evolution and species diversification. The American Naturalist 180:225–245.
- Stadler, T. 2011. Mammalian phylogeny reveals recent diversification rate shifts. Proceedings of the National Academy of Sciences 108:6187–6192.

Chapter 14

Environmental-correlated Diversification Rate Estimation

Estimating Environmental-dependent Speciation & Extinction Rates

Outline

This tutorial describes how to specify a branching-process model with diversification rate correlated with an environmental variable in RevBayes. Diversification rates are assumed to be equal among all lineages but vary through time correlated with an environmental predictor variable. Thus, this model can be used to test for correlations between diversification rates and environmental variables, such as CO₂ and temperature. However, these tests are only to establish a correlation, not a causality.

As usual, we provide the probabilistic graphical model at the beginning of this tutorial. Hopefully this will help you to get a better idea of all the variables in the model and their dependencies. Our goal in this tutorial is to estimate the correlation coefficient between speciation and extinction rates to historical CO₂ measurements using Markov chain Monte Carlo (MCMC).

Requirements

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

- [Getting started](#)
- [Rev basics](#)
- [Basic Diversification Rate Estimation](#)
- [Diversification Rates Through Time](#)

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

For this tutorial it is particularly important that you have read the two tutorials on diversification rate estimation: [Basic Diversification Rate Estimation tutorial](#) and [Diversification Rates Through Time tutorial](#). Specifically the [Diversification Rates Through Time tutorial](#) present the underlying diversification model and thus foundation for this tutorial. Here we will build on the episodic diversification rate tutorial by modifying the prior model on diversification rates through time to depend on some environmental variable.

Data and files

We provide the data file(s) which we will use in this tutorial. You may want to use your own data instead. In the **data** folder, you will find the following files

- **primates_springer.tre**: Dated primates phylogeny including 369 out of 377 species from [Springer et al. \(2012\)](#).

→ Open the tree **data/primates_springer.tre** in FigTree.

Environmental-dependent Diversification Rates

The fundamental idea of this model is the question if diversification rates are correlated with an environmental variable. Examples of environmental variables are CO₂ and temperature. Have a look at Figure 14.1

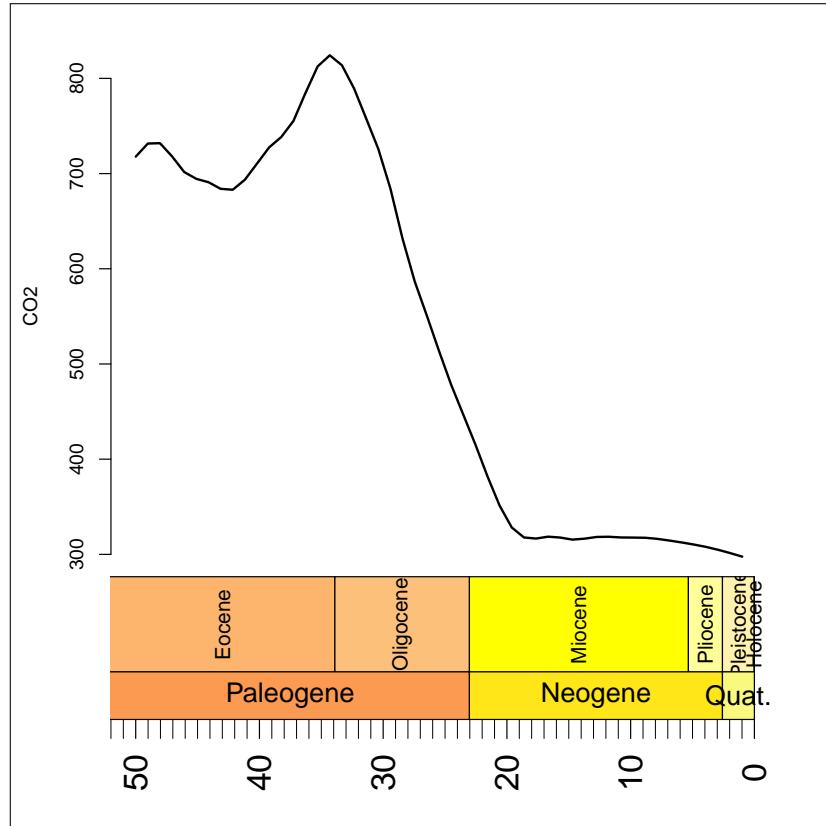


Figure 14.1: Estimates of historical CO₂ values. These estimates are obtained from XXX. The unit of CO₂ represents XXX.

which shows the historical value CO₂ in the last 50 million years. We can clearly see that the CO₂ dropped drastically around 30 million years ago.

In our previous [Diversification Rates Through Time tutorial](#) we estimated diversification as shown in Figure 14.2. We clearly see that diversification rates were not constant through time. Now we wonder if perhaps the diversification rates are correlated with CO₂.

We want to build on our episodic birth-death model so that our environmental correlation model collapses to the episodic birth-death model if there is no correlation. Recall that we used a Brownian motion model on the log-transformed rates. Hence, we assumed that the rates in the next time interval (epoch) have the current value as their expectation:

$$E[\log(\lambda(t))] = \log(\lambda(t - \Delta t)) \quad (14.1)$$

For the environmental dependent birth-death model, we have additional observation from the environmental variable. Thus, we know how much the environmental variable changed between time intervals (epochs). We can compute this change by taking the ratio between two consecutive measurements: $\frac{CO_2(t)}{CO_2(t-\Delta t)}$. Hence, if the CO₂ double from one epoch to the next we would compute a change of 2. This has the clear advantage that our computation is less sensitive to the unit and magnitude of the environmental variable.

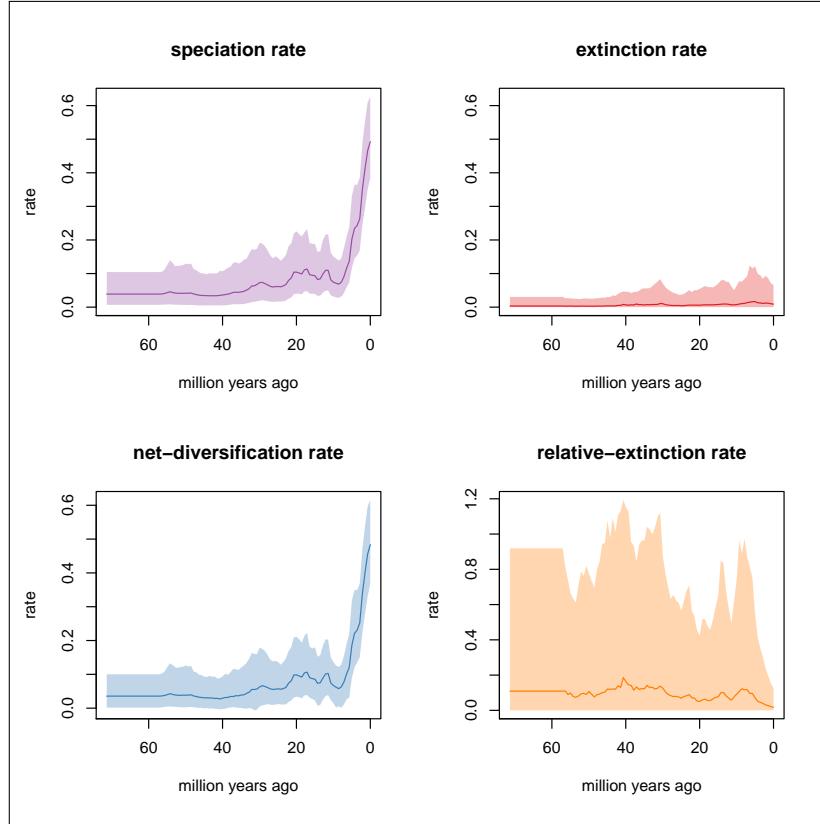


Figure 14.2: Estimated diversification rates through time. These estimates are taken from the episodic birth-death model with autocorrelated (Brownian motion) rate as described in the [Diversification Rates Through Time tutorial](#).

Now let us assume that our diversification rates shift synchronously with the environmental variable if they are actually correlated. Then we can express our expectation of the log-transformed diversification rate in the next time interval (epoch) as being equal the log-transform diversification rate in the current time interval plus the log-transformed change in the environmental variable:

$$E[\log(\lambda(t))] = \log(\lambda(t - \Delta t)) + \beta \times \log\left(\frac{\text{CO}_2(t)}{\text{CO}_2(t - \Delta t)}\right). \quad (14.2)$$

Here we denote the correlation coefficient by β . If $\beta > 0$ then there is a positive correlation between the speciation rate and CO_2 , that is, if the CO_2 increases then the speciation increases also. If $\beta < 0$ then there is a negative correlation between the speciation rate and CO_2 , that is, if the CO_2 increases then the speciation decreases. Finally, if $\beta = 0$ then there is no correlation and our model collapses to the episodic birth-death model.

In summary, we use a regression-like prior model for the speciation and extinction rate where the environmental variable (here CO_2) is the predictor variable. Specifically, we use a Brownian motion model for the log-transformed speciation and extinction rates where the expectation depends on the shift in the environmental variable. Thus, our model can be considered as a Brownian motion model with drift where the drift parameter is the environmental variable.

We will now walk you through setting up this analysis in RevBayes.

Read the tree

Begin by reading in the “observed” tree.

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

From this tree, we get some helpful variables, such as the taxon information which we need to instantiate the birth-death process.

```
taxa <- T.taxa()
```

Additionally, we initialize an iterator variable for our vector of moves and monitors.

```
mvi = 0  
mni = 0
```

Set up the environmental data

We take the CO₂ measurement from **XXX** and store the values on a vector; one measurement (value) per interval.

```
var <- v(297.6, 301.36, 304.84, 307.86, 310.36, 312.53, 314.48, 316.31, 317.42,  
317.63, 317.74, 318.51, 318.29, 316.5, 315.49, 317.64, 318.61, 316.6, 317.77,  
328.27, 351.12, 381.87, 415.47, 446.86, 478.31, 513.77, 550.74, 586.68, 631.48,  
684.13, 725.83, 757.81, 789.39, 813.79, 824.25, 812.6, 784.79, 755.25, 738.41,  
727.53, 710.48, 693.55, 683.04, 683.99, 690.93, 694.44, 701.62, 718.05, 731.95,  
731.56, 717.76)
```

Then we specify the maximum age of the measurements. This corresponds to the time of the last interval.

```
MAX_VAR_AGE = 50
```

We will later use this maximum age to compute the times for each interval by assuming that each interval is equal in time.

Finally, we create a helper variable that specifies the number of intervals.

```
NUM_INTERVALS = var.size()-1
```

This variable will help us to create the episodic diversification rate using a **for**-loop.

Setting up the time intervals

In RevBayes you actually have the possibility to specify unequal time intervals or even different intervals for the speciation and extinction rate. This is achieved by providing a vector of times when each interval ends. However, here we assume for simplicity that each interval has the same length because this is how we obtained our environmental data.

```
interval_times <- MAX_VAR_AGE * (1:NUM_INTERVALS) / NUM_INTERVALS
```

This vector of times will be used for both the speciation and extinction rates. Also, remember that the times of the intervals represent ages going backwards in time.

Specifying the model

Priors on amount of rate variation

We follow here exactly the prior specification as in the [Diversification Rates Through Time tutorial](#) because we want our model to collapse to the episodic birth-death if there is no correlation.

We start by specifying prior distributions on the rates. Each interval-specific speciation and extinction rate will be drawn from a normal distribution. Thus, we need a parameter for the standard deviation of those normal distributions. We use an exponential hyperprior with rate 1.0 to estimate the standard deviation, but assume that all speciation rates and all extinction rates share the same standard deviation. The motivation for an exponential hyperprior is that it has the highest probability density at 0 which would make the variance of rates between consecutive time intervals 0 and thus represent a constant rate process. The data will tell us if there should be much variation in rates through time. (You may want to experiment with this hyperprior if you are interested.)

```
speciation_sd ~ dnExponential(1.0)
extinction_sd ~ dnExponential(1.0)
```

We apply a simple scaling move on each prior parameter.

```
moves[++mvi] = mvScale(speciation_sd, weight=5.0)
moves[++mvi] = mvScale(extinction_sd, weight=5.0)
```

Specifying the correlation coefficients

Then we specify normal prior distributions on the correlation coefficient β for the speciation and extinction rate. Again, out total lack of prior knowledge, we will assume that the standard deviation of β is 1.0 and you may want to modify this value. Nevertheless, this normal prior distribution is motivated by being centered at 0.0 (no correlation) and gives equal weight to positive and negative correlations.

```
beta_speciation ~ dnNormal(0,1.0)
beta_extinction ~ dnNormal(0,1.0)
```

We apply simple sliding-window moves for the two correlation coefficients because they are defined on the whole real line.

```
moves[++mvi] = mvSlide(beta_speciation,delta=1.0,weight=10.0)
moves[++mvi] = mvSlide(beta_extinction,delta=1.0,weight=10.0)
```

Additionally, we might be interested in the posterior probability that there is a positive correlation, $\mathbb{P}(\beta > 0)$, or a negative correlation, $\mathbb{P}(\beta < 0)$, respectively. We achieve this using a deterministic variable that is 1 if $\beta < 0$

```
speciation_corr_neg_prob := ifelse(beta_speciation < 0.0, 1, 0)
extinction_corr_neg_prob := ifelse(beta_extinction < 0.0, 1, 0)
speciation_corr_pos_prob := ifelse(beta_speciation > 0.0, 1, 0)
extinction_corr_pos_prob := ifelse(beta_extinction > 0.0, 1, 0)
```

Note that in this model the probability of β being 0.0 ($\mathbb{P}(\beta = 0) = 0$) because we are working with a prior and posterior *density* on β and thus any specific value, *e.g.*, 0.0, has a probability of 0.0. We will circumvent this issue in the next chapter when we use reversible-jump MCMC to set β specifically to 0.0. Here you can also check that the posterior probability of **speciation_corr_pos_prob** equals 1-**speciation_corr_neg_prob**.

Specifying correlated rates

As we mentioned before, we will apply normal distributions as priors for each log-transformed rate. We begin with the rate at the present which is our initial rate parameter. The rates at the present will be specified slightly differently because they are not correlated to any previous rates. This is because we are actually modeling rate-changes backwards in time and there is no previous rate for the rate at the present.

We use a uniform distribution between -10 and 10 because of our lack of prior knowledge on the diversification rate. This actually means that we allow speciation and extinction rates between e^{-10} and e^{10} we should clearly cover the true values. (Note that for diversification rate estimates e^{-10} is virtually 0 since the rate is so slow).

```
log_speciation[1] ~ dnUniform(-10.0,10.0)
log_speciation[1] ~ dnUniform(-10.0,10.0)
```

Notice that we store the diversification rate variables in vectors. Storing the rate parameters in vectors will be useful and important later when we pass the rates into the birth-death process.

We apply simple sliding window moves for the rates. Normally we would use scaling moves but in this case we work on the log-transformed parameters and thus sliding moves perform better. (If you are keen you can test the differences.)

```
moves[++mvi] = mvSlide(log_speciation[1], weight=2)
moves[++mvi] = mvSlide(log_extinction[1], weight=2)
```

Now we transform the diversification rate parameters into actual rates.

```
speciation[1] := exp( log_speciation[1] )
extinction[1] := exp( log_extinction[1] )
```

Next, we specify the speciation and extinction rates for each time interval (*i.e.*, epoch). This can be done efficiently using a **for**-loop. We will use a specific index variable so that we can easier refer to the rate at the previous interval. Remember that we want to model the rates as a Brownian motion, which we achieve by specify a normal distribution as the prior distribution on the rates centered around the previous rate plus the change in the environmental variable (*i.e.*, the mean is equal to the previous rate plus the log-transformed ratio of the environmental variable divided by the previous value).

```
for (i in 1:NUM_INTERVALS) {
    index = i+1

    expected_speciation[index] := log_speciation[i] + beta_speciation * ln( var[index]
        / var[i] )
    expected_extinction[index] := log_extinction[i] + beta_extinction * ln( var[index]
        / var[i] )

    log_speciation[index] ~ dnNormal( mean=expected_speciation[index], sd=speciation_sd
        )
    log_extinction[index] ~ dnNormal( mean=expected_extinction[index], sd=extinction_sd
        )

    moves[++mvi] = mvSlide(log_speciation[index], weight=2)
    moves[++mvi] = mvSlide(log_extinction[index], weight=2)

    speciation[index] := exp( log_speciation[index] )
```

```

extinction[index] := exp( log_extinction[index] )

}

```

Finally, we apply moves that slide all values in the rate vectors, *i.e.*, all speciation or extinction rates. We will use an **mvVectorSlide** move.

```

moves[++mvi] = mvVectorSlide(log_speciation, weight=10)
moves[++mvi] = mvVectorSlide(log_extinction, weight=10)

```

Additionally, we apply a **mvShrinkExpand** move which changes the spread of several variables around their mean.

```

moves[++mvi] = mvShrinkExpand( log_speciation, sd=speciation_sd, weight=10 )
moves[++mvi] = mvShrinkExpand( log_extinction, sd=extinction_sd, weight=10 )

```

Both moves considerably improve the efficiency of our MCMC analysis.

Incomplete Taxon Sampling

We know that we have sampled 367 out of 377 living primate species. To account for this we can set the sampling parameter as a constant node with a value of 367/377. For simplicity, and since almost all species have been sampled, we assume *uniform* taxon sampling (Höhna et al. 2011; Höhna 2014),

```

rho <- T.nTips()/377

```

Root age

The birth-death process requires a parameter for the root age. In this exercise we use a fix tree and thus we know the age of the tree. Hence, we can get the value for the root from the Springer et al. (2012) tree.

```

root_time <- T.rootAge()

```

The time tree

Now we have all of the parameters we need to specify the full episodic birth-death model. We initialize the stochastic node representing the time tree.

```
timetree ~ dnEpisodicBirthDeath(rootAge=T.rootAge(), lambdaRates=speciation,
    lambdaTimes=interval_times, muRates=extinction, muTimes=interval_times, rho=rho,
    samplingStrategy="uniform", condition="survival", taxa=taxa)
```

You may notice that we explicitly specify that we want to condition on survival. It is possible to change this condition to the *time of the process* or *the number of sampled taxa* too.

Then we attach data to the **timetree** variable.

```
timetree.clamp(T)
```

Finally, we create a workspace object of our whole model using the **model()** function.

```
mymodel = model(speciation)
```

The **model()** function traversed all of the connections and found all of the nodes we specified.

Running an MCMC analysis

Specifying Monitors

For our MCMC analysis, we need to set up a vector of *monitors* to record the states of our Markov chain. First, we will initialize the model monitor using the **mnModel** function. This creates a new monitor variable that will output the states for all model parameters when passed into a MCMC function.

```
monitors[++mni] = mnModel(filename="output/primates_EBD_Corr.log", printgen=10,
    separator = TAB)
```

Additionally, we create four separate file monitors, one for each vector of speciation and extinction rates and for each speciation and extinction rate epoch (*i.e.*, the times when the interval ends). We want to have the speciation and extinction rates stored separately so that we can plot them nicely afterwards.

```
monitors[++mni] = mnFile(filename="output/primates_EBD_Corr_speciation_rates.log",
    printgen=10, separator = TAB, speciation)
monitors[++mni] = mnFile(filename="output/primates_EBD_Corr_speciation_times.log",
    printgen=10, separator = TAB, interval_times)
monitors[++mni] = mnFile(filename="output/primates_EBD_Corr_extinction_rates.log",
    printgen=10, separator = TAB, extinction)
monitors[++mni] = mnFile(filename="output/primates_EBD_Corr_extinction_times.log",
    printgen=10, separator = TAB, interval_times)
```

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

```
monitors[++mni] = mnScreen(printgen=1000, beta_speciation, beta_extinction)
```

Initializing and Running the MCMC Simulation

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

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

First, we will run a pre-burnin to tune the moves and to obtain starting values from the posterior distribution.

```
mymcmc.burnin(generations=10000,tuningInterval=200)
```

Now, run the MCMC:

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

When the analysis is complete, you will have the monitored files in your output directory. You can then visualize the rates through time using R using our package `RevGadgets`. If you don't have the R-package `RevGadgets` installed, or if you have trouble with the package, then please read the separate tutorial about the package.

Just start R in the main directory for this analysis and then type the following commands:

```
library(RevGadgets)
tree <- read.tree("data/primates_Springer.tre")

# the CO2 values as a reference in our plot
co2 <- c(297.6, 301.36, 304.84, 307.86, 310.36, 312.53, 314.48, 316.31, 317.42,
  317.63, 317.74, 318.51, 318.29, 316.5, 315.49, 317.64, 318.61, 316.6, 317.77,
  328.27, 351.12, 381.87, 415.47, 446.86, 478.31, 513.77, 550.74, 586.68, 631.48,
  684.13, 725.83, 757.81, 789.39, 813.79, 824.25, 812.6, 784.79, 755.25, 738.41,
  727.53, 710.48, 693.55, 683.04, 683.99, 690.93, 694.44, 701.62, 718.05, 731.95,
  731.56, 717.76)

MAX_VAR_AGE = 50
```

```

NUM_INTERVALS = length(co2)
co2_age <- MAX_VAR_AGE * (1:NUM_INTERVALS) / NUM_INTERVALS
predictor.ages <- co2_age
predictor.var <- co2

rev_out <- rev.process.div.rates(speciation_times_file = "output/
    primates_EBD_Corr_speciation_times.log",
        speciation_rates_file = "output/
            primates_EBD_Corr_speciation_rates.log",
        extinction_times_file = "output/
            primates_EBD_Corr_extinction_times.log",
        extinction_rates_file = "output/
            primates_EBD_Corr_extinction_rates.log",
        tree,
        burnin=0.25,numIntervals=100)

pdf("EBD_Corr.pdf")
par(mfrow=c(2,2))
rev.plot.div.rates(rev_out, predictor.ages=co2_age, predictor.var=co2, use.geoscale=
    TRUE)
dev.off()

```

- The Rev file for performing this analysis: [mcmc_EBD.Rev](#).

A brief discussion on estimated diversification rates

Figure 15.3 shows the estimated diversification rates through time and the CO₂. If you compare these estimates with Figure 14.2 then you may notice that the diversification rate estimate are virtually identical. This is a good sign for the analysis because it shows that the information in the estimates comes from the data (the tree in this case) and not from the assumed model. Thus, we are not artificially forcing the diversification rates to follow our environmental variable but instead estimate if there is a correlation. Small deviation between the estimated rates under the different analyses are expected because there will be some interaction between the environmental variable and the diversification rate estimates. Additionally, the uncertainty in estimated diversification rates through time is large und minor changes are within this uncertainty.

Exercise 1

- Run an MCMC simulation to estimate the posterior distribution of the speciation rate and extinction rate.
- Visualize the rate through time using R.
- Open the file `output/primates_EBD_Corr.log` in Tracer. What is the estimated probability that $\beta < 0$? You'll find the estimate in the variable `speciation_corr_neg_prob`.
- We specified a normal prior with mean 0 on β and thus used a prior probability of 0.5 that $\beta < 0$. Now you can use the posterior ratio divided by the prior ratio to compute the Bayes factor. What is the Bayes factor support for or against a positive correlation between the speciation rate and CO₂?

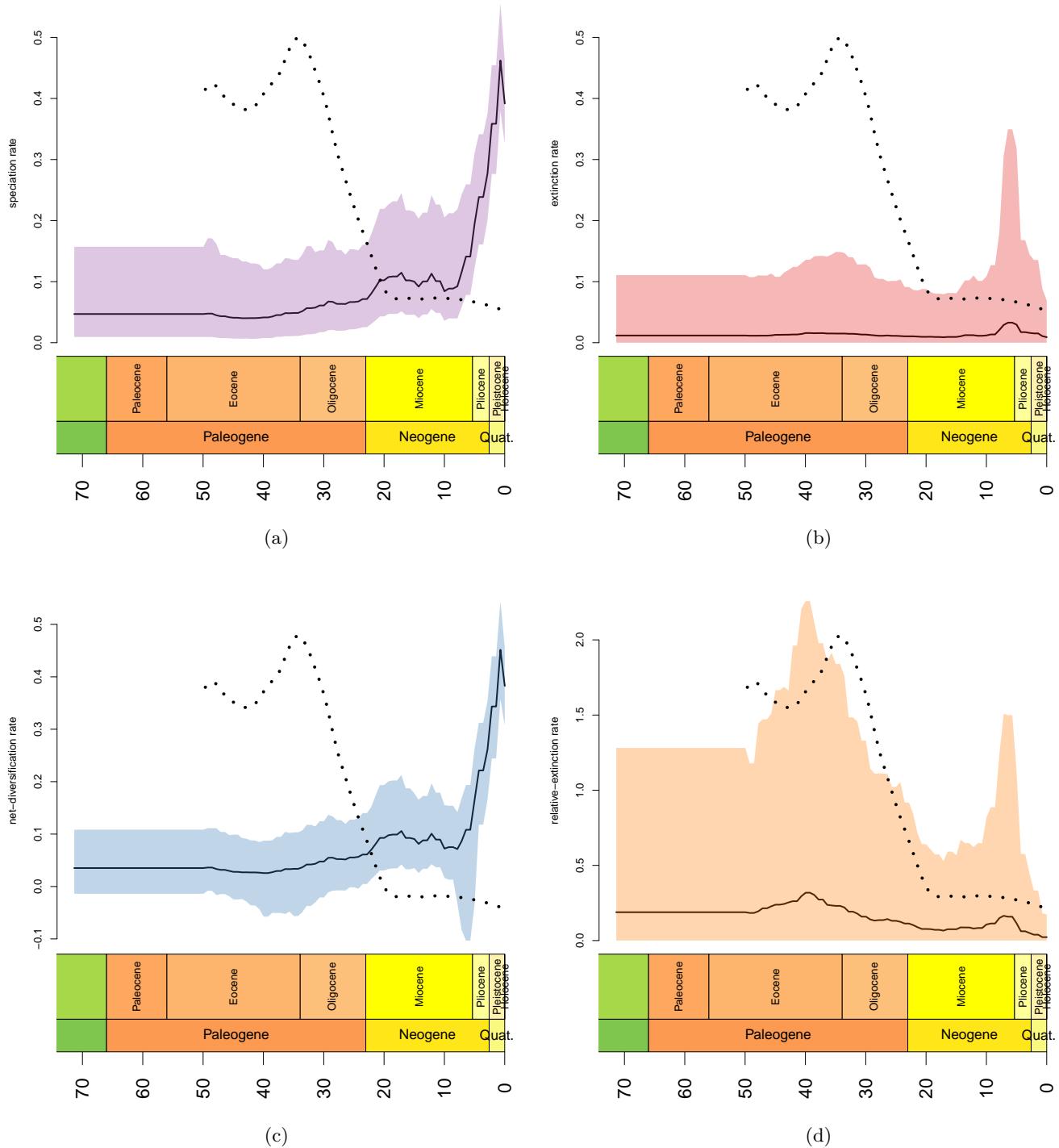


Figure 14.3: Resulting diversification rate estimations

- Similarly, is there support for a positive or negative correlation between the extinction rate and CO₂?

Testing for correlation using reversible-jump MCMC

In the previous exercise we wanted that our model collapses to the episodic birth-death process if there is no environmental correlation. We achieved this by setting up our prior model so that if $\beta = 0$ the model collapses. However, we also used a normal prior distribution with mean 0.0 and standard deviation 1.0 for β . Thus, we implicitly specified that β being exactly 0.0 has probability 0.0 because every specific value of a continuous distribution has a 0.0 probability despite having a positive probability density. For example, you might notice that you will never sample in your MCMC run the value 0.0 exactly although we might sample values that are close to 0.0.

Now we want to use reversible jump MCMC to test specifically if the hypothesis $\beta = 0$ is rejected. Remember that reversible jump MCMC can estimate the posterior probability for different models. The first model will be that $\beta = 0$ and the second model will be that $\beta \sim \text{norm}(0, 1)$. Then we can simply compute Bayes factors by computing the posterior ratio divided by the prior ratio to assess the support for either model.

In RevBayes we have a very flexible way to specify a reversible-jump MCMC. We can provide any constant value and distribution to the distribution `dnReversibleJumpMixture`. This will mean that the value, `beta_speciation` and `beta_extinction`, will either take on the constant value or drawn from the base-distribution.

```
beta_speciation ~ dnReversibleJumpMixture(constantValue=0.0, baseDistribution=dnNormal(0,1.0), p=0.5)
beta_extinction ~ dnReversibleJumpMixture(constantValue=0.0, baseDistribution=dnNormal(0,1.0), p=0.5)
```

Additionally we also need a specific move that switches if the value is equal to the constant value or drawn from the base-distribution. This is where we use the reversible-jump move `mvRJSwitch`.

```
moves[++mvi] = mvRJSwitch(beta_extinction, weight=5)
```

Now we can also monitor for convenience what the probability of `beta_speciation` and `beta_extinction` being 0.0 is. We will set this up by a deterministic variable that will be 1.0 if $\beta \neq 0$ and will be 0.0 if $\beta = 0.0$. Thus the two variables `speciation_corr_prob` and `extinction_corr_prob` represent the probability that there is a correlation between the speciation rate or the extinction rate and CO₂.

```
speciation_corr_prob := ifelse(beta_speciation == 0.0, 0, 1)
extinction_corr_prob := ifelse(beta_extinction == 0.0, 0, 1)
```

These are the only necessary changes to the above analysis to run a reversible-jump MCMC.

Exercise 2

- Make a copy of the script `mcmc_EBD_Corr.Rev` and call it `mcmc_EBD_Corr_RJ.Rev`.
- Replace the prior distribution on `beta_speciation` and `beta_extinction` to use the `dnReversibleJumpMixture` instead.
- Also add the new moves `mvRJSwitch` and deterministic variables `speciation_corr_prob` and `extinction_corr_prob`.
- Don't forget to change the output filenames in the monitors, *e.g.*, add `_RJ` to the name.
- Run the reversible-jump MCMC analysis.
- Open the file `output/primates_EBD_Corr_RJ.log` in Tracer. What is the estimated probability that $\beta = 0$? You'll find the estimate in the variable `speciation_corr_prob`.
- We specified a prior probability of 0.5 on β being fixed to 0.0. Now you can use the posterior ratio divided by the prior ratio to compute the Bayes factor. What is the Bayes factor support for or against any correlation between the speciation rate and CO₂?
- Similarly, is there support for a positive or negative correlation between the extinction rate and CO₂?

Bibliography

- Höhna, S. 2014. Likelihood Inference of Non-Constant Diversification Rates with Incomplete Taxon Sampling. PLoS One 9:e84184.
- Höhna, S., T. Stadler, F. Ronquist, and T. Britton. 2011. Inferring speciation and extinction rates under different species sampling schemes. Molecular Biology and Evolution 28:2577–2589.
- Springer, M. S., R. W. Meredith, J. Gatesy, C. A. Emerling, J. Park, D. L. Rabosky, T. Stadler, C. Steiner, O. A. Ryder, J. E. Janečka, et al. 2012. Macroevolutionary dynamics and historical biogeography of primate diversification inferred from a species supermatrix. PLoS One 7:e49521.

Chapter 15

Diversification Rate Estimation with Missing Taxa

Estimating Speciation & Extinction Rates Through Time

Outline

This tutorial describes how to specify different models of incomplete taxon sampling (Höhna et al. 2011; Höhna 2014) for estimating diversification rates in RevBayes (Höhna et al. 2016). Incomplete taxon sampling, if not modeled correctly, severely biases diversification-rate parameter estimates (Cusimano and Renner 2010; Höhna et al. 2011). Specifically, we will discuss *uniform*, *diversified*, and *empirical* taxon sampling. All analyses in this tutorial will focus on diversification rate estimation through-time and use a birth-death process where diversification rates vary episodically which we model by piecewise constant rates RevBayes (Höhna 2015; May et al. 2016). The probabilistic graphical model is given only once for this tutorial as an overview. The model itself does not change between the different analyses; only the assumptions of incomplete taxon sampling. For each analysis you will estimate speciation and extinction rates through-time using Markov chain Monte Carlo (MCMC) and assess the impact of incomplete taxon sampling as well as the sampling scheme.

Requirements

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

- [Getting started](#)
- [Rev basics](#)
- [Basic Diversification Rate Estimation](#)
- [Diversification Rates Through Time](#)

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

For this tutorial it is especially important that you have read the two tutorials on diversification rate estimation: [Basic Diversification Rate Estimation tutorial](#) and [Diversification Rates Through Time tutorial](#). Specifically the [Diversification Rates Through Time tutorial](#) present the underlying diversification model and thus foundation for this tutorial. Here we will build on the tutorial by modifying the assumptions of incomplete taxon sampling in different ways.

Data and files

We provide the data file(s) which we will use in this tutorial. You may want to use your own data instead. In the **data** folder, you will find the following file

- **primates.tre**: Dated primates phylogeny including 23 out of 377 species.

Note that we use here the small primate phylogeny including only 23 of the 377 taxa instead of the much more complete primate phylogeny from [Springer et al. \(2012\)](#). This choice was solely made to emphasize

the point and impact of incomplete taxon sampling, which is a very prominent feature in many large scale phylogenies.

- Open the tree `data/primates.tre` in FigTree.

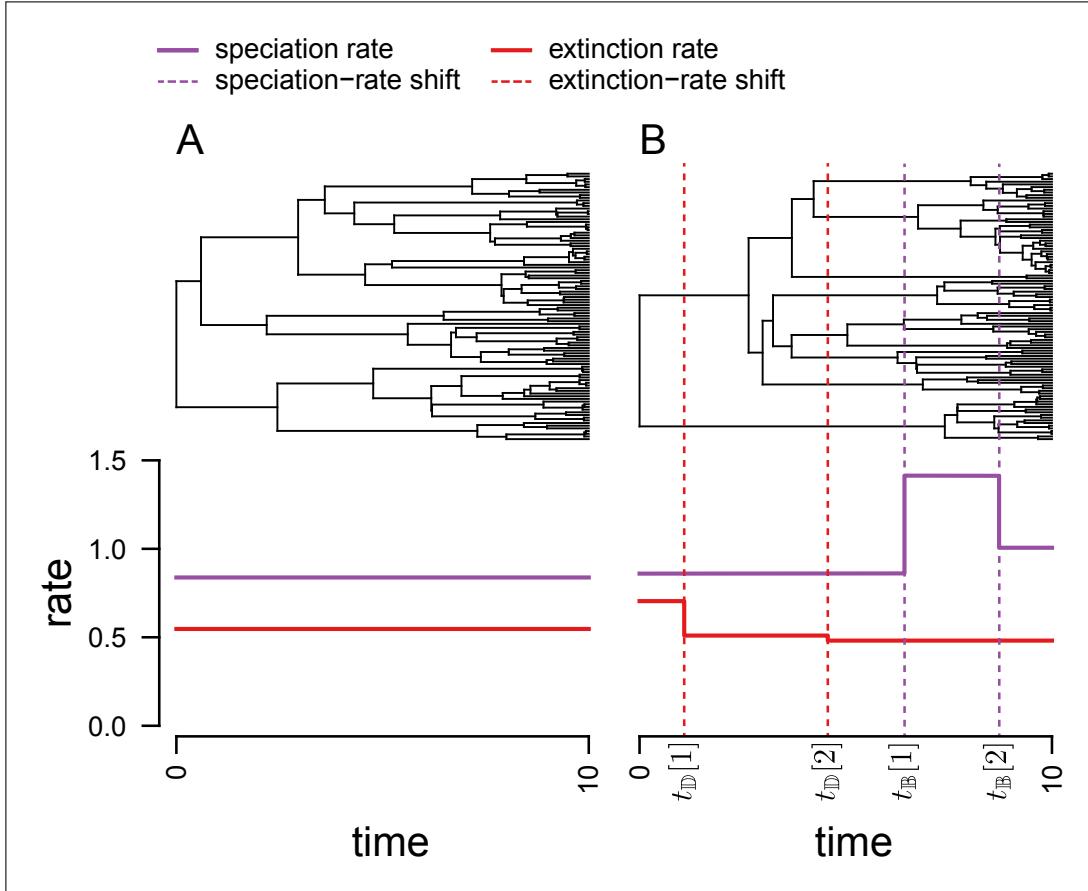


Figure 15.1: Two scenarios of birth-death models. On the left we show constant diversification. On the right we show an example of an episodic birth-death process where rates are constant in each time interval (epoch). The top panel of this figure shows example realization under the given rates.

Episodic Birth-Death Model

Here we study the impact of incomplete taxon sampling by estimating diversification rates through time. The goal is to compare the impact of different taxon sampling strategies rather than the description of the diversification-rate model itself. The episodic birth-death model used here is equivalent to the model described in our previous tutorial. Please read the [Diversification Rates Through Time](#) tutorial for more detailed information about the model.

We have included in Figure 15.1 again the cartoon of episodic birth-death process with piecewise constant diversification rates. As mentioned above, diversification rate estimates are biased when only a fraction of the species is included and the sampling scheme is not accommodated appropriately (Cusimano and Renner 2010; Höhna et al. 2011; Cusimano et al. 2012; Höhna 2014). Hence, the diversification rate through time model will be an excellent example to study the impact of the assumed incomplete sampling strategy on diversification rates.

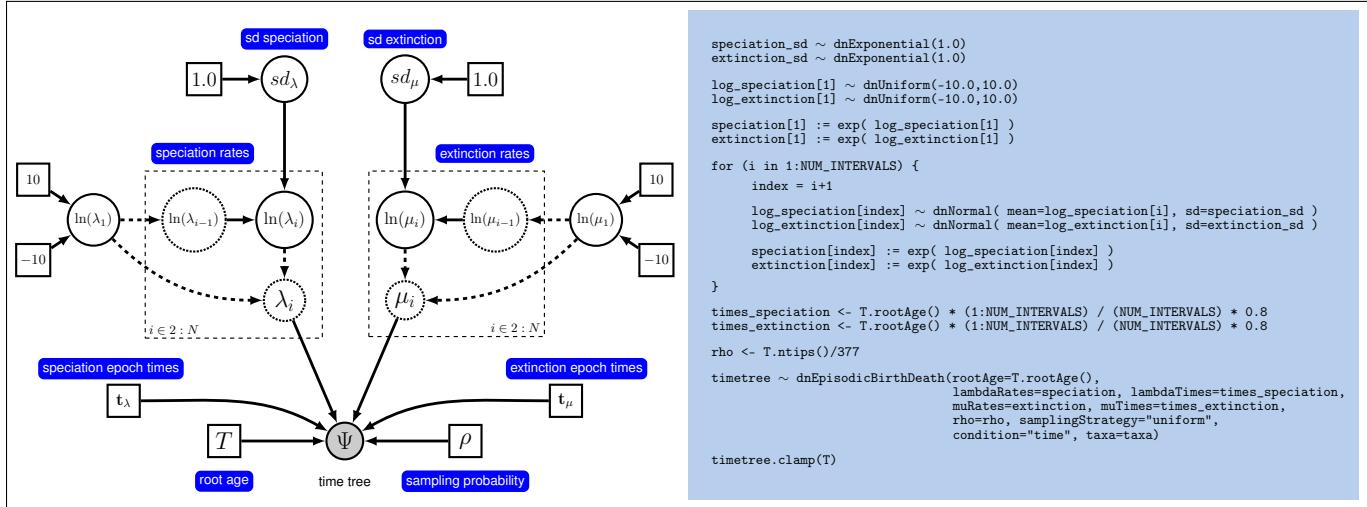


Figure 15.2: A graphical model with the outline of the Rev code. On the left we see the graphical model describing the correlated (Brownian motion) model for rate-variation through time. On the right we show the correspond Rev commands to instantiate this model in computer memory. This figure gives a complete overview of the model that we use here in this analysis.

We additionally include the graphical model representing the episodic birth-death process with autocorrelated diversification rates. This graphical model shows you which variables are included in the model, and the dependency between the variables. Thus, it makes the structure and assumption of the model clear and visible instead of a black-box (Höhna et al. 2014). Here we will focus only on the variable **rho**, the sampling probability, to model incomplete taxon sampling.

Specifying the model in Rev

We will give a very brief and compressed version of the model with fewer comments and explanation. The more detailed explanation can be found in the [Diversification Rates Through Time tutorial](#). Any attempt from us to present the full description here would only be a duplication/copy of the original tutorial with the additional to be less complete and less up to date.

Here are the summarized steps for running the episodic birth-death model in Rev.

```

#####
# Reading in the Data #
#####

### Read in the "observed" tree
T <- readTrees("data/primates.tre")[1]

# Get some useful variables from the data. We need these later on.
taxa <- T.taxa()

# set my move index
mvi = 0
mni = 0

```

```

NUM_INTERVALS = 10

#####
# Create the rates #
#####

# first we create the standard deviation of the rates between intervals
# draw the sd from an exponential distribution
speciation_sd ~ dnExponential(1.0)
moves[++mvi] = mvScale(speciation_sd, weight=5.0)

extinction_sd ~ dnExponential(1.0)
moves[++mvi] = mvScale(extinction_sd, weight=5.0)

# create a random variable at the present time
log_speciation[1] ~ dnUniform(-10.0,10.0)
log_extinction[1] ~ dnUniform(-10.0,10.0)

# apply moves on the rates
moves[++mvi] = mvSlide(log_speciation[1], weight=2)
moves[++mvi] = mvSlide(log_extinction[1], weight=2)

speciation[1] := exp( log_speciation[1] )
extinction[1] := exp( log_extinction[1] )

for (i in 1:NUM_INTERVALS) {
    index = i+1

    # specify normal priors (= Brownian motion) on the log of the rates
    log_speciation[index] ~ dnNormal( mean=log_speciation[i], sd=speciation_sd )
    log_extinction[index] ~ dnNormal( mean=log_extinction[i], sd=extinction_sd )

    # apply moves on the rates
    moves[++mvi] = mvSlide(log_speciation[index], weight=2)
    moves[++mvi] = mvSlide(log_extinction[index], weight=2)

    # transform the log-rate into actual rates
    speciation[index] := exp( log_speciation[index] )
    extinction[index] := exp( log_extinction[index] )
}

```

```

}

moves[++mvi] = mvVectorSlide(log_speciation, weight=10)
moves[++mvi] = mvVectorSlide(log_extinction, weight=10)

moves[++mvi] = mvShrinkExpand( log_speciation, sd=speciation_sd, weight=10 )
moves[++mvi] = mvShrinkExpand( log_extinction, sd=extinction_sd, weight=10 )

interval_times <- T.rootAge() * (1:NUM_INTERVALS) / (NUM_INTERVALS) * 0.8

### rho is the probability of sampling species at the present
### fix this to 23/377, since there are ~377 described species of primates
### and we have sampled 23
rho <- T.nTips()/377

timetree ~ dnEpisodicBirthDeath(rootAge=T.rootAge(), lambdaRates=speciation,
    lambdaTimes=interval_times, muRates=extinction, muTimes=interval_times, rho=rho,
    samplingStrategy="uniform", condition="survival", taxa=taxa)

### clamp the model with the "observed" tree
timetree.clamp(T)

#####
# The Model #
#####

### workspace model wrapper #####
mymodel = model(timetree)

### set up the monitors that will output parameter values to file and screen
monitors[++mni] = mnModel(filename="output/primates_uniform.log", printgen=10,
    separator = TAB)
monitors[++mni] = mnFile(filename="output/primates_uniform_speciation_rates.log",
    printgen=10, separator = TAB, speciation)
monitors[++mni] = mnFile(filename="output/primates_uniform_speciation_times.log",
    printgen=10, separator = TAB, interval_times)#
monitors[++mni] = mnFile(filename="output/primates_uniform_extinction_rates.log",
    printgen=10, separator = TAB, extinction)
monitors[++mni] = mnFile(filename="output/primates_uniform_extinction_times.log",
    printgen=10, separator = TAB, interval_times)
monitors[++mni] = mnScreen(printgen=1000, extinction_sd, speciation_sd)

```

```
#####
# The Analysis #
#####

### workspace mcmc ####
mymcmc = mcmc(mymodel, monitors, moves)

### pre-burnin to tune the proposals ####
mymcmc.burnin(generations=10000,tuningInterval=200)

### run the MCMC ####
mymcmc.run(generations=50000)
```

This Rev code shows the template for estimating episodic diversification rates. In the next sections we will tweak the script for the different sampling schemes.

Uniform Taxon Sampling

In our first analysis we will assume *uniform* taxon sampling (see Höhna et al. 2011; Höhna 2014). Uniform taxon sampling is the oldest and most basic technique to include incomplete taxon sampling (Nee et al. 1994; Yang and Rannala 1997). Uniform taxon sampling corresponds to the assumption that every species has the same probability ρ to be included (*i.e.*, sampled) in our study. Imagine flipping a coin that has the probability ρ to show up heads. For every species you flip the coin and are going to include the species, for example by sequencing it, in your study. This is what the assumption of uniform taxon sampling means.

For our study, we know that we have sampled 23 out of 377 living primate species. To account for this we can set the sampling parameter as a constant variable with a value of 23/377.

```
rho <- T.ntips()/377
```

Note that in principle you could specify a hyperprior distribution on the sampling probability **rho**. However, all three parameters (speciation rate, extinction rate, and sampling probability) are not identifiable (Stadler 2009). Nevertheless, we could specify informative priors on the sampling fraction if, for example, we know that the true diversity is in same range.

Moreover, we specify the *uniform* sampling scheme by setting **samplingStrategy="uniform"** in the birth-death process.

```
timetree ~ dnEpisodicBirthDeath(rootAge=T.rootAge(), lambdaRates=speciation,
lambdaTimes=interval_times, muRates=extinction, muTimes=interval_times, rho=rho,
samplingStrategy="uniform", condition="survival", taxa=taxa)
```

This is exactly what we did in the `Rev` script above.

- The `Rev` file for performing this analysis: [mcmc_uniform.Rev](#).

Exercise 1

- Run an MCMC simulation to estimate the posterior distribution of the speciation rate and extinction rate through time assuming *uniform* taxon sampling. You can use the script `mcmc_uniform.Rev` to run the analysis.
- Visualize the rate through time using `R` and `RevGadgets`.

Summarizing and plotting diversification rates through time

When the analysis is complete, you will have the monitored files in your output directory. You can then visualize the rates through time using `R` using our package `RevGadgets`. If you don't have the R-package `RevGadgets` installed, or if you have trouble with the package, then please read the separate tutorial about the package.

Just start `R` in the main directory for this analysis and then type the following commands:

```
library(RevGadgets)

tree <- read.nexus("data/primates.tre")
files <- c("output/primates_uniform_speciation_times.log", "output/
         primates_uniform_speciation_rates.log", "output/primates_uniform_extinction_times.
         log", "output/primates_uniform_extinction_rates.log")

rev_out <- rev.process.output(files, tree, burnin=0.25, numIntervals=100)

pdf("uniform.pdf")
par(mfrow=c(2,2))
rev.plot.output(rev_out)
dev.off()
```

You can see the resulting plot in Figure 15.3.

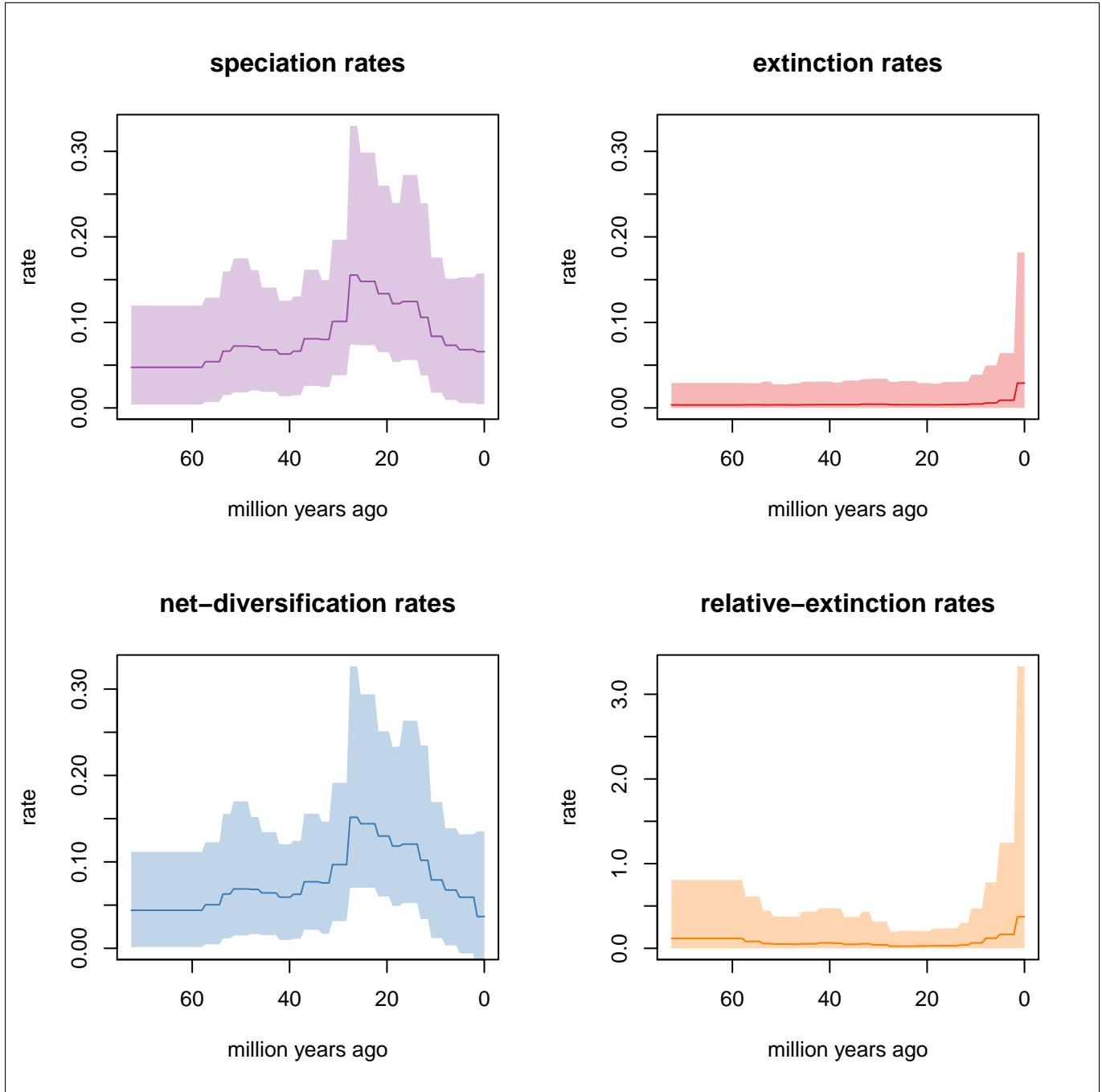


Figure 15.3: Resulting diversification rate estimations when using 20 intervals and assuming uniform taxon sampling. You should create similar plots for the other sampling schemes and compare the rates through time.

Diversified Taxon Sampling

In the previous analysis we assumed that species were sampled uniformly. However, this assumption is very often violated (Cusimano and Renner 2010; Höhna et al. 2011). For example, the primate phylogeny that we use in this tutorial includes one species for almost all genera. Thus, we had selected the species for the study neither uniformly nor randomly but instead by including one species per genera and hence

maximizing diversity. This sampling scheme is called *diversified* taxon sampling (Höhna et al. 2011).

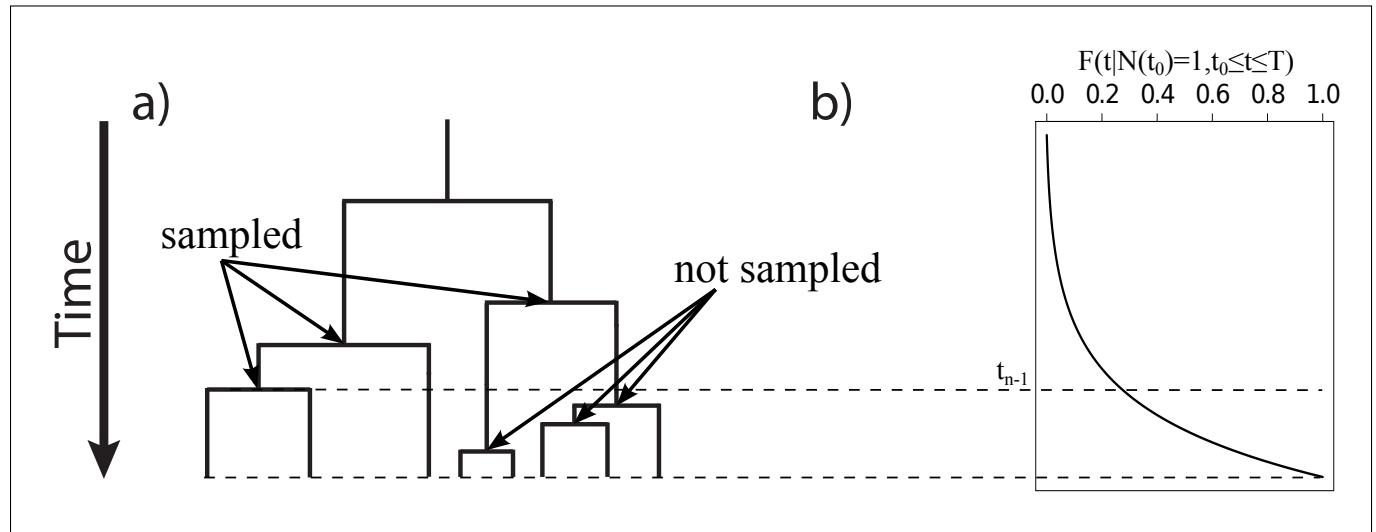


Figure 15.4: Example of diversified taxon sampling. a) An example phylogeny showing that all species after a certain time are not sampled. b) The cumulative probability of a speciation event occurring as a function of time. Here we see that the highest probability for a speciation event is more recently.

Figure 15.4 shows an example of diversified sampling. The example shows the same tree as in Figure 11.2 where 5 species are sampled. In fact, here we sampled 5 species so that every group is included and the most recent speciation events are excluded (not sampled).

In RevBayes we can specify *diversified* taxon sampling in the same way as we did *uniform* taxon sampling. First, we specify a constant variable for the sampling fraction `rho` which we set to the number of included (sampled) taxa divided by the number of total taxa in the group.

```
rho <- T.nTips()/377
```

Then, we specify that our sampling strategy was diversified by setting the argument of the birth-death process to `samplingStrategy="diversified"`.

```
timetree ~ dnEpisodicBirthDeath(rootAge=T.rootAge(), lambdaRates=speciation,
    lambdaTimes=interval_times, muRates=extinction, muTimes=interval_times, rho=rho,
    samplingStrategy="diversified", condition="time", taxa=taxa)
```

This is all we needed to do to change our previous script to model *diversified* taxon sampling.

Exercise 2

- Copy the Rev script `mcmc_uniform.Rev` and name it `mcmc_diversified.Rev`.
- Make the changes so that you assume now *diversified* taxon sampling.

- Change the file names in the monitors from **uniform** to **diversified**.
- Run the analysis and plot the diversification rates.
- How does the new sampling assumption influence your estimated rates?

Empirical Taxon Sampling

Unfortunately, *diversified* taxon sampling was derived under a strict mathematical concept that assumes all species that speciated before a given time were included and all other species were discarded (not sampled); see Figure 15.4. The *diversified* sampling strategy is clearly too restrictive to be realistic and can bias parameter estimates too (Höhna 2014). As another alternative we apply an *empirical* taxon sampling strategy that uses empirical information on the clade relationships and speciation times of the missing species. For example, in the primate phylogeny we know the crown age of Hominoidea and know that 19 additional speciation events must have happened between the crown age of the Hominoidea and the present time to accommodate the 19 missing species (see Figure 15.5). In fact, we can obtain for all larger groups the crown ages and the number of missing species and thus narrow down with empirical evidence the times when these missing speciation events have happened.

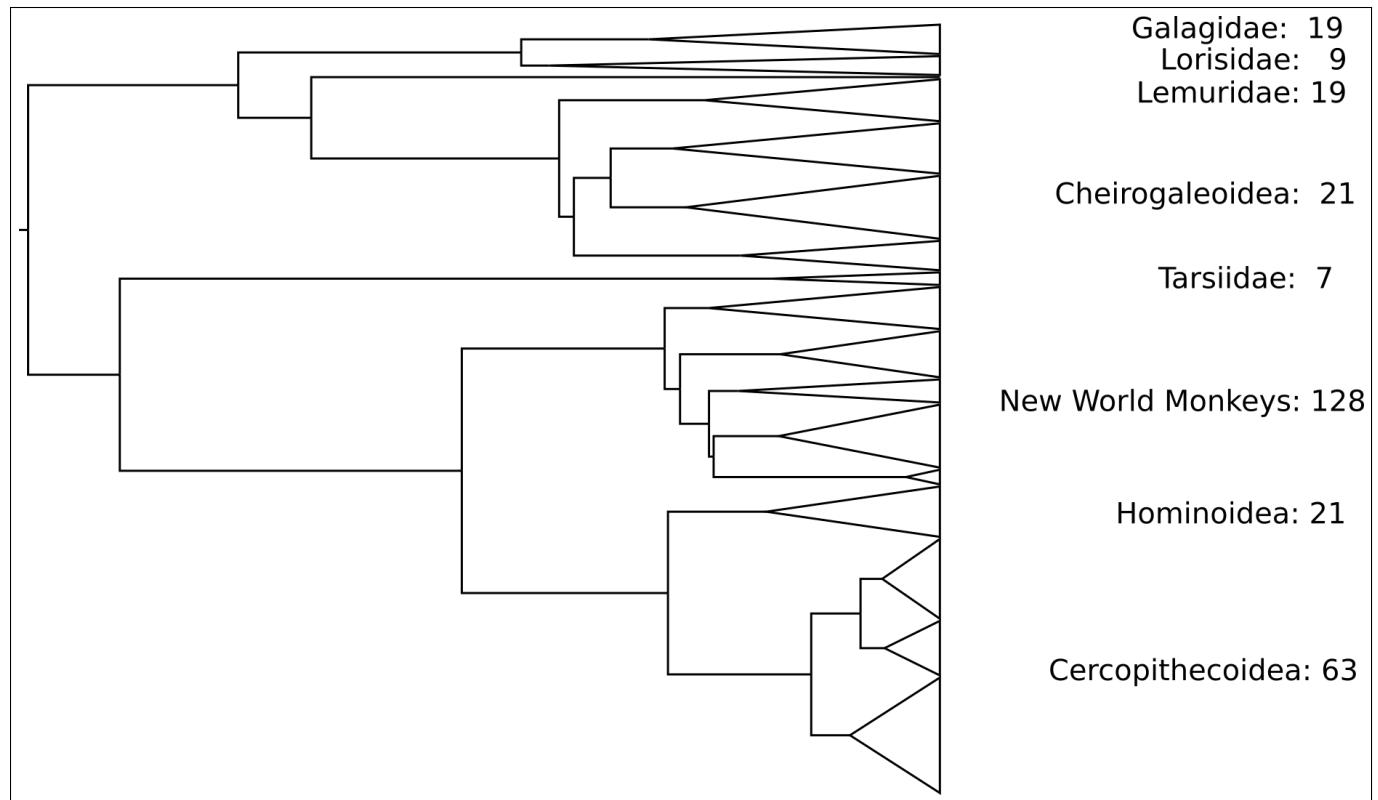


Figure 15.5: Cartoon of empirical taxon sampling. The triangle in the phylogeny depict clades with missing species. To illustrate the point we have written the names of higher taxa on the right with the number of species belonging to them. From this number of taxa in the clade we can compute how many species are missing per clade and which crown age the clade.

In your phylogeny you can count the number of species belonging to a given clade and thus compute how many species are missing for this clade. Then, you can pick two or more species to define the clade. These species will be used to compute the crown age. For example, we use *Pan paniscus* *Hylobates lar* to define the *Hominoidea* clade. If we would have *Homo sapiens* sampled as well then we could additionally include it in the clade but we could not leave out *Hylobates lar*.

In Rev we specify several clades and give the number of missing species.

```

Galagidae      = clade("Galago_senegalensis", "Otolemur_crassicaudatus", missing=17)
Lorisidae      = clade("Perodicticus_potto", "Loris_tardigradus", "Nycticebus_coucang", missing=6)
Cheirogaleoidea = clade("Cheirogaleus_major", "Microcebus_murinus", missing=19)
Lemuridae      = clade("Lemur_catta", "Varecia_variegata_variegata", missing=17)
Lemuriformes   = clade(Lemuridae, Cheirogaleoidea, missing=29)
Atelidae_Aotidae = clade("Alouatta_palliata", "Aotus_trivirgatus", missing=30)
NWM            = clade(Atelidae_Aotidae, "Callicebus_donacophilus", "Saimiri_sciureus", "Cebus_albifrons", missing=93)
Hominoidea     = clade("Pan_paniscus", "Hylobates_lar", missing=19)
Cercopithecoidea = clade("Colobus_guereza", "Macaca_mulatta", "Chlorocebus_aethiops", missing=60)

```

Next, we combine all clades into a single vector for later use.

```

missing_species_per_clade = v(Galagidae, Lorisidae, Cheirogaleoidea, Lemuridae,
                               Lemuriformes, Atelidae_Aotidae, NWM, Hominoidea, Cercopithecoidea)

```

In the birth-death model we include these missing speciation events by integrating over the known interval when these speciation events have happened (between the crown age and the present). This integral of the probability density of a speciation event is exactly the same as one minus the cumulative distribution function of a speciation event,

$$F(t|N(t_1) = 1, t_1 \leq t \leq T) = 1 - \frac{1 - P(N(T) > 0 | N(t) = 1) \exp(r(t, T))}{1 - P(N(T) > 0 | N(t_1) = 1) \exp(r(t_1, T))} \quad (15.1)$$

which was previously derived by Höhna (2014; Equation (6)) (see also Yang and Rannala (1997; Equation (3)) for constant rates and Höhna (2013; Equation (8))).

Let us define \mathbb{K} as the set of missing species and $|\mathbb{K}|$ the number of clades with missing species. In our example we have $|\mathbb{K}| = 9$ clades. Additionally, we define c_i as the time of most recent common ancestor of the i^{th} clade.

Then, the joint probability density of the sampled reconstructed tree and the empirically informed missing speciation times is

$$\begin{aligned} f(\Psi, \mathbb{K} | N(t_1=0)=2, S(2, t_1=0, T)) &= f(\Psi | N(t_1=0)=2, S(2, t_1=0, T)) \\ &\times \prod_{i=1}^{|\mathbb{K}|} (1 - F(t|N(c_i) = 1, c_i \leq t \leq T))^{k_i} \end{aligned} \quad (15.2)$$

Equation (15.2) is actually proportional to the original equation under the birth-death process times the probabilities of the missing species. There are two things to consider when specifying empirical taxon sampling in **RevBayes**. First, we set the “traditional” sampling fraction to one (**rho=1.0**). Second, we provide the clades with missing species as an argument of the birth-death model (**incompleteClades=missing_species_per_c**

```
timetree ~ dnEpisodicBirthDeath(rootAge=T.rootAge(), lambdaRates=speciation,
    lambdaTimes=interval_times, muRates=extinction, muTimes=interval_times, rho=1.0,
    taxa=taxa, incompleteClades=missing_species_per_clade, condition="time")
```

These are the only necessary changes to perform a diversification rate analysis under *empirical* taxon sampling.

Exercise 3

- Copy the Rev script `mcmc_uniform.Rev` and name it `mcmc_empirical.Rev`.
- Make the changes so that you assume now *empirical* taxon sampling.
- Change the file names in the monitors from `uniform` to `empirical`.
- Run the analysis and plot the diversification rates.
- How does the new sampling assumption influence your estimated rates?

Bibliography

- Cusimano, N. and S. Renner. 2010. Slowdowns in diversification rates from real phylogenies may not be real. *Systematic Biology* 59:458.
- Cusimano, N., T. Stadler, and S. S. Renner. 2012. A new method for handling missing species in diversification analysis applicable to randomly or nonrandomly sampled phylogenies. *Systematic Biology* 61:785–792.
- Höhna, S. 2013. Fast simulation of reconstructed phylogenies under global time-dependent birth-death processes. *Bioinformatics* 29:1367–1374.
- Höhna, S. 2014. Likelihood Inference of Non-Constant Diversification Rates with Incomplete Taxon Sampling. *PLoS One* 9:e84184.
- Höhna, S. 2015. The time-dependent reconstructed evolutionary process with a key-role for mass-extinction events. *Journal of Theoretical Biology* 380:321–331.
- Höhna, S., T. A. Heath, B. Boussau, M. J. Landis, F. Ronquist, and J. P. Huelsenbeck. 2014. Probabilistic graphical model representation in phylogenetics. *Systematic Biology* 63:753–771.
- Höhna, S., M. J. Landis, T. A. Heath, B. Boussau, N. Lartillot, B. R. Moore, J. P. Huelsenbeck, and F. Ronquist. 2016. RevBayes: Bayesian phylogenetic inference using graphical models and an interactive model-specification language. *Systematic Biology* 65:726–736.
- Höhna, S., T. Stadler, F. Ronquist, and T. Britton. 2011. Inferring speciation and extinction rates under different species sampling schemes. *Molecular Biology and Evolution* 28:2577–2589.
- May, M. R., S. Höhna, and B. R. Moore. 2016. A Bayesian Approach for Detecting the Impact of Mass-Extinction Events on Molecular Phylogenies When Rates of Lineage Diversification May Vary. *Methods in Ecology and Evolution* 7:947–959.

- Nee, S., R. M. May, and P. H. Harvey. 1994. The Reconstructed Evolutionary Process. *Philosophical Transactions: Biological Sciences* 344:305–311.
- Springer, M. S., R. W. Meredith, J. Gatesy, C. A. Emerling, J. Park, D. L. Rabosky, T. Stadler, C. Steiner, O. A. Ryder, J. E. Janečka, et al. 2012. Macroevolutionary dynamics and historical biogeography of primate diversification inferred from a species supermatrix. *PLoS One* 7:e49521.
- Stadler, T. 2009. On incomplete sampling under birth-death models and connections to the sampling-based coalescent. *Journal of Theoretical Biology* 261:58–66.
- Yang, Z. and B. Rannala. 1997. Bayesian phylogenetic inference using DNA sequences: a Markov Chain Monte Carlo Method. *Molecular Biology and Evolution* 14:717–724.

Chapter 16

Branch-Specific Diversification Rate Estimation

Estimating Branch-Specific Speciation & Extinction Rates

Outline

This tutorial describes how to specify a branch-specific branching-process models in RevBayes; a birth-death process where diversification rates vary among branches, similar to [Rabosky \(2014\)](#). The probabilistic graphical model is given for each component of this tutorial. The goal is to obtain estimate of branch-specific diversification rates using Markov chain Monte Carlo (MCMC).

Requirements

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

- RB_Getting_Started
- RB_Basics_Tutorial
- RB_BayesFactor_Tutorial
- RB_BasicDiversificationRate_Tutorial

Note that the RB_Basics_Tutorial introduces the basic syntax of Rev but does not cover any phylogenetic models. You may skip the RB_Basics_Tutorial if you have some familiarity with R. The RB_BayesFactor_Tutorial introduces Bayesian model selection by means of Bayes factors, which can be skipped by readers familiar with Bayesian model selection. We tried to keep this tutorial very basic and introduce all the language concepts and theory on the way. You may only need the RB_Basics_Tutorial for a more in-depth discussion of concepts in Rev.

Data and files

We provide the data file(s) which we will use in this tutorial. You may want to use your own data instead. In the **data** folder, you will find the following files

- **primates_tree.nex**: Dated primates phylogeny including 233 out of 367 species from [Magnuson-Ford and Otto \(2012\)](#).

→ Open the tree **data/primates_tree.nex** in FigTree.

Branch-Specific Birth-Death Model

The basic idea behind the model is that speciation and extinction rates are allowed to vary across branches of the tree (see Figure 16.1). Unfortunately, it is not possible to model rates drawn from a continuous distribution directly, as done for example in BAMM, because in that case one needs to integrate over any number of possible rate shifts, any time of these shifts and most importantly over all possible new rates. This is unfeasible to do and failure to do so has been shown to make parameter estimates unreliable ([Moore et al. 2016](#)).

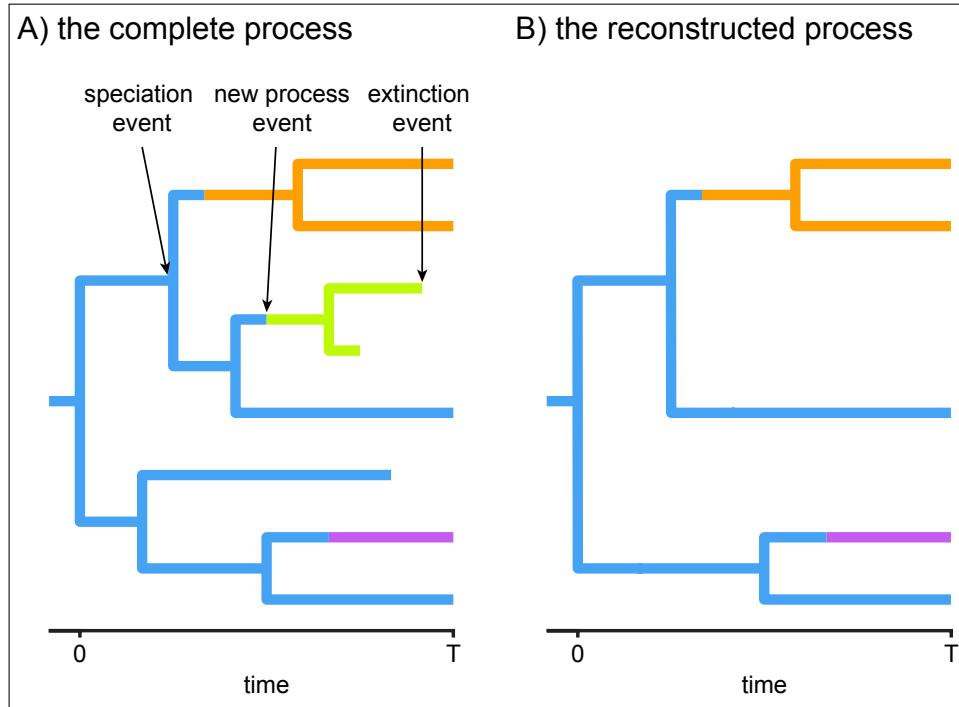


Figure 16.1: Cartoon of a branch-specific birth-death process. On the left we see the full process. On the right we only see the branches of the reconstructed tree, thus missing one rate-shift event.

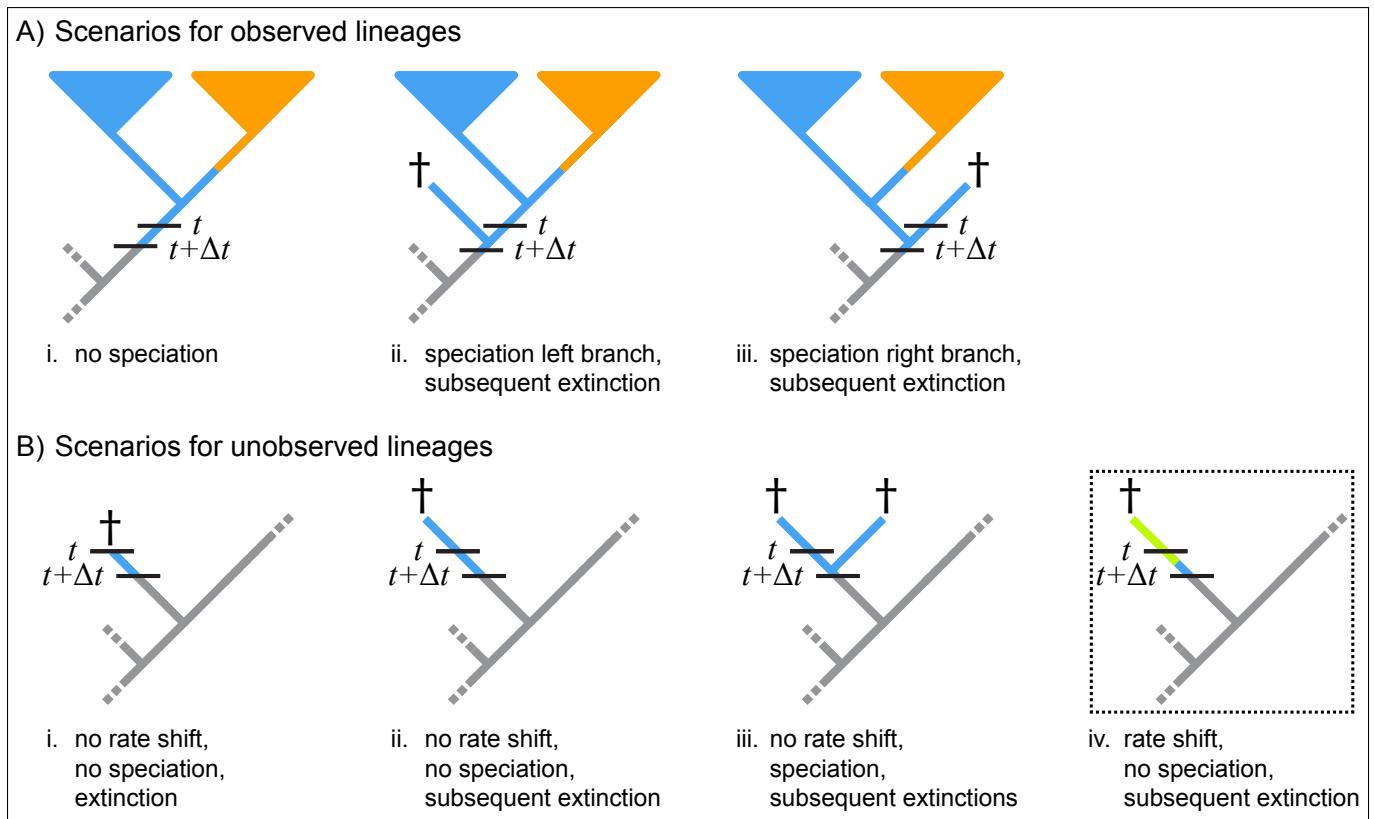


Figure 16.2: Cartoon of the likelihood computation using numerical integration.

Here we adopt an approach using (few) discrete rate categories instead. This allows us to numerically integrate over all possible rate categories using a system of differential equation originally described by [Maddison et al. \(2007\)](#) (see also [FitzJohn et al. \(2009\)](#) and [FitzJohn \(2010\)](#)). The numerical procedure breaks time into very small time intervals and sums over all possible events occurring in that interval (see Figure 16.2).

- You don't need to worry about any of the technical details. It is important for you to realize that this model assumes that new rates at a rate-shift event are drawn from a given (discrete) set of rates.

In RevBayes we have two implementations (*i.e.*, distributions) for modeling a branch-specific birth-death process. The first distribution is the `dnBirthDeathMultiRate` (or its alias `dnMRBDP`) and the second is the `dnHeterogeneousBirthDeath` (or its alias `dnHBDP`). We have designed this tutorial so that each section can be read independently although we recommend that you work through both of them. You will find that some parts are redundant, which is intentional to emphasize the similarities between the analysis but also to make the sections independent.

Testing for Branch-Specific-Diversification Rates

In this first exercise we are interested in knowing if there is diversification-rate variation among branches for our study tree. That is, we want to see if we can reject a constant rate birth-death process. Therefore, we don't focus on branch-specific parameter estimates but instead on the marginal likelihood estimation for model testing.

- We assume that you have completed the `RB_BasicDiversificationRate_Tutorial` to estimate the marginal likelihood under a constant-rate birth-death process. If you haven't done so, then you should go back and do this now!

Read the tree

Begin by reading in the observed tree.

```
observed_phylogeny <- readTrees("data/primates_tree.nex") [1]
```

From this tree, we can get some helpful variables:

```
taxa <- observed_phylogeny.taxa()
root <- observed_phylogeny.rootAge()
tree_length <- observed_phylogeny.treeLength()
```

Additionally, we can initialize an iterator variable for our vector of moves and monitors:

```
mvi = 0
mni = 0
```

Finally, we create a helper variable that specifies the number of discrete rate categories, another helper variable for the expected number of rate-shift events, the total number of species, and the variation in rates.

```
NUM_RATE_CATEGORIES = 4
EXPECTED_NUM_EVENTS = 2
NUM_TOTAL_SPECIES = 367
H = 0.587405
```

Using these variables we can easily change our script, for example, to use more or fewer categories and test the impact. For example, setting `NUM_RATE_CATEGORIES = 1` gives the constant rate birth-death process.

Specifying the model

Priors on rates

Instead of using a continuous probability distribution we will use a discrete approximation of the distribution, as done for modeling rate variation across sites (Yang 1994) and for modeling relaxed molecular clocks (Drummond et al. 2006). That means, we assume that the speciation rates are drawn from one of the N quantiles of the lognormal distribution. For this we will use the function `fnDiscretizeDistribution` which takes in a distribution as its first argument and the number of quantiles as the second argument. The return value is a vector of quantiles. We use it as a deterministic variable and every time the parameters of the base distribution (*i.e.*, the lognormal distribution in our case) change the quantiles will update automatically as well. Thus we only need to specify parameters for our base distribution, the lognormal distribution. We choose a stochastic variable for the mean parameter of the lognormal distribution drawn from yet another lognormal prior distribution. We fix the prior mean on this mean speciation rate on our expected diversification rate, which is $\ln(\ln(\frac{\#Taxa}{2})/age)$. Remember that the median of a lognormal distribution is equal to the exponential of the mean parameter. This is why we used a log-transform of the actual mean. This prior density is analogous to the prior on the speciation-rate parameter in the constant-rate birth-death process.

```
speciation_prior_mean <- ln( ln(NUM_TOTAL_SPECIES/2.0) / root )
speciation_mean ~ dnLognormal(mean=speciation_prior_mean, sd=H)
moves[++mvi] = mvScale(speciation_mean,lambda=1,tune=true,weight=5)
```

Additionally, we choose a fixed standard deviation of $2 * H$ ($0.587405 * 2$) for the speciation rates because it represents two orders of magnitude variance in the rate categories.

```
speciation_sd <- 2*H
speciation_categories := fnDiscretizeDistribution( dnLognormal(ln(speciation_mean),
    speciation_sd), NUM_RATE_CATEGORIES )
```

We also need discretized extinction-rate categories. We are completely free to choose how we construct these rate categories. For example, we could choose a similar discretization of a lognormal distribution

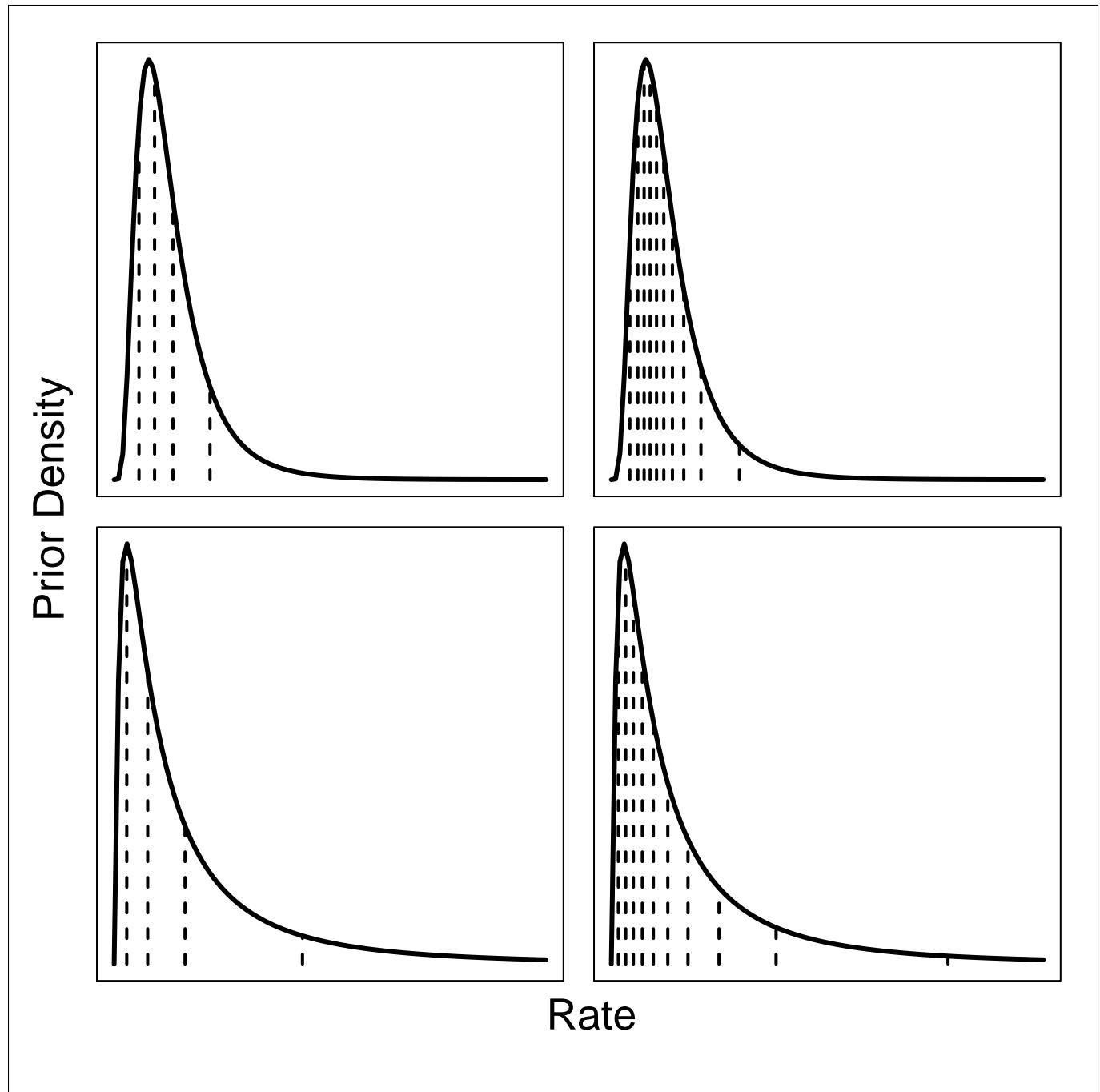


Figure 16.3: Discretization of a lognormal distribution. The two left figures have 4 rate categories and the two right plots have 10 rate categories. The top plots have the 95% probability interval spanning one order of magnitude (**sd** = 0.587405) and the bottom plots have the 95% probability interval spanning two orders of magnitude (**sd** = 2 * 0.587405)

using its quantiles to provide different extinction-rate categories. For simplicity, this is how we specify the current model. Alternatively, we could assume that each rate category has the same extinction rate.

```
extinction_prior_mean <- ln( ln(NUM_TOTAL_SPECIES/2.0) / root )
extinction_mean ~ dnLognormal(mean=extinction_prior_mean, sd=2*H)
moves[++mvi] = mvScale(extinction_mean, lambda=1.0, tune=true, weight=3.0)
```

As with the speciation rate, we discretize the lognormal distribution into a finite number of rate categories.

```
extinction_categories := fnDiscretizeDistribution( dnLognormal(ln(extinction_mean), H,
    , NUM_RATE_CATEGORIES )
```

Now, we must create a vector that contains each combination of speciation- and extinction-rates. This allows the rate of speciation to change without changing the rate of extinction and vice versa. The resulting vector should be N^2 elements long. We call these the ‘paired’ rate categories.

```
k = 1
for(i in 1:NUM_RATE_CATEGORIES) {
    for(j in 1:NUM_RATE_CATEGORIES) {
        speciation[k] := speciation_categories[i]
        extinction[k++] := extinction_categories[j]
    }
}
```

Next, we need a rate parameter for the rate-shifts events. We do not have much prior information about this rate but we can provide some realistic ranges. For example, we can specify a mean rate so that the resulting number of expected rate-shift events is 2 (as specified in our global variable `EXPECTED_NUM_EVENTS`). Furthermore, we can say that the 95% prior ranges exactly one order of magnitude. We achieve all this by specifying a lognormal prior distribution with mean `ln(EXPECTED_NUM_EVENTS/tree_length)` and standard deviation of `H`. Remember that this is only possible if the tree is known and not estimated simultaneously because only if the tree is do we also know the tree length. As usual for rate parameter, we apply a scaling move to the `event_rate` variable.

```
event_rate ~ dnLognormal( ln( EXPECTED_NUM_EVENTS/tree_length ), H )
moves[++mvi] = mvScale(event_rate, lambda=1, tune=true, weight=5)
```

Additionally, we need a rate-matrix parameter providing the relative rates between paired rate categories. In this case we simply use equal rates between each rate category; and thus use the Jukes-Cantor rate matrix. You could, for example, also use an ordered rate matrix where the process needs to go through rate 2 before going to rate 3 when starting in rate 1.

```
rate_matrix <- fnJC( NUM_RATE_CATEGORIES * NUM_RATE_CATEGORIES )
```

Furthermore, we need prior probabilities for the process being in either paired rate category at the root. Given our lack of prior knowledge we create a flat prior distribution giving each rate category equal weight. We do this by create a constant variable using the simplex function.

```
rate_category_prior <- simplex( rep(1, NUM_RATE_CATEGORIES * NUM_RATE_CATEGORIES) )
```

Incomplete Taxon Sampling

We know that we have sampled 233 out of 367 living primate species. To account for this we can set the sampling parameter as a constant node with a value of 233 / 367.

```
rho <- observed_phylogeny.nTips() / NUM_TOTAL_SPECIES
```

Root age

The birth-death process requires a parameter for the root age. In this exercise we use a fix tree and thus we know the age of the tree. Hence, we can get the value for the root from the [Magnuson-Ford and Otto \(2012\)](#) tree. This is done using our global variable `root` defined above and nothing else has to be done here.

The time tree

Now we have all of the parameters we need to specify the full episodic birth-death model. We initialize the stochastic node representing the time tree.

```
timetree ~ dnMRBDP(lambda=speciation, mu=extinction, Q=rate_matrix, rootAge=root, rho=rho, pi=rate_category_prior, delta=event_rate, taxa=taxa)
```

And then we attach data to it.

```
timetree.clamp(observed_phylogeny)
```

Finally, we create a workspace object of our whole model using the `model()` function.

```
mymodel = model(speciation)
```

The `model()` function traversed all of the connections and found all of the nodes we specified.

Running a marginal likelihood estimation

Specifying Monitors

For the marginal likelihood analysis we don't necessarily need monitors because we are not going to look into the samples. However, as good practice we still define our two standard monitors: the model monitor and a screen monitor

```
monitors[++mni] = mnModel(filename="output/primates_MRBD.log", printgen=10, separator =
    TAB)
monitors[++mni] = mnScreen(printgen=10, diversification_mean, turnover)
```

Initializing and Running the MCMC Simulation

- If you don't feel comfortable with Bayesian model selection anymore, then have a look at the [RB_BayesFactor_Tutorial](#) again.

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/MRBD_powp.out", cats=100)
```

We can start the power posterior by first burning in the chain and discarding the first 5000 states.

```
pow_p.burnin(generations=5000, tuningInterval=200)
```

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

```
pow_p.run(generations=2000)
```

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/MRBD_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 16.1.

```
ss.marginal()
```

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

```
ps = pathSampler(file="output/MRBD_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 16.1.

```
ps.marginal()
```

- The Rev file for performing this analysis: [ml_MRBD.Rev](#).

Exercise 1

- Enter the marginal likelihood estimate from the previous exercise on the constant-rate birth-death process in the table below.
- Compute the marginal likelihood under the 2-rate model, *i.e.*, set the `NUM_Rate_CATEGORIES` variable to 2.
- Repeat the estimation of the marginal likelihoods with other number of rate categories to fill out the table.
- What is the most supported model? Can we reject the constant-rate birth-death process?

Estimating Branch-Specific Diversification Rates

In this second analysis we are interested in estimating the branch-specific diversification rates. We are going to use a very similar model to the one described in the previous section. However, now we are going to use the `dnHBDP` distribution instead which will require some slightly different parameterization and moves. The main difference, as mentioned above, is that the `dnHBDP` uses a data-augmentation scheme to sample the locations and parameters of rate-shift events across branches of the tree.

Read the tree

Begin by reading in the observed tree.

```
observed_phylogeny <- readTrees("data/primates_tree.nex") [1]
```

Table 16.1: Marginal likelihoods and Bayes factors*.

Estimate	<i>Stepping-stone</i>	<i>Path sampling</i>
Marginal likelihood constant-rate ($N = 1$)		
Marginal likelihood two rate ($N = 2$)		
Marginal likelihood four rate ($N = 4$)		
Marginal likelihood six rate ($N = 6$)		
Marginal likelihood eight rate ($N = 8$)		
Marginal likelihood ten rate ($N = 10$)		
Supported model?		

*you can edit this table

From this tree, we can get some helpful variables:

```
taxa <- observed_phylogeny.taxa()
root <- observed_phylogeny.rootAge()
tree_length <- observed_phylogeny.treeLength()
```

Additionally, we can initialize an iterator variable for our vector of moves:

```
mvi = 0
mni = 0
```

Finally, we create a helper variable that specifies the number of discrete rate categories, another helper variable for the expected number of rate-shift events, the total number of species, and the variation in rates.

```
NUM_RATE_CATEGORIES = 4
EXPECTED_NUM_EVENTS = 2
NUM_TOTAL_SPECIES = 367
H = 0.587405
```

Using these variables we can easily change our script, for example, to use more or fewer categories and test the impact.

Specifying the model

Priors on rates

Similar to the previous section, we will set up the rate categories using the exact same model and Rev syntax. Thus, we first create our hyper-prior on the mean speciation rate, which is drawn from a lognormal distribution.

```
speciation_prior_mean <- ln( ln(NUM_TOTAL_SPECIES/2.0) / root_age )
speciation_mean ~ dnLognormal(mean=speciation_prior_mean, sd=H)
moves[++mvi] = mvScale(speciation_mean,lambda=1,tune=true,weight=5)
```

Additionally, we choose a fixed standard deviation of $H * 2$ for the speciation rates because it represents two orders of magnitude variance in the rate categories.

```
speciation_sd <- H*2
speciation_categories := fnDiscretizeDistribution( dnLognormal(ln(speciation_mean),
    speciation_sd), NUM_RATE_CATEGORIES )
```

We define the prior on the extinction rate in the same way as we did for the speciation rate, with the only difference that we allow for two orders of magnitude of uncertainty.

```
extinction_prior_mean <- ln( ln(NUM_TOTAL_SPECIES/2.0) / root_age )
extinction_mean ~ dnLognormal(mean=extinction_prior_mean, sd=H*2)
moves[++mvi] = mvScale(extinction_mean,lambda=1.0,tune=true,weight=3.0)
```

As with the speciation rate, we discretize the lognormal distribution into a finite number of rate categories.

```
extinction_categories := fnDiscretizeDistribution( dnLognormal(ln(extinction_mean), H
    , NUM_RATE_CATEGORIES )
```

Now, we must create a vector that contains each combination of speciation- and extinction-rates. This allows the rate of speciation to change without changing the rate of extinction and vice versa. The resulting vector should be N^2 elements long. We call these the ‘paired’ rate categories.

```
k = 1
for(i in 1:NUM_RATE_CATEGORIES) {
    for(j in 1:NUM_RATE_CATEGORIES) {
        speciation[k] := speciation_categories[i]
        extinction[k++] := extinction_categories[j]
    }
}
```

Next, we need a rate parameter for the rate-shifts events. We do not have much prior information about this rate but we can provide some realistic ranges. For example, we can specify a mean rate so that the resulting number of expected rate-shift events is 2 (as specified in our global variable `EXPECTED_NUM_EVENTS`). Furthermore, we can say that the 95% prior ranges exactly one order of magnitude. We achieve all this by specifying a lognormal prior distribution with mean `ln(EXPECTED_NUM_EVENTS/tree_length)` and standard deviation of `H`. Remember that this is only possible if the tree is known and not estimated simultaneously because only if the tree is do we also know the tree length. As usual for rate parameter, we apply a scaling move to the `event_rate` variable.

```
event_rate ~ dnLognormal( ln( EXPECTED_NUM_EVENTS/tree_length ), H )
moves[++mvi] = mvScale(event_rate,lambda=1,tune=true,weight=5)
```

Additionally, we need a parameter for the category of the process at root. We use a uniform prior distribution on the indices 1 to N^2 since we do not have any prior information in which rate category the process is at the root. The move for this random variable is a random integer walk because the random variable is defined only on the indices (*i.e.*, with real number).

```
root_category ~ dnUniformNatural(1,NUM_RATE_CATEGORIES * NUM_RATE_CATEGORIES)
moves[++mvi] = mvRandomIntegerWalk(root_category,weight=1)
```

Incomplete Taxon Sampling

We know that we have sampled 233 out of 367 living primate species. To account for this we can set the sampling parameter as a constant node with a value of 233 / 367.

```
rho <- observed_phylogeny.n tips() / NUM_TOTAL_SPECIES
```

Root age

The birth-death process requires a parameter for the root age. In this exercise we use a fix tree and thus we know the age of the tree. Hence, we can get the value for the root from the [Magnuson-Ford and Otto \(2012\)](#) tree. This is done using our global variable `root` defined above and nothing else has to be done here.

The time tree

Now we have all of the parameters we need to specify the full branch-specific birth-death model. We initialize the stochastic node representing the time tree.

```
timetree ~ dnHBDP(lambda=speciation, mu=extinction, rootAge=root, rho=rho, rootState=
root_category, delta=event_rate, taxa=taxa )
```

And then we attach data to it.

```
timetree.clamp(observed_phylogeny)
```

This specific implementation of the branch-specific birth-death process augments the tree with rate-shift events. In order to sample the number, the location, and the types of the rate-shift events, we have to apply special moves to the tree. These moves will not change the tree but only the augmented rate-shift events. We use a **mvBirthDeathEvent** to add and remove events, a **mvEventTimeBeta** move to change the time and location of the events, and a **mvDiscreteEventCategoryRandomWalk** to change the paired-rate category to which a rate-shift event belongs.

```
moves[++mvi] = mvBirthDeathEvent(timetree, weight=2)
moves[++mvi] = mvEventTimeBeta(timetree, weight=2)
moves[++mvi] = mvDiscreteEventCategoryRandomWalk(timetree, weight=2)
```

In this analysis, we are interested in the branch-specific diversification rates. So far we do not have any variables that directly give us the number of rate-shift events per branch or the rates per branch. Fortunately, we can construct deterministic variables and query these properties from the tree. These function are made available by the branch-specific birth-death process distribution.

```
num_events := timetree.numberEvents()
avg_lambda := timetree.averageSpeciationRate()
avg_mu    := timetree.averageExtinctionRate()
avg_net   := avg_lambda - avg_mu
avg_rel   := avg_mu / avg_lambda

total_num_events := sum( num_events )
```

Finally, we create a workspace object of our whole model using the **model()** function.

```
mymodel = model(speciation)
```

The **model()** function traversed all of the connections and found all of the nodes we specified.

Running an MCMC analysis

Specifying Monitors

For our MCMC analysis, we need to set up a vector of *monitors* to record the states of our Markov chain. First, we will initialize the model monitor using the **mnModel** function. This creates a new monitor variable that will output the states for all model parameters when passed into a MCMC function.

```
monitors[++mni] = mnModel(filename="output/primates_BSBD.log", printgen=10, separator =
    TAB)
```

Additionally, we create an extended-Newick monitor. The extended-Newick monitor writes the tree to a file and adds parameter values to the branches and/or nodes of the tree. We can thus print the tree with the average speciation and extinction rates, as well as the net diversification (speciation - extinction) and relative extinction (extinction / speciation) rates, for each branch into a file. We will need this file later to estimate and visualize the posterior distribution of the rates at the branches.

```
monitors[++mni] = mnExtNewick(filename="output/primates_BSBD.trees", isNodeParameter=
    FALSE, printgen=10, separator = TAB, tree=timetree, avg_lambda, avg_mu, avg_net,
    avg_rel)
```

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

```
monitors[++mni] = mnScreen(printgen=10, event_rate, mean_speciation, root_category,
    total_num_events)
```

Initializing and Running the MCMC Simulation

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

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

First, we will run a pre-burnin to tune the moves and to obtain starting values from the posterior distribution.

```
mymcmc.burnin(generations=1000, tuningInterval=200)
```

Now, run the MCMC:

```
mymcmc.run(generations=5000)
```

When the analysis is complete, you will have the monitored files in your output directory. You can then visualize the branch-specific rates by attaching them to the tree. This is actually done automatically in our `mapTree` function.

```
treetrace = readTreeTrace("output/primates_BSBD.trees", treetype="clock")
map_tree = mapTree(treetrace,"output/primates_BSBD_MAP.tree")
```

Now you can open the tree in **FigTree**.

→ The Rev file for performing this analysis: [mcmc_BSBD.Rev](#).

Exercise

- Run an MCMC simulation to estimate the posterior distribution of the speciation rate and extinction rate.
- Visualize the branch-specific rates in **FigTree**.
- Do you see evidence for rate decreases or increases? What is the general trend?
- Run the analysis using a different number of categories, *e.g.*, 2 or 6. How do the rates change?
- Modify the model by specifying a prior on the log-diversification and log-turnover rate and then estimate the diversification rates through time. Do you see any differences in the estimates?

Bibliography

- Drummond, A., S. Ho, M. Phillips, and A. Rambaut. 2006. Relaxed Phylogenetics and Dating with Confidence. *PLoS Biology* 4:e88.
- FitzJohn, R., W. Maddison, and S. Otto. 2009. Estimating trait-dependent speciation and extinction rates from incompletely resolved phylogenies. *Systematic Biology* 58:595–611.
- FitzJohn, R. G. 2010. Quantitative traits and diversification. *Systematic Biology* 59:619–633.
- Maddison, W., P. Midford, and S. Otto. 2007. Estimating a binary character's effect on speciation and extinction. *Systematic Biology* 56:701.
- Magnuson-Ford, K. and S. P. Otto. 2012. Linking the investigations of character evolution and species diversification. *The American Naturalist* 180:225–245.
- Moore, B. R., S. Höhna, M. R. May, B. Rannala, and J. P. Huelsenbeck. 2016. Critically evaluating the theory and performance of Bayesian analysis of macroevolutionary mixtures. *Proceedings of the National Academy of Sciences* 113:9569–9574.
- Rabosky, D. L. 2014. Automatic detection of key innovations, rate shifts, and diversity-dependence on phylogenetic trees. *PLoS One* 9:e89543.
- Yang, Z. 1994. Maximum likelihood phylogenetic estimation from DNA sequences with variable rates over sites: Approximate methods. *Journal of Molecular Evolution* 39:306–314.

Part VIII

Gene tree - Species tree estimation

Chapter 17

Overview

Overview: Gene tree-species tree models

Ever since ?, researchers have acknowledged that phylogenies reconstructed from homologous gene sequences could differ from species phylogenies. As molecular sequences accumulated, the link between gene trees and species trees started to be modeled. The first models were based on parsimony, and aimed for instance at reconciling a gene tree with a species tree by minimizing the number of events of gene duplication and gene loss. In the past dozen years, probabilistic models have been proposed to reconstruct gene trees and species trees in a rigorous statistical framework. Models and algorithms have quickly grown in complexity, to model biological processes with increasing realism, to accommodate several processes at the same time, or to handle genome-scale data sets. In this overview we will not detail these models, and we invite the interested reader to take a look at recent reviews (*e.g.*, (?)).

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 gene tree reconstruction. Such a mistake would produce an incorrect gene tree. Here we do not mean this kind of discord, but rather discord that comes from a real biological process that generates true gene histories that differ from true species histories. These processes include gene duplication, gene loss, gene transfer (used loosely here to also include reticulation, hybridization between species), and incomplete lineage sorting (Fig. 17.1). In this tutorial we focus on Incomplete lineage sorting, which will be discussed in more details in the following subsection.

Fig. 17.1 suggests that for all processes the gene tree can be seen as the product of a branching process operating inside the species tree. Indeed, all processes are modeled as some type of birth-death process running along the species tree. For duplication/loss models, birth correspond to gene duplication events, and death to gene loss events. Transfers can be added to the model by introducing another type of birth, with a child lineage appearing in another branch of the species tree. Incomplete lineage sorting is also modeled with a birth-death type of model, the coalescent. All these models can be made heterogeneous, for instance by allowing different sets of parameters for different branches of the species tree. This is useful to model differences in rates of duplication, loss or transfer among species, or to model different effective population sizes in a species tree. In RevBayes so far only models of incomplete lineage sorting have been implemented (models of duplication and loss and transfer will soon be added). Thanks to RevBayes modular design, there is quite a lot of flexibility in specifying the model, for instance by associating different parameters to different branches of the species tree, or by combining the gene tree-species tree model to other types of models, for instance models of trait evolution, or models of relaxed molecular clock.

Gene tree discordance is a problem for species tree reconstruction

There have been 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. More sophisticated approaches allow different partitions for different genes, but the main assumption at the heart of this approach is that all sites of all genes have evolved according to the same species tree. This assumption is often not correct because all the processes of discord presented above conspire to make gene trees

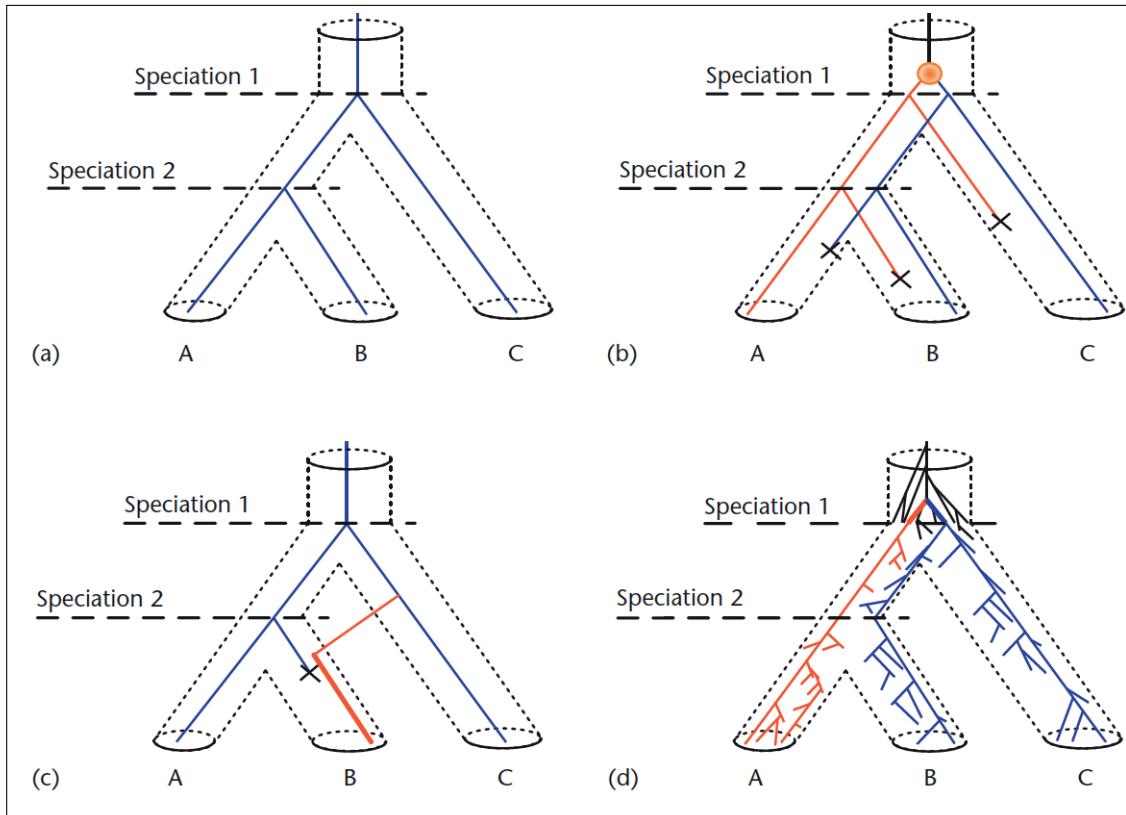


Figure 17.1: The processes of discord. The species tree is represented as a tubular structure. Gene trees are blue and red lines running along the species trees. a) A gene tree that perfectly matches the species tree. b) The gene tree and the species tree differ because of gene duplications and losses. c) The gene tree and the species tree differ because of gene transfer and gene loss. d) The gene tree and the species tree differ because of incomplete lineage sorting. [Replicated from Fig. 2 in [Boussau et al. \(2009\)](#).]

different from the species tree. In practice, this matters: simulation studies have found that in the presence of incomplete lineage sorting, in some particular areas of the parameter space, concatenation will often return an incorrect species tree ([Leaché and Rannala 2011](#)). Concatenation may also be a questionable approach in prokaryotic phylogenetics, where the quest for a tree of life has been difficult, to the point that some doubted that one could find a meaningful species tree representing vertical descent. Nonetheless, the concatenation approach may be fairly robust to lateral gene transfers, as it returns good species trees (arguably better than small subunit or large subunit rRNA trees) in a range of prokaryotic groups (?).

2. Supertree approaches differ from concatenation notably by discarding sequence information once individual gene trees have been built. Contrary to concatenation approaches that combine individual gene alignments, supertree approaches combine individual gene trees to obtain a species tree. Most supertree methods are not based on an explicit model of the processes causing discordance between gene trees and species tree (although there are exceptions, notably modelling incomplete lineage sorting, see below). Instead, they aim at finding a tree that would best describe the distribution of gene trees, according to some fairly arbitrary criterion. In practice, these methods have been found to provide reasonable results in many cases, but in simulations they are usually less accurate than concatenation.
3. Methods that rely on gene tree-species tree models appear very promising as they explicitly model

the processes of discord. The advantage of these models is that we account for processes that we know have taken a part in generating the data, thus possibly improving the accuracy and robustness of our inferences. Further, these models can be combined with *e.g.*, models of sequence evolution, models of co-evolution between gene trees, or models of trait evolution. However, these models are computationally challenging to use, because they require estimating jointly gene trees, species trees, and other parameters that entertain strong correlations. As a consequence, in many gene tree-species tree models, devising a well-mixing MCMC strategy can be problematic.

Bibliography

- Boussau, B., L. Guéguen, and M. Gouy. 2009. A mixture model and a hidden markov model to simultaneously detect recombination breakpoints and reconstruct phylogenies. Evolutionary bioinformatics online 5:67.
- Leaché, A. D. and B. Rannala. 2011. The accuracy of species tree estimation under simulation: a comparison of methods. Systematic Biology 60:126–137.

Chapter 18

Estimating species tree using gene concatenation

Concatenating genes to model species evolution

The simplest model to estimate a species tree when you have several genes is to concatenate your genes and to assume that all gene trees are exactly equal to the species tree. You thus assume that there is no discordance between gene trees in the hope that the combined information in all genes will provide a reliable signal for the species tree.

The exercises assume you have a working installation of RevBayes. In this introductory tutorial, we will apply the concatenated model to 10 gene alignments from 23 primate species. We will assume that:

- The species tree is drawn from a constant birth-death process.
- All genes share the same tree, that is, both the topology and branch lengths.
- Each gene has its own set of substitution model and clock parameters which are drawn from a shared prior distribution.
- Gene sequences are evolved according to an HKY model with gamma distributed rate variation among sites and a strict global clock.
- Here, we run an MCMC on this model, using data from 10 genes in 23 mammalian species.

Scripts are all placed in *tutorials/RB_GeneConcatenation_Tutorial/RevBayes_scripts/*.

1. Open RevBayes
2. Let's load all 10 gene alignments.

```
# read in each data matrix together which will create a vector of objects
data = readDiscreteCharacterData("data/merged.nex")

# Now we get some useful variables from the data. We need these later on.
num_loci = data.size()
# get the number of species
n_species <- data[1].ntaxa()
# get the taxon information (e.g. the taxon names)
taxa <- data[1].taxa()
```

```
n_branches <- 2 * n_species - 1 # number of branches in a rooted tree

# We set our move index
mi = 0
```

3. We specified a constant-rate birth-death process as our prior on the species tree. The birth-death process has a speciation and extinction rate as its parameters. We will use here a transformation and specify priors on the speciation rate and relative extinction rate. Additionally, we calibrate the tree by assuming that the crown age of primates is around 75 MYA. Thus, we specify a normal distribution with mean 75 and standard deviation 2.5 as the prior on the root age. Since the root age can only be a positive real number we truncate the normal distribution at 0.

```
# Specify a prior on the diversification and turnover rate
speciation ~ dnGamma(2,2)
relativeExtinction ~ dnBeta(1,1)

# Now transform the diversification and turnover rates into speciation and
# extinction rates
extinction := speciation * relativeExtinction

# Specify a prior on the root age (our informed guess is about ~75 mya)
# Note that we use a truncated normal distribution because the root age must be
# positive
root ~ dnNormal(mean=75,sd=2.5,min=0.0, max=Inf)

sampling_fraction <- 23 / 450 # we sampled 23 out of the ~ 450 primate species

# create some moves that change the stochastic variables
# Moves are sliding and scaling proposals
moves[++mvi] = mvSlide(diversification,delta=1,tune=true,weight=2)
moves[++mvi] = mvSlide(relativeExtinction,delta=1,tune=true,weight=2)
moves[++mvi] = mvScale(diversification,lambda=1,tune=true,weight=2)
moves[++mvi] = mvScale(relativeExtinction,lambda=1,tune=true,weight=2)
moves[++mvi] = mvSlide(root,delta=1,tune=true,weight=0.2)

# construct a variable for the tree drawn from a birth-death process
psi ~ dnBDP(lambda=speciation, mu=extinction, rootAge=root, rho=sampling_fraction,
taxa=taxa )

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

4. Now that we have a species tree, which we assume is shared exactly for all genes. That is, we assume each gene evolves under exactly the same tree, and thus each gene tree is equivalent to the species tree. Nevertheless, we assume that each gene evolves at a different rate and with its own substitution model parameters. Here we will assume for simplicity that every gene evolves under a global strict clock but has its own independent clock rate. We assume that the logarithm of the clock rate is uniformly distribution, thus we specify in effect a log-uniform prior distribution. This prior assumption means that we put the same prior probability on values of each magnitude, e.g., values between 0.0001 and 0.001 have the same prior probability as values between 10 and 100.

```
for ( i in 1:num_loci ) {
    log_clock_rate[i] ~ dnUniform(-8,4)
    clock_rate[i] := 10^log_clock_rate[i]

    moves[++mvi] = mvSlide(log_clock_rate[i], weight=1.0)
}
```

5. Next we need our model for the substitution process. Hence, we just need to define the substitution matrix. We use a single HKY matrix that will apply to all sites per gene. Additionally, we assume that sites evolve according to one of four possible rates, where each rate corresponds to a quantile from a gamma distribution.

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

    ##### specify the HKY substitution model applied uniformly to all sites of a
    ##### gene
    kappa[i] ~ dnLognormal(0,1)
    moves[++mvi] = mvScale(kappa[i],weight=1)

    pi_prior[i] <- v(1,1,1,1)
    pi[i] ~ dnDirichlet(pi_prior[i])
    moves[++mvi] = mvSimplexElementScale(pi[i],weight=2)

    ##### create a deterministic variable for the rate matrix
    Q[i] := fnHKY(kappa[i],pi[i])

    ##### create the rates to model the gamma distributed rate variation among
    ##### sites.
    alpha_prior[i] <- 0.05
    alpha[i] ~ dnExponential(alpha_prior[i])
    gamma_rates[i] := fnDiscretizeGamma(alpha[i], alpha[i], 4, false)

    # add moves for the stationary frequencies, exchangeability rates and the
    # shape parameter
    moves[++mvi] = mvScale(alpha[i],weight=2)
```

```
}
```

6. Finally, we can create our distribution for the character evolution. We will use the common **PhyloCTMC** distribution, which is a continuous time Markov process along a phylogenetic tree. We create a **seq** variable and attach/clamp each gene to one of the **seq** variables.

```
for ( i in 1:num_loci ) {
    # the sequence evolution model
    seq[i] ~ dnPhyloCTMC(tree=psi, Q=Q[i], branchRates=clock_rate[i], siteRates=
        gamma_rates[i], type="DNA")

    # attach the data
    seq[i].clamp(data[i])
}
```

7. Now we have defined all the bricks of the model, and create our model object from it.

```
# We get a handle on our model.
# We can use any node of our model as a handle, here we choose to use the topology
.

mymodel = model(psi)
```

8. Finally, we need to perform inference under the model, using the data.

```
# Monitors to check the progression of the program
monitors[1] = mnScreen(printgen=100, root)
monitors[2] = mnModel(filename="output/primates_concatenation_root_calibration",
    printgen=10, separator = TAB)
monitors[3] = mnFile(filename="output/primates_concatenation_root_calibration",
    printgen=10, separator = TAB, psi)

# Here we use a plain MCMC. You could also set nruns=2 for a replicated analysis
# or use mcmcmc with heated chains.
mymcmc = mcmc(mymodel, monitors, moves)

# This should be sufficient to obtain enough MCMC samples
mymcmc.burnin(generations=3000,tuningInterval=100)
mymcmc.run(generations=10000)
```

9. Now we can perform some post-run analyses.

```
# Now, we will analyze the tree output.  
# Let us start by reading in the tree trace  
treetrace = readTreeTrace("output/primates_concatenation_root_calibration",  
    treetype="clock")  
# and get the summary of the tree trace  
treetrace.summarize()  
  
mapTree(treetrace,"output/primates_concatenation_root_calibration")
```

Chapter 19

Multispecies Coalescent

Modeling incomplete lineage sorting: the multispecies coalescent

Incomplete lineage sorting is a population-level process. In a species, at a given time, there are several alleles for a given locus in the genome. These alleles have their own history, they diverged from each other at various times in the past. This history can differ from the species history, because several alleles can persist through a speciation event, and because, without selective effects, the sorting of alleles during a speciation event is random and can result in a tree that differs from the species tree (Fig. 17.1d). In all cases, incongruence between the gene tree and the species tree occurs when alleles persist over the course of several speciation events. When reconstructing a gene tree, one therefore gets the history of the alleles that have been sampled (at best), not the history of the species.

In 2003, Rannala and Yang proposed a powerful way to model the sorting of alleles along a phylogeny of several species (?), the multispecies coalescent (Fig. 19.1). This model is at the origin of most model-based approaches to reconstruct gene and species trees (Edwards et al. 2007; Heled and Drummond 2010). The multispecies coalescent appropriately models the evolution of a population of alleles along a species tree. Along the species tree, it allows different branch lengths, in units of time, and also allows different effective population sizes. Computing the probability of a gene tree given a species tree and other parameters is quite easy. Basically it works by cutting the gene tree into independent species-specific subtrees, computing probabilities for each of those subtrees, and combining them all at the end to get the probability of the gene tree according to the multispecies coalescent, given the current parameter values. Cutting the gene tree into species-specific subtrees is quite easy, because we can use the dates of speciation events to identify parts of the gene trees that are before and after speciation events. The resulting subtrees are represented with the grey boxes in Fig. 19.1. In this figure, each subtree corresponds to one particular population, either extant or ancestral. Inside each subtree, given its length, the effective population size, and dates of coalescence (divergences of alleles), the coalescent model provides simple formulas for computing the probability of the gene subtree given other parameters. Because we consider that these subtree probabilities are all independent of one another, they are then multiplied to get the gene tree probability given current parameter values.

Two parameters associated to branches of the species tree have a direct impact on the expected amount of gene tree-species tree incongruence:

- **Time between speciations.** The more a branch length increases, the more the pool of alleles is expected to change. Alleles are therefore less likely to persist for several speciation events if the branches between these speciation events are long.
- **Effective population size between speciations.** In populations with small effective population sizes, chance events can cause large shifts in allele frequencies, and possibly disappearance of alleles. In large populations, because an allele is likely carried by a large number of individuals, its disappearance is less likely, the population of alleles is more stable. Alleles are therefore less likely to persist for several speciation events if the branches between these speciation events are characterized by small effective population sizes.

Overall, larger amounts of gene tree-species tree incongruence are expected in phylogenies characterized by short branches with large population sizes. A corollary of that is that larger amounts of gene tree-gene tree incongruence are expected as well. To measure the susceptibility of species phylogenies to generate incomplete lineage sorting, the concept of *coalescent time units* has been introduced. Coalescent time units are obtained when branch length λ , in number of generations, is divided by effective population size N_e .

As a consequence, in a species tree whose branches are expressed in coalescent time units, a branch length of 1 *coalescent time unit* means a branch length of N_e generations. Once branch lengths on the species tree are measured in coalescent time units, it becomes easy to spot species trees that generate a lot of incongruence: those are short trees.

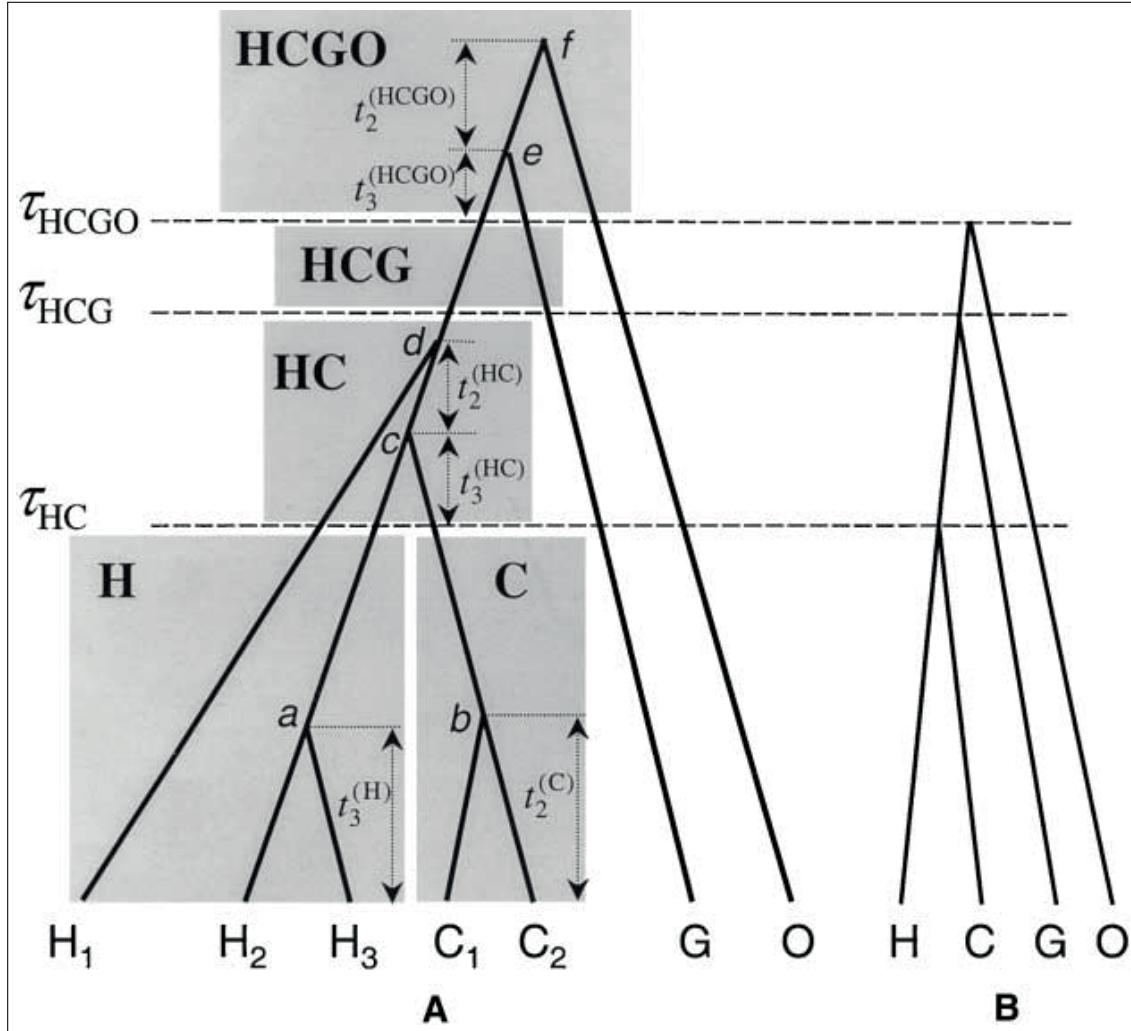


Figure 19.1: The multispecies coalescent. A) A gene tree, including 3 human alleles, 2 Chimp alleles, one Gorilla allele, and one Orang-outan allele. τ parameters are speciation times, t parameters are divergence time in the gene tree, the grey squares represent the ancestral populations, with their respective 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. In this introductory tutorial, we will apply the multispecies coalescent model to 10 gene alignments from 23 primate species. We will

specify the multispecies coalescent, with different effective population sizes for each branch of the species tree. We will assume that:

- The species tree is drawn from a constant birth-death process.
- Along the branches of the species tree, a multispecies coalescent process generates gene trees. Different effective population sizes are assigned to each branch of the species tree.
- Along each gene tree, gene sequences are evolved according to an HKY model with gamma distributed rate variation among sites and a strict global clock.
- Here, we run an MCMC on this model, using data from 10 genes in 23 mammalian species.

Scripts are all placed in *tutorials/RB_MultispeciesCoalescent_Tutorial/RevBayes_scripts/*.

1. Open RevBayes
2. Let's load all 10 gene alignments.

```

locus_names = ["COIII", "FGA", "GHRmeredith", "lrpprc_169", "npas3", "sim1", "tex2",
               "ttr", "zfy", "zic3"]

num_loci = locus_names.size()

# read in each data matrix separately
for ( i in 1:num_loci ) {
    data[i] <- readDiscreteCharacterData("data/" + locus_names[i] + ".fasta")
}

# Now we get some useful variables from the data. We need these later on.
primate_tree = readTrees("data/primates.tree")[1]
# get the number of species
n_species <- primate_tree.ntips()
# get the taxon information (e.g. the taxon names)
taxa <- primate_tree.taxon()
n_branches <- 2 * n_species - 1 # number of branches in a rooted tree

# We set our move index
mi = 0

```

3. We specified a constant-rate birth-death process as our prior on the species tree. The birth-death process has a speciation and extinction rate as its parameters. We will use here a transformation

and specify priors on the speciation rate and relative extinction rate. Additionally, we calibrate the tree by assuming that the crown age of primates is around 75 MYA. Thus, we specify a normal distribution with mean 75 and standard deviation 2.5 as the prior on the root age. Since the root age can only be a positive real number we truncate the normal distribution at 0.

```

# Specify a prior on the diversification and turnover rate
speciation ~ dnGamma(2,2)
relativeExtinction ~ dnBeta(1,1)

# Now transform the diversification and turnover rates into speciation and
extinction rates
extinction := speciation * relativeExtinction

# Specify a prior on the root age (our informed guess is about ~75 mya)
# Note that we use a truncated normal distribution because the root age must be
positive
root ~ dnNormal(mean=75,sd=2.5,min=0.0, max=Inf)

sampling_fraction <- 23 / 450 # we sampled 23 out of the ~ 450 primate species

# create some moves that change the stochastic variables
# Moves are sliding and scaling proposals
moves[++mvi] = mvSlide(diversification,delta=1,tune=true,weight=2)
moves[++mvi] = mvSlide(relativeExtinction,delta=1,tune=true,weight=2)
moves[++mvi] = mvScale(diversification,lambda=1,tune=true,weight=2)
moves[++mvi] = mvScale(relativeExtinction,lambda=1,tune=true,weight=2)
moves[++mvi] = mvSlide(root,delta=1,tune=true,weight=0.2)

# construct a variable for the tree drawn from a birth-death process
psi ~ dnBDP(lambda=speciation, mu=extinction, rootAge=root, rho=sampling_fraction,
taxa=taxa )

moves[++mvi] = mvNarrow(psi, weight=5.0)
moves[++mvi] = mvNNI(psi, weight=1.0)
moves[++mvi] = mvFNPR(psi, weight=3.0)
moves[++mvi] = mvGPR(psi, weight=3.0)
moves[++mvi] = mvSubtreeScale(psi, weight=3.0)
moves[++mvi] = mvNodeTimeSlideUniform(psi, weight=15.0)
moves[++mvi] = mvTreeNodeAgeSlide(psi, weight=50)

```

- Now that we have a species tree, we can specify the prior on the gene trees by using a multispecies coalescent process. First, we need to load in a map of the individual names to the species names. In that way we can attribute which individual belongs to which species. Have a look in one of these files, for example *primates_COIII_species_map.txt*. We will assume that each branch of the species tree, which represents a population, has its own population size. Thus, our prior is that each population size per branch is identically distributed from an exponential distribution with rate 0.1 (giving an

expectation of 10 and thus a relatively flat prior distribution). Note that we use fixed population sizes for the terminal branches because we have only a single individual per species and thus have no information about its population size. You could use other models for the population sizes too, if you wanted. For example we could assume that all branches have the same population size.

```

# We assume independent effective population size parameters for each branch of
# the species tree.
for (i in 1:n_species) {
    Ne[i] <- 10.0
}
for (i in (n_species+1):n_branches) {
    Ne[i] ~ dnExponential(0.1)
    moves[++mvi] = mvScale(Ne[i],1,true,1.0)
}

# We could instead assume a single effective population size for the entire
# species tree with the following two lines:
#Ne ~ dnGamma(shape=1.0,rate=1.0)
#moves[++mui] = mvScale(Ne,1,true,1.0)

for (i in 1:num_loci) {

    # We need to read in files providing the link between gene names and species
    # names
    taxon_map = readTaxonData("data/species_maps/primates_" + locus_names[i] +
        "_species_map.txt")

    # The gene tree from the multispecies coalescent process
    # Note that Ne is a vector of effective population sizes,
    # allowing 1 parameter per branch of the species tree.
    geneTree[i] ~ dnCoalMultiSpeciesConst(speciesTree=psi, Ne=Ne, taxa=taxon_map)

    # moves on the tree
    moves[++mvi] = mvNNI(geneTree[i], 5.0)
    moves[++mvi] = mvNarrow(geneTree[i], 5.0)
    moves[++mvi] = mvFNPR(geneTree[i], 3.0)
    moves[++mvi] = mvGPR(geneTree[i], 2.0)
    moves[++mvi] = mvSubtreeScale(geneTree[i], 5.0)
    moves[++mvi] = mvTreeScale(geneTree[i], 1.0, true, 3.0)
    moves[++mvi] = mvNodeTimeSlideUniform(geneTree[i], 20.0)
}

```

- Now we have gene trees, complete with branch lengths. The next element we need is a clock rate which transforms/scales the branch times into branch lengths that represent the expected number of substitutions. Here we will assume for simplicity that every gene evolves under a global strict clock but has its own independent clock rate. You can later look into the estimate to see how much the

clock rate estimates actually differ.

```
for ( i in 1:num_loci ) {
    log_clock_rate[i] ~ dnUniform(-4,1)
    clock_rate[i] := 10^log_clock_rate[i]

    moves[++mvi] = mvSlide(log_clock_rate[i], weight=1.0)
}
```

6. Next we need our model for the substitution process. Hence, we just need to define the substitution matrix. We use a single HKY matrix that will apply to all sites per gene. Additionally, we assume that sites evolve according to one of four possible rates, where each rate corresponds to a quantile from a gamma distribution.

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

    ##### specify the HKY substitution model applied uniformly to all sites of a
    ##### gene
    kappa[i] ~ dnLognormal(0,1)
    moves[++mvi] = mvScale(kappa[i],weight=1)

    pi_prior[i] <- v(1,1,1,1)
    pi[i] ~ dnDirichlet(pi_prior[i])
    moves[++mvi] = mvSimplexElementScale(pi[i],weight=2)

    ##### create a deterministic variable for the rate matrix
    Q[i] := fnHKY(kappa[i],pi[i])

    ##### create the rates to model the gamma distributed rate variation among
    ##### sites.
    alpha_prior[i] <- 0.05
    alpha[i] ~ dnExponential(alpha_prior[i])
    gamma_rates[i] := fnDiscretizeGamma(alpha[i], alpha[i], 4, false)

    # add moves for the stationary frequencies, exchangeability rates and the
    # shape parameter
    moves[++mvi] = mvScale(alpha[i],weight=2)

}
```

7. Finally, we can create our distribution for the character evolution. We will use the common **PhyloCTMC** distribution, which is a continuous time Markov process along a phylogenetic tree. We create a **seq** variable and attach/clamp each gene to one of the **seq** variables.

```

for ( i in 1:num_loci ) {
    # the sequence evolution model
    seq[i] ~ dnPhyloCTMC(tree=geneTree[i], Q=Q[i], branchRates=clock_rate[i],
                          siteRates=gamma_rates[i], type="DNA")

    # attach the data
    seq[i].clamp(data[i])
}

```

8. Now we have defined all the bricks of the model, and create our model object from it.

```

# We get a handle on our model.
# We can use any node of our model as a handle, here we choose to use the topology
.
mymodel = model(psi)

```

9. Finally, we need to perform inference under the model, using the data.

```

# Monitors to check the progression of the program
monitors[1] = mnScreen(printgen=100, root)
monitors[2] = mnModel(filename="output/primates_root_calibration.log",printgen=10,
                      separator = TAB)
monitors[3] = mnFile(filename="output/primates_root_calibration.trees",printgen
                     =10, separator = TAB, psi)
for ( i in 1:num_loci ) {
    # We add a monitor for each gene tree
    monitors[i+3] = mnFile(filename="output/primates_root_calibration_"+
                           locus_names[i] + ".trees",printgen=10, separator = TAB, geneTree[i])
}

# Here we use a plain MCMC. You could also set nruns=2 for a replicated analysis
# or use mcmc with heated chains.
mymcmc = mcmc(mymodel, monitors, moves)

# This should be sufficient to obtain enough MCMC samples
mymcmc.burnin(generations=3000,tuningInterval=100)
mymcmc.run(generations=10000)

```

10. Now we can perform some post-run analyses.

```

# Now, we will analyze the tree output.
# Let us start by reading in the tree trace

```

```
treetrace = readTreeTrace("output/primates_root_calibration.trees", treetype="clock")
# and get the summary of the tree trace
treetrace.summarize()

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

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 than 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?

Bibliography

Edwards, S. V., L. Liu, and D. K. Pearl. 2007. High-resolution species trees without concatenation. *Proceedings of the National Academy of Sciences* 104:5936–5941.

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

Part IX

Biogeography

Chapter 20

Dispersal, Extirpation and Cladogenesis (DEC)

Introduction

How did species come to live where they're found today? To answer this, we can leverage phylogenetic and geological information to model species distributions as the outcome of biogeographic processes. These natural processes require some additional considerations, such as how ranges are inherited following speciation events, how geological events might influence dispersal rates, and what factors affect rates of dispersal and extirpation. The major challenge of modeling range evolution is how to translate these natural processes into stochastic processes that remain tractable for inference. This tutorial provides a brief background in some of these models, then describes how to perform Bayesian inference of historical biogeography using RevBayes.

Dispersal-Extinction-Cladogenesis model

Range characters

Discrete biogeographical models typically rely on presence-absence data, where a species is observed or not observed across multiple discrete areas. For example, say there are three areas: A, B, and C. Say a species is present in areas A and C, then its range equals AC, which can also be encoded into the length-3 bit vector, 101. Bit vectors may also be transformed into (decimal) integers, *e.g.*, the binary number 101 equals the decimal number 5.

$$(\emptyset, A, B, AB, C, AC, BC, ABC) \Leftrightarrow (000, 100, 010, 110, 101, 011, 111) \Leftrightarrow (0, 1, 2, 3, 4, 5, 6, 7)$$

Decimal representation is rarely used in discussion, but it is useful to keep in mind when considering the total number of possible ranges for a species.

Modeling anagenic range evolution

How might we model the dynamics of species range evolution? In this section, we'll cover the Dispersal-Extinction-Cladogenesis model proposed by [Ree et al. \(2005\)](#). To begin, we'll focus on anagenesis: evolution that occurs between speciation events within lineages. Since we have discrete characters we'll use the continuous-time Markov chain, which allows us to compute transition probability of a character changing from i to j in time t through matrix exponentiation

$$\mathbf{P}_{i,j}(t) = [\exp \{\mathbf{Q}t\}]_{i,j},$$

where \mathbf{Q} is the instantaneous rate matrix defining the rates of change between all pairs of characters, and \mathbf{P} is the transition probability rate matrix. Remember, i and j represent different ranges, each of which is encoded as the set of areas occupied by the species. Exponentiation of the rate matrix is powerful because it integrates over all possible scenarios of character transitions that could occur during t so long as the chain begins in state i and ends in state j .

We can then encode \mathbf{Q} to reflect the allowable classes of range evolution events with biologically meaningful parameters. We'll take a simple model of range expansion (*e.g.* $BC \rightarrow ABC$) and range contraction (*e.g.* $BC \rightarrow C$). (Range expansion may also be referred to as dispersal or area gain and range contraction as extirpation, (local) extinction, or area loss.) The rates in the transition matrix for three areas might appear as

	\emptyset	A	B	AB	C	AC	BC	ABC
$\mathbf{Q} =$	\emptyset	—	0	0	0	0	0	0
	A	e_A	—	0	d_{AB}	0	d_{AC}	0
	B	e_B	0	—	d_{BA}	0	d_{BC}	0
	AB	0	e_A	e_B	—	0	0	$d_{AC} + d_{BC}$
	C	e_C	0	0	0	—	d_{CA}	d_{CB}
	AC	0	e_C	0	e_A	—	0	$d_{AB} + d_{CB}$
	BC	0	0	e_C	0	e_B	0	—
	ABC	0	0	0	e_C	0	e_B	e_A
,								

where $e = (e_A, e_B, e_C)$ are the (local) extinction rates per area, and $d = (d_{AB}, d_{AC}, d_{BC}, d_{CB}, d_{CA}, d_{BA})$ are the dispersal rates between areas. Notice that the sum of rates leaving state \emptyset is zero, meaning any species that loses all areas in its range remains permanently extinct.

?

For the three-area DEC rate matrix above, what is the rate of leaving state AC in terms of dispersal and extinction parameters?

Note the rate of more than one event occurring simultaneously is zero, so a range must expand twice by one area in order to expand by two areas.

?

What series of transition events might explain a lineage evolving from range ABC to range A ? From range AB to range C ?

Of course, this model can be specified for more than three areas.

?

Imagine a DEC rate matrix with four areas, $ABCD$. What would be the dispersal rate for $Q_{BC,BCD}$? How many states does a DEC rate matrix with four areas have? What is the relationship between the number of areas and the number of states under the DEC model?

Let's consider what happens to the size of \mathbf{Q} when the number of areas, N , becomes large. For three areas, \mathbf{Q} is size 8×8 . For ten areas, \mathbf{Q} is size $2^{10} \times 2^{10} = 1024 \times 1024$, which approaches the largest size matrices that can be exponentiated in a practical amount of time. For twenty areas, \mathbf{Q} is size $2^{20} \times 2^{20} \approx 10^6 \times 10^6$ and exponentiation is not viable.

Modeling cladogenic range evolution

Cladogenesis describes evolutionary change accompanying speciation. Daughter species are not expected to inherit their ancestral range identically in general. For each internal node in the reconstructed tree, one of two cladogenic events can occur: sympatry or allopatry. Say the range of a species is A the moment before speciation occurs at an internal phylogenetic node. Since the species range is size one, both daughter lineages necessarily inherit the ancestral species range (A). In DEC parlance, this is called a *narrow sympatry* event.

Now suppose the ancestral range is ABC . Under *subset sympatric cladogenesis*, one lineage identically inherits the ancestral species range, ABC , while the other lineage inherits only a single area, i.e. only A or B or C . For *widespread sympatric cladogenesis*, both lineages inherit the ancestral range, ABC . Under *allopatric cladogenesis*, the ancestral range is split evenly among daughter lineages, e.g. one lineage may inherit AB and the other inherits C .

For an excellent overview of described state transitions for cladogenetic events, see [Matzke \(2012\)](#).

[?] Given the state is AB before cladogenesis, and allowing subset sympatry, widespread sympatry, and allopatry, what are the 7 possible states in the daughter lineages after cladogenesis?

The probabilities of anagenic change along lineages must account for all combinations of starting states and ending states. For 3 areas, there are 8 states, and thus $8 \times 8 = 64$ probability terms for pairs of states. For cladogenic change, we need transition probabilities for all combinations of states before cladogenesis, after cladogenesis for the left lineage, and after cladogenesis for the right lineage. Like above, for three areas, there are 8 states, and $8 \times 8 \times 8 = 512$ cladogenic probability terms.

[?] For three areas, there are three narrow, four widespread, 18 subset sympatric events and 12 allopatric cladogenesis events. What proportion of terms in the cladogenesis matrix are zero?

The DEC model ignores speciation events hidden by extinction or incomplete taxon sampling. The probability of cladogenesis and local extinction events would ideally be linked to a birth-death process, as it is in the GeoSSE model (Goldberg et al. 2011). Unfortunately, since the numerical method for SSE models scale poorly, and DEC models remain the only option when the geography has more than two or three areas. For more than ten areas, data augmentation may be used to infer ancestral ranges, as described in Section 21.1.

The rest of this section will describe how to run a simple DEC analysis using RevBayes.

Specifying a simple DEC model

We'll use the primate dataset with 23 taxa. To keep the model simple, we'll discretize their ranges into just three areas: the New World (A), Africa (B), and Eurasia (C). For simplicity, we'll assume their phylogeny is time-calibrated, errorless, and fixed.

Create some **String** variables for file handling,

```
data_fn = "data/primates_bg_n3.tsv"
tree_fn = "data/primates.tree"
out_fn = "output/bg_same"
```

then read in our character data, using `type="Bitset"` to indicate ranges are encoded with bits, e.g. 011

```
data = readCharacterDataDelimited(file=data_fn, type="Bitset")
```

and our tree

```
psi <- readTrees(tree_fn)[1]
```

Next, compute the number of states from the number of areas

```
n_areas = 3
n_states = floor(2^n_areas)
```

Declare index variables for our move vectors for future use

```
mi = 0
```

Now, we'll begin to construct the rate matrix for anagenic events. First create a matrix, 8-by-8 in size, initialized with all zeroes

```
for (i in 1:n_states) {
    for (j in 1:n_states) {
        r[i][j] <- 0.0
    }
}
```

Now we need to populate the non-zero rate matrix elements, which are in terms of dispersal and extinction rates. We'll use one dispersal rate and one extinction rate for this tutorial, and explore more complex models in later sections. For later reference, this will be called the “same rate” model.

First, create a extinction rate parameter and assign it a scale move

```
r_e ~ dnExponential(10.0)
mv[++mvi] = mvScale(r_e, weight=5)
```

Before assigning the rates to the rate matrix, we'll create a vector to hold the per-area extinction rates

```
for (i in 1:n_areas) {
    e[i] := r_e
}
```

Now create the dispersal rate and scale move

```
r_d ~ dnExponential(10.0)
mv[++mvi] = mvScale(r_d, weight=5)
```

then assign the between-area dispersal rates as determined by r_d

```
for (i in 1:n_areas) {
    for (j in 1:n_areas) {
        d[i][j] <- 0.
        if (i != j) {
            d[i][j] := r_d
        }
    }
}
```

Although the DEC rate matrix can be easily constructed by typing

```
q := fnDECRateMatrix(d,e)
```

manually encoding the structure of the DEC rate matrix illuminates the relationship between dispersal rates, extinction rates, area gain rates, and area loss rates. To start, we'll populate the non-zero rate matrix elements. Rates are indexed by the natural number value of the range (plus one), e.g. the range spanning Eurasia and Africa is coded as 011, which is state 4.

First assign the extinction (range loss) rates

```
r[2][1] := e[1]          # 100 -> 000 : Extirpate in area 1
r[3][1] := e[2]          # 010 -> 000 : Extirpate in area 2
r[4][2] := e[2]          # 011 -> 001 : Extirpate in area 2
r[4][3] := e[3]          # 011 -> 010 : Extirpate in area 3
r[5][1] := e[3]          # 001 -> 000 : Extirpate in area 3
r[6][2] := e[1]          # 101 -> 001 : Extirpate in area 1
r[6][5] := e[3]          # 101 -> 100 : Extirpate in area 3
r[7][3] := e[1]          # 110 -> 010 : Extirpate in area 1
r[7][5] := e[2]          # 110 -> 100 : Extirpate in area 2
r[8][4] := e[1]          # 111 -> 011 : Extirpate in area 1
r[8][6] := e[2]          # 111 -> 101 : Extirpate in area 2
r[8][7] := e[3]          # 111 -> 110 : Extirpate in area 3
```

then the dispersal (range gain) rates

```
r[2][4] := d[3][2]      # 001 -> 011 : Disperse from area 3 to 2
r[2][6] := d[3][1]      # 001 -> 101 : Disperse from area 3 to 1
r[3][4] := d[2][3]      # 010 -> 011 : Disperse from area 2 to 3
r[3][7] := d[2][1]      # 010 -> 110 : Disperse from area 2 to 1
r[5][6] := d[1][3]      # 100 -> 101 : Disperse from area 1 to 3
r[5][7] := d[1][2]      # 100 -> 110 : Disperse from area 1 to 2
r[4][8] := d[2][1] + d[3][1] # 011 -> 111 : Disperse from area 2 to 1 and from 3 to 1
r[6][8] := d[1][2] + d[3][2] # 101 -> 111 : Disperse from area 1 to 2 and from 3 to 2
r[7][8] := d[1][3] + d[2][3] # 110 -> 111 : Disperse from area 1 to 3 and from 2 to 3
```

Show the value of **r** and compare it to the matrix in Section 2.2.

Of course we did not need to declare **d** and **e** to assign **r**, but we'll see these intermediate variables act as a template expose the structure of **r** for modification.

So far, we only have the desired parameterization of the rate matrix, but we still haven't created a rate matrix function. Converting the vector-of-vectors, **r**, into a simplex allows us to use existing rate matrix functions.

First, we'll convert **r** into a one-dimensional vector, skipping the diagonal elements.

```
k = 1
for (i in 1:n_states) {
    for (j in 1:n_states) {
        if (i != j) {
            er_nat[k++] := r[i][j]
        }
    }
}
```

Finally, normalize `er_nat` using a simplex, then pass the resulting exchangeability rates as arguments into the rate matrix function, `q`.

```
er := simplex(er_nat)
q := fnFreeK(er)
```

This yields the desired three-area DEC rate matrix modeling anagenic character change.

In contrast, cladogenic event probabilities are given by a transition probability matrix and do not require a rate matrix. First, we will create a vector of prior weights on cladogenesis events. Here, we assign a flat prior to all cladogenic events

```
widespread_sympatry_wt <- 1.0
subset_sympatry_wt   <- 1.0
allopatry_wt          <- 1.0
clado_prior           <- [ widespread_sympatry_wt, subset_sympatry_wt, allopatry_wt ]
```

then create the distribution over cladogenic event types and add its MCMC move

```
clado_type           ~ dnDirichlet(clado_prior)
mv[++mvi]           = mvSimplexElementScale(clado_type, alpha=10, weight=5)
```

To give the simplex elements descriptive names when monitored, assign the values to deterministic nodes

```
widespread_sympatry := clado_type[1]
subset_sympatry      := clado_type[2]
allopatry             := clado_type[3]
```

Then create the cladogenic transition probability matrix, which assigns probabilities to cladogenic event classes according to `clado_type_prob`

```
clado_prob := fnCladoProbs(clado_type, n_areas, 2)
```

Add a parameter for a biogeographical clock, which scales the overall rate of range evolution. As a prior, an exponential distribution with rate 10 generates one dispersal or extinction event per 10 million years.

```
clock_bg ~ dnExponential(10)
mv[++mvi] = mvScale(clock_bg, weight=5)
```

Finally, all our model components are encapsulated in the `dnPhyloCTMCClado` distribution, which is similar to `dnPhyloCTMC` except specialized to integrate over cladogenic events. Although this dataset has three areas, it is recognized single character with states valued from 1 to 2^3 , hence `nSites=1`.

```
m ~ dnPhyloCTMCClado( tree=psi, Q=q, cladoProbs=clado_prob, branchRates=clock_bg, nSites =1, type="NaturalNumbers" )
```

The remaining tasks should be familiar by now, so we can proceed briskly. Attach the observed ranges to the model.

```
m.clamp(data)
```

Compose the model.

```
mdl = model(m)
```

Add the monitors. (The `mnJointConditionalAncestralState` monitor will be described in the next section.)

```
mn[1] = mnScreen(clock_bg, d[1][2], d[1][3], d[2][1], d[2][3], d[3][1], d[3][2], e[1], e [2], e[3], widespread_sympatry, subset_sympatry, allopatry, printgen=1000)
mn[2] = mnFile(clock_bg, d[1][2], d[1][3], d[2][1], d[2][3], d[3][1], d[3][2], e[1], e [2], e[3], widespread_sympatry, subset_sympatry, allopatry, file=out_fn+".params.txt ")
mn[3] = mnJointConditionalAncestralState(tree=psi, ctmc=m, filename=out_fn+".states.txt ", type="NaturalNumbers", printgen=10, withTips=true, withStartStates=true)
```

Create the MCMC object, and run the chain after burn-in.

```
ch = mcmc(mv,mn,mdl)
ch.burnin(1000, 10)
ch.run(10000)
```

Per-area rates

Biologically, local extinction events probably do not occur at equal rates across all areas, as done above. Ecological factors, geographical distances, etc. might cause these parameters to be weakly correlated or completely uncorrelated. Dispersal rates, also, might not be the same between pairs of areas, or even symmetric depending on the direction of dispersal. Rather than constraining all events of a type to share a common rate, instead you might give each area its own extinction parameter

```
for (i in 1:3) {
    e[i] ~ dnExponential(10.0)
    mv[++mvi] = mvScale(e[i], weight=5)
}
```

or give each ordered pair of areas it's own dispersal rate

```
for (i in 1:3) {
    for (j in 1:3) {
        d[i][j] <- 0.0
        if (i != j) {
            d[i][j] ~ dnExponential(10.0)
            mv[++mvi] = mvScale(d[i][j], weight=5)
        }
    }
}
```

Note that you don't need the global dispersal rate `r_d` and extinction rate `r_e` anymore and you can remove the variable from your analysis. `RevBayes` might give you an error message if you have left them in.

Exercises

Exercises are independent of each other, except for Exercises 3a and 3b.

- 1) Widespread sympatric speciation is thought to be evolutionarily rare. Set the Dirichlet prior on cladogenic event types to heavily disfavor these events. Using Tracer, describe how changing the cladogenesis prior affects the extinction rate when compared with the “common rate” model.
- 2) Modifying the `RevBayes` script, parameterize the rate matrix so ranges may not grow beyond two areas in size. You may use `q_test := fnDECRateMatrix(e,d,2)` to confirm your results, which will give the same rate structure (and rates, up to a rescaling constant).
- 3a) Saving your commands to a file, create a script to produce the “per-area rate” model.
- 3b) Determine if the data support the “common rate” model over the “per-area rate” model. Use the stepping stone method to compute marginal likelihoods, which will let you compute Bayes factors for model selection.

(Advanced) Joint inference of phylogeny and historical biogeography

This is a slightly more advanced question that you may want to skip if you will always use known, fixed trees in your analyses. Using what you learned in this tutorial and the CTMC tutorial, perform a joint analysis of molecular and biogeographic evolution for primates.

Start by loading the sequence data matrix specified in `data/primates_cytb.nex`.

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

We need to get some useful variables from the data so that we will be able to specify the tree prior below. These variables are the number of tips, the number of nodes and the names of the species which we all can query from the continuous character data object.

```
numTips = seqData.ntaxa()
names = seqData.names()
numNodes = numTips * 2 - 1
```

Instead of having a fixed tree as in the previous, we should now define a *random* tree. We use a birth death prior with prior distributions on the **speciation** rate and **extinction** rate.

```
speciation ~ dnExponential(10.)
extinction ~ dnExponential(10.)
moves[++mvi] = mvScale(speciation, lambda=1, tune=true, weight=3.0)
moves[++mvi] = mvScale(extinction, lambda=1, tune=true, weight=3.0)
```

The phylogeny that we used are obviously not a complete sample of all the species and you should take the incomplete sampling into account. We will simply use an empirical estimate of the fraction of species which we included in this study. For more information about incomplete taxon sampling see [Höhna et al. \(2011\)](#) and [Höhna \(2014\)](#).

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

Now we are able to specify of tree variable **psi** which is drawn from a constant rate birth-death process. We will condition the age of the tree to be 75 million years old which is approximately the crown age of primates, although this estimate is still debated. We only condition here on the crown age for simplicity because we do not use any other fossil calibration.

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

Note that, here, we do not have included any fossil information: we are merely doing *relative* dating.

The first moves on the tree which we specify are moves that change the node ages. The first move randomly picks a subtree and rescales it, and the second move randomly pick a node and uniformly proposes a new node age between its parent age and oldest child's age.

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

We also need moves on the tree topology to estimate the phylogeny. The two moves which you use are the nearest-neighbor interchange (NNI) and the fixed-nodeheight-prune-and-regraft (FNPR) (Höhna and Drummond 2012).

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

In the next step we set up the substitution model. First, we create a substitution model, just like what you probably did in previous tutorial (*e.g.*, RB_CTMC_Tutorial). In a first step, we will use a GTR+Gamma model. We can use a flat Dirichlet prior density on the exchangeability rates **er_mol** and the base frequencies **pi_mol**.

```
er_mol_prior <- v(1,1,1,1,1,1)
er_mol ~ dnDirichlet(er_mol_prior)
pi_mol_prior <- v(1,1,1,1)
pi_mol ~ dnDirichlet(pi_mol_prior)
```

Now add the simplex scale move one each the exchangeability rates **er_mol** and the stationary frequencies **pi_mol** to the moves vector:

```
moves[++mvi] = mvSimplexElementScale(er_mol,weight=15)
moves[++mvi] = mvSimplexElementScale(pi_mol,weight=5)
```

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

```
Q_mol := fnGTR(er_mol,pi_mol)
```

The next part of the substitution process is the rate variation among sites. We will model this using the commonly applied 4 discrete gamma categories which only have a single parameter **alpha**. Let us specify the rate of **alpha** to 0.05 (thus the mean will be 20.0).

```
alpha_prior <- 0.05
```

Then create a stochastic node called **alpha** with an exponential prior:

```
alpha ~ dnExponential(alpha_prior)
```

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

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

The random variable that controls the rate variation is the stochastic node **alpha**. We will apply a simple scale move to this parameter.

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

This finishes the substitution process part of the model.

Then next part of the model is the clock model. Here we need a clock model because we work on a time tree. We use an exponential distribution with expectation 0.1.

```
clock_mol ~ dnExponential(10)
moves[++mvi] = mvScale(clock_mol, lambda=1, tune=true, weight=2.0)
```

Introduce a common prior for the molecular and biogeographical clocks.

```
clock_scale_bg ~ dnGamma(2.0,2.0)
clock_bg      := clock_mol * clock_scale_bg
```

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

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

Finally we need to attach the molecular sequence data to our model.

```
seq.clamp(seqData)
```

We are essentially done now. We only need to add a new monitor for the tree so that we can monitor and build the maximum a posteriori tree later.

```
monitors[3] = mnFile(filename="output/biogeography_DEC_joint.trees", printgen=100, separator =
TAB, psi)
```

Create the range evolution model as before, being sure to use the same `psi` and `clock_bg` from above.

The remaining challenge is to compose both submodels together using `model` and creating an MCMC object with `mcmc`. Good luck!

Epoch models and ancestral range reconstruction

Ancestral range reconstruction

From the previous section, we created an ancestral state monitor by

```
mn[3] = mnJointConditionalAncestralState(tree=psi, ctmc=m, filename=out_fn+".states.txt"
", type="NaturalNumbers", printgen=10, withTips=true, withStartStates=true)
```

In this section, we'll use the output from running the DEC analysis assuming all each ordered pair of areas has its own dispersal parameter, and each area has its own extinction parameter. To generate these results, run

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

The joint-conditional ancestral state monitor samples the joint distribution of ancestral states every `printgen` iterations over the entire tree, conditioning on the observed tip states. Due to cladogenesis, the state before speciation and the states inherited after speciation may differ, which we monitor setting `withStartStates=true`. For convenience we record the tip states using `withTips=true` though these are known through the input data.

The resulting file, `bg_diff.states.txt`, appears as

Iteration	start_0	end_0	start_1	end_1	start_2	end_2	start_3	end_3	...
0	2	2	2	4	2	2	2	6	...
10	2	2	4	4	2	2	2	6	...
20	2	2	2	4	2	2	6	6	...
...									

where columns give the integer-valued range corresponding to either the start or end of a branch leading to a particular node, and each row corresponds to a single sample drawn from the joint distribution of ancestral states. For example, the lineage leading to the node indexed 1 starts in state 2 (Africa) and ends in state 4 (Eurasia) at iteration 20. In Section 21.1, we will look at how stochastic mappings—the completely realized biogeographic history—may be analysed for data-augmented models. If you wish to write your own scripts to analyse the ancestral range reconstructions, know that each node's index is recorded when writing a tree's Newick string to file so states may be mapped on to the tree.

Data exploration with Phylowood

To interpret the biogeographical history of primates, we will generate a Phylowood (<http://mlandis.github.io/phylowood>) animation. First, we'll create variables for the relevant files

```
state_fn = "output/bg_diff.states.txt"
atlas_fn = "data/earth3.still.atlas.txt"
phw_fn = "output/bg_diff.phw.txt"
```

then create the animation file

```
convertToPhylowood(treefile=tree_fn, geofile=atlas_fn, statefile=state_fn, outfile=
    phw_fn, burnin=0., chartype="NaturalNumbers", bgtype="Range")
```

This file summarizes the MCMC output from a RevBayes biogeographical analysis as a Nexus-formatted file, which is used by Phylowood to generate interactive animations to explore biogeographic reconstructions.

- Open <http://mlandis.github.io/phylowood>.
- Drag and drop ./output/bg_diff.phw.txt into the text field.

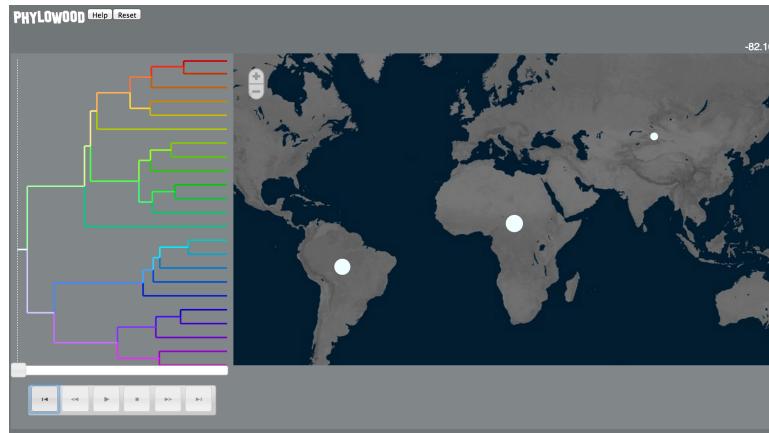


Figure 20.1: Phylowood frame showing posterior ancestral range of root node.

- Click the Play button to view the animation.

There are three control panels to help you filter data: the media panel, the map panel, and the phylogeny panel. The media buttons correspond to Beginning, Slow/Rewind, Play, Stop, Fast Forward, Ending (from left to right). The animation will play the timeframe corresponding to the slider.

- Drag the slider to the right (the present).

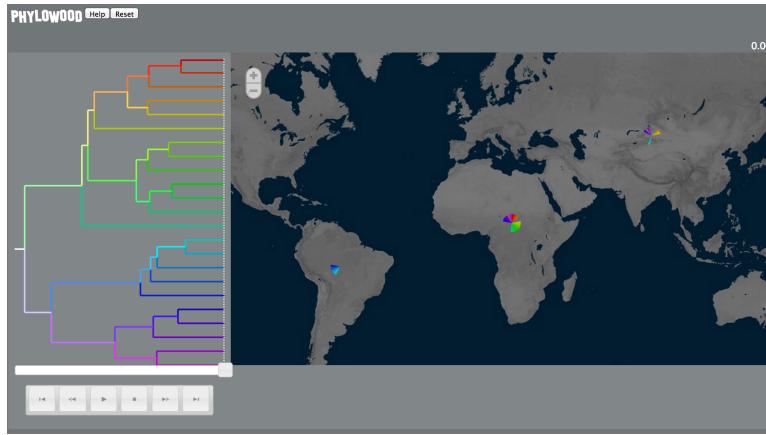


Figure 20.2: Phylowood frame showing distribution of extant taxon ranges.

- Pan and zoom around the map.

Marker colors correspond to the phylogenetic lineages in the phylogeny panel. Markers are split into slices and (loosely) sorted phylogenetically, so nearby slices are generally closely related. At divergence events, a marker's radius is proportional to the marginal posterior probability the node was present in the area at that time. Between divergence events, marker's radius is simply an interpolation of the values at the two endpoints. Some information about geological constraints and cladogenic events is lost.

- Mouseover an area to learn which lineage it belongs to and its presence probability.

Since it's difficult to see how specific clades evolve with so many taxa, Phylowood offers two ways to filter taxa from the animation. We call the set of a lineage, all its ancestral lineages towards the root, and all descendant lineages a phylogenetic heritage. The root's heritage is the entire clade. A leaf node's heritage is a path from the tip to the root.

- Mouseover a lineage to temporarily highlight the lineage's heritage. Remove the mouseover to remove the highlight effect.

The highlight effect is temporary and quickly allows you to single out lineages of interest during animation. Phylowood also offers a masking effect that persists until an unmask command is issued.

- Double-click the white root branch to mask the root node's heritage (all lineages). Single click a lineage to unmask that lineage's heritage.

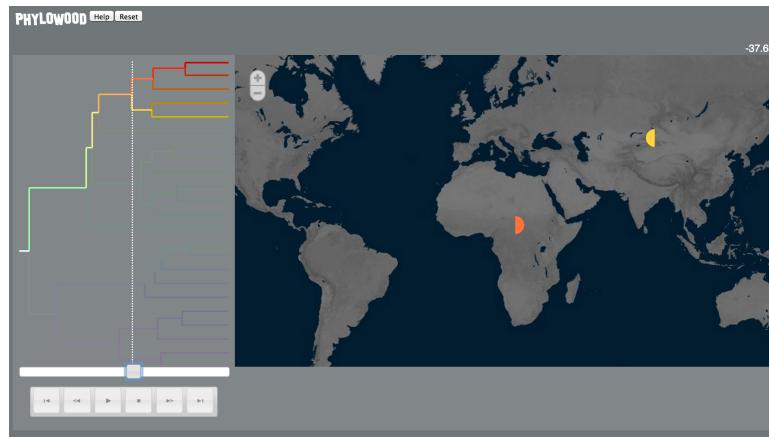


Figure 20.3: Phylowood frame highlighting the ancestral range for the MRCA extant lorises.

Now that the masking effects are in place, you’re free to interact with other map components. In addition, the area of marker sizes is only distributed among unmasked lineages.

Visit <https://github.com/mlandis/phylowood/wiki> to learn more about Phylowood.

Phylowood is useful to understand whether your model generates sensible reconstructions. Keep in mind the animation and inference both use modern geographies, and that dispersal rates are not modeled to depend on geographical features. Notice the primate MRCA is widespread in two areas, the New World and Africa. The root age of the primate tree is about 80Mya, which is about 40Mya after East and West Gondwanaland split to separate modern Africa and South America, so vicariance did not give rise to New World Monkeys. This is corroborated by examining the fossil record, where the first appearance of primates in South America is about 35Mya. The standing theory is primates disperse from Africa to South America in a rare “sweepstakes.” The naive primate biogeographic model described so far should produce more realistic reconstructions by taking continental drift into account.

Epoch models

To model the effects of geography, we will use *epoch* models. Rather than assuming the evolutionary process is constant with respect to time, it assumes the process is piecewise-constant. For example, Africa (along with all of Gondwanaland) split from Eurasia (and all of Laurasia) around 180 Mya. Africa merged with Eurasia only about 50 Mya. It is reasonable to expect that dispersal rates between Africa and Eurasia were lower before 50 Mya, and higher after 50 Mya until present.

To model this, we will specify one rate matrix that describes anagenic dispersal and extinction processes before for ages $t \geq 50$ and a second rate matrix for the process operating after $0 \leq t < 50$.

To proceed, first we will read in an **Atlas** file, which fully describes the geography per time interval. For more on the **Atlas** format, see Section XXX.

```
# read the atlas
atlas <- readAtlas("data/earth3.drift.atlas.txt")
n_epochs = atlas.nEpochs()
times <- atlas.epochTimes()
```

This atlas contains two epochs, each with three areas, and a single breakpoint at age 50 (50Mya). For example, the area for Africa in the second epoch from 50Mya – present reads

```
{
    "latitude": 10.0,
    "longitude": 20.0,
    "dispersalValues": [ 0.1, 0.0, 1.0 ],
    "extinctionValues": [ 1.0 ],
    "name": "Africa"
}
```

The `dispersalValues` values share the ordering of the areas, and each area reports a value regarding its relationship to other areas. Here, 0.1 in position 1 indicates the Atlantic Ocean obstructs dispersal from Africa to the Americas, while the 1.0 in position 3 indicates African primates may easily disperse into Eurasia by way of the Arabian Peninsula. (Note 0.0 in position 2 is the dispersal value from Africa to itself, so the value is arbitrary.)

To extract these values in matrix form

```
d_prior <- atlas.getValues("dispersal")
```

where, for example, the dispersal rate more than 50 Myr in the past (epoch 1) from Africa (area 2) to Eurasia (area 3) is accessed by

```
d_prior[1][2][3]
```

To use these empirical priors to rescale our naive priors

```
for (t in 1:n_epochs) {
    for (i in 1:n_areas) {
        for (j in 1:n_areas) {
            if (i != j) {
                d_raw[t][i][j] ~ dnExp( 10. )
                mv[mvi++] = mvScale(d_raw[t][i][j], weight=2)
                d[t][i][j] := d_raw[t][i][j] * abs(d_prior[t][i][j])
            } else {
                d[t][i][j] <- abs(0.)
            }
        }
    }
}
```

We'll make no strong prior assumptions about epochs or areas differentially affecting extinction rates.

```
# set up extinction per epoch
e_prior <- atlas.getValues("extinction")
for (t in 1:n_epochs) {
    for (i in 1:n_areas) {
        e_raw[t][i] ~ dnExp( 10. )
        mv[mvi++] = mvScale(e_raw[t][i], weight=2)
        e[t][i] := e_raw[t][i] * abs(d_prior[t][i][1])
    }
}
```

Now we have dispersal rates and extinction rates per epoch in the matrices `d` and `e`.

Extant primate ranges do not appear capable of spanning all three areas simultaneously, so we might constrain the range evolution process to only allow ranges of size two

```
rangeSize <- simplex(0,1,1,0)
```

where the first and last elements are set to zero, so ranges cannot be size 0 or size 3. By calling `fnDECRoot` with the `rangeSize` parameter, we can also force the MRCA range size to be a single area.

```
rf := fnDECRoot( rep(1,n_states), rangeSize=simplex(0,1,0,0) )
```

Next, we'll create a rate matrix for each epoch

```
for (t in 1:n_epochs) {
    rates[t] ~ dnGamma(2,2)
    mv[mvi++] = mvScale(rates[t], weight=2)
    q[t] := fnDECRateMatrix(d[t], e[t], rangeSize)
}
```

with `rates` being an epochal rate multiplier, each with mean 1. Notice, we construct `q[t]` for each epoch `t` in `num_epochs`, using the epoch's dispersal values `d[t]` and extinction values `e[t]`. Finally, we wrap our vector of rate matrices, `q`, along with the epoch boundary times, `times`, and our epochal rates, `rates`

```
q_epoch := fnEpoch( Q=q, times=times, rates=rates )
```

Parameters like `clado_prob` and `clock_bg` still need to be created, as they were in the previous section.

Otherwise, the model is created as before, except passing `q_epoch` into `dnPhyloCTMCClado`.

```
m ~ dnPhyloCTMCClado( tree=psi, Q=q_epoch, rootFrequencies=rf, cladoProbs=clado_prob,
  branchRates=clock_bg, nSites=1, type="NaturalNumbers" )
m.clamp(data)
mdl = model(m)
```

This biologically and geographically informed model produces a more realistic range reconstruction

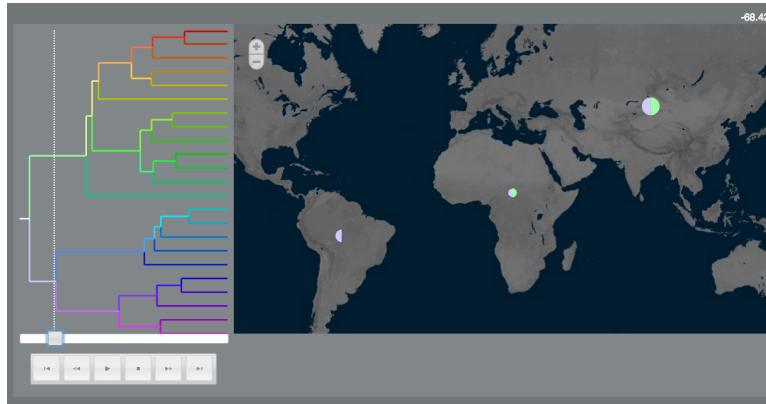


Figure 20.4: Phylowood frame showing Asian origin of primates, with subsequent dispersal into Africa and Americas.

This example model may be run by typing

```
RevBayes_scripts/biogeography_epoch.Rev}
```

which also produces the above Phylowood animation file.

Bibliography

- Goldberg, E. E., L. T. Lancaster, and R. H. Ree. 2011. Phylogenetic inference of reciprocal effects between geographic range evolution and diversification. *Systematic Biology* 60:451–465.
- Höhna, S. 2014. Likelihood Inference of Non-Constant Diversification Rates with Incomplete Taxon Sampling. *PLoS One* 9:e84184.
- Höhna, S. and A. J. Drummond. 2012. Guided tree topology proposals for Bayesian phylogenetic inference. *Systematic Biology* 61:1–11.
- Höhna, S., T. Stadler, F. Ronquist, and T. Britton. 2011. Inferring speciation and extinction rates under different species sampling schemes. *Molecular Biology and Evolution* 28:2577–2589.
- Matzke, N. J. 2012. Founder-event speciation in BioGeoBEARS package dramatically improves likelihoods and alters parameter inference in dispersal-extinction-cladogenesis DEC analyses. *Frontiers of Biogeography* 4:210.

Ree, R. H., B. R. Moore, C. O. Webb, M. J. Donoghue, and K. Crandall. 2005. A likelihood framework for inferring the evolution of geographic range on phylogenetic trees. *Evolution* 59:2299–2311.

Chapter 21

Many Area DEC

Large numbers of areas

Data augmentation

For small rate matrices, transition probabilities of beginning in state i and ending in state j equal the matrix exponential of the underlying rate matrix, scaled by the elapsed time of the process. This integrates over all unobserved transition events during the time interval t . Unfortunately, computing the matrix exponential scales poorly as the state space increases, *i.e.*, $O(n^4)$ for n states.

Alternatively, the probability of beginning in state i and ending in state j can be computed easily when the explicit series of event types and times are known. While we will never know the exact history of events, we can use stochastic mapping in conjunction with Markov chain Monte Carlo (MCMC) to repeatedly sample range evolution histories that are consistent with the ranges observed in the study taxa at the tips of the phylogeny. This technique is called data augmentation. It was first applied in phylogenetics to tertiary structure-dependent evolution of protein-coding nucleotide sequences (Robinson et al. 2003), and later extended to range evolution in Landis et al. (2013).

Using data augmentation, we will approximate $\text{Prob}(\mathbf{X}_{\text{aug}}, \theta | \mathbf{X}_{\text{obs}}, T, M)$, where \mathbf{X}_{obs} is the range data observed at the tips, \mathbf{X}_{aug} is the distribution of ancestral range reconstructions over the phylogeny, T , where \mathbf{X}_{aug} is inferred jointly with the parameters, θ , assuming the range evolution model, M , that describes \mathbf{Q} above. From a Bayesian perspective, \mathbf{X}_{aug} is effectively treated as part of the parameter space in the posterior distribution. Just as MCMC evaluates the posterior distribution for sampled parameter values, the same is done for sampled character histories. Thus, ancestral range reconstructions are a by-product of data augmentation, which are often of primary interest to biogeographers.

You may wonder why matrix exponentiation works fine for molecular substitution models and large multiple sequence alignments. It is the non-independence of character evolution that induces large state spaces, along with cladogenic processes that affect the entire set of characters (*e.g.* non-identical range inheritance following speciation). Molecular substitution models typically assume each character in an alignment evolves independently, which may be justified by the ability of recombination to degrade linkage disequilibrium over geological time scales. Conveniently, this keeps \mathbf{Q} small even for datasets with many sites.

Large rate matrices and distance-dependent dispersal

A rate matrix may 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, λ_0 is the per-area rate of loss, and λ_1 is the per-area rate of gain. For example, $q_{011,111}^{(1)}$ is the rate of range expansion for $011 \rightarrow 111$ to gain area 1, which equals λ_1 . Note the rate of more than one event occurring simultaneously is zero, so a range must expand twice by one area in order to expand by two areas. This model is analogous to the Jukes-Cantor model for three independent characters with binary states, except the all-zero “null range” is forbidden.

Extending this idea, we may reasonably expect that a range expansion event into an area depends on which

nearby areas are currently inhabited, which imposes non-independence between characters. The transition rate might then appear as

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

For this tutorial, you can take $\eta(\cdot)$ to adjust the rate of range expansion into area a by considering how close it is to the current range, \mathbf{y} relative to the closeness of all other areas unoccupied by the taxon. The β parameter rescales the importance of geographic distance between two areas by a power law. Importantly, $\eta(\cdot) = 1$ when $\beta = 0$, meaning geographic distance between areas is irrelevant. Moreover, when $\beta > 0$, $\eta(\cdot) < 1$ when area a is relatively distant and $\eta(\cdot) > 1$ when area a is relatively close. See [Landis et al. \(2013\)](#) for a full description of the model.

? Write down a rate function where the per-area rates of gain and loss depend on the number of currently occupied areas.

Specifying a data augmented DEC model

Start to create helper variables for file handling,

```
area_fn = "data/earth25.still.atlas.txt"
data_fn = "data/primates_bg_n25.nex"
tree_fn = "data/primates.tree"
out_fn = "output/bg_large"
```

read in our tree,

```
psi <- readTrees( tree_fn )[1]
```

populate our range observations,

```
data = readDiscreteCharacterData( data_fn )
```

and read in our geographical information (for details, see Section XX).

```
atlas = readAtlas( area_fn )
n_areas = atlas.nAreas()
```

Lastly, create index variables to populate our move and monitor vectors,

```
mi = 0
```

and assign the number of generations to run the MCMC analysis

```
ngen = 500000
```

To later interpret ancestral state monitors phylogenetically, save a copy of `tree` annotated with internal node indexes.

```
write(psi,filename=tree_fn+".index.tre")
```

Proceeding with the model configuration, we'll first create our rate matrix that determines the rate of per-area gain and loss given the current geographical layout of the range. In `RevBayes`, data-augmented CTMC analyses require `RateMap` functions to determine event rates, which differ from the familiar `RateMatrix` functions that include `fnJC` and `fnGTR` as members. In their simplest form, `RateMap` functions generate the rates of change over the full set of characters, where each character evolves according to a provided `RateMatrix` function. Additionally, a `RateMap` accepts a rate modifier function that induces some correlation structure to character change evolution. In this section, we'll be creating a biogeographic `RateMap` function for the dispersal-extinction process given in Section 21.1.2.

First, create a biogeographical clock to scale the rate of range evolution.

```
clock_bg ~ dnExponential(10)
moves[++mvi] = mvScale(clock_bg, lambda=0.5, weight=5.0)
```

Next, instantiate a simplex of gain and loss rates, distributed by a flat Dirichlet prior,

```
glr ~ dnDirichlet([1,1])
moves[++mvi] = mvSimplexElementScale(glr, alpha=30.0, weight=5.0)
```

and use deterministic nodes to assign the rates nicknames.

```
r_gain := glr[1]
r_loss := glr[2]
```

Insert the simplex into the rate matrix q_{area} , which gives the average rate of area gain and loss per area.

```
q_area := fnFreeBinary(glr)
```

Next, we will create `dp` to represent the β parameter, which determines the importance of geographical distance to dispersal. Remember that values of β far from zero means distance is important. So, if we assign a prior that pulls β towards zero, then posterior values of β far from zero indicate the range data are informative of the importance of distance to dispersal. We'll use an exponential distribution with mean 1.0 as a prior for `dp`.

```
dp ~ dnExponential(10.0)
moves[++mvi] = mvScale(x=dp, lambda=0.5, tune=true, weight=5.0)
```

We will also create a deterministic node to modify the rate of dispersal between areas by evaluating `dp` and `atlas`. This node is determined by the function `fnBiogeogrM`, where GRM stands for “geographical rate modifier”, and plays the role of the $\eta(\cdot)$ rate-modifier function mentioned earlier. We will tell the `fnBiogeogrM` function to modify dispersal rates based on distances and whether or not the area exists during an epoch.

```
grm := fnBiogeogrM(atlas=atlas, distancePower=dp, useDistance=true)
```

Now we need a deterministic node to represent the rate matrix, \mathbf{Q} . To determine the value of this node, we'll use the function `fnBiogeodE` to assign our model parameters to transition rates as described in the introduction. As input, we'll pass our gain and loss rates, `q_area`, our geographical rate modifier, `grm`, and the biogeographical clock `clock_bg`. In addition, we'll inform the function of the number of areas in our analysis and whether we will allow species to be absent in all areas (i.e. have the null range).

```
q_range := fnBiogeodE(gainLossRates=q_area, branchRates=clock_bg, geoRateMod=grm,
numAreas=n_areas, forbidExtinction=true)
```

As with the simple DEC model, we assign a flat Dirichlet prior over cladogenic event type probabilities

```
clado_prob ~ dnDirichlet( [1, 1, 1] )
widespread_sympatry := clado_prob[1]
subset_sympatry     := clado_prob[2]
allopatry           := clado_prob[3]
moves[++mvi] = mvSimplexElementScale(clado_prob, alpha=20.0, weight=5.0)
```

Finally, the data evolve according to a phylogenetic CTMC process. Here, we decalre a stochastic node using `dnPhyloDACTMC` where DA indicates the distribution requires data augmentation to compute the

likelihood rather than Felsenstein's pruning algorithm. To create the distribution, we must pass it our `tree` and `q_range` objects, but additionally inform the distribution that it will be using a biogeographic model, that it will introduce the simple cladogenic range evolution events described in [Ree and Smith \(2008\)](#) (`useCladogenesis=true`), and that it will assign zero probability to a transition away from the null range state.

```
m ~ dnPhyloDACTMC(tree=psi, Q=q_range, cladoProbs=clado_prob, type="Biogeo",
    forbidExtinction=true, useCladogenesis=true)
```

So we may evaluate the graphical model's likelihood, we tell the CTMC to observe the `data` object, which then primes the model with data-augmented character histories.

Now `m` has a defined likelihood value.

```
m.clamp(data)
m.lnProbability()
-156.0288
```

To integrate over the space of possible range histories, we still need to add moves to propose new data augmented histories. The major challenge to sampling character histories is ensuring the character histories are consistent with the observations at the tips of the tree. The proposals in this tutorial use [Nielsen \(2002\)](#)'s rejection sampling algorithm, with some modifications to account for cladogenic events and epoch-based rate maps.

The basic idea is simple. Each time a character history proposal is called, it selects a node at random from the tree. Branch history proposals propose a new character history to a single randomly-chosen branch. Node history proposals propose a new character history for the three branches connecting to a randomly chosen node, in addition to the cladogenic state of the node itself.

For each proposal, each character's history is resampled with probability `lambda` – i.e. `lambda=0.01` resamples 1% of characters, while `lambda=1.` resamples all characters. Once the new character history is proposed, the likelihood of the model is evaluated and the MCMC accepts or rejects the new state according to e.g. the Metropolis-Hastings algorithm.

Cladogenic events require special considerations during sampling, so we indicate `type="Biogeo"`. Currently, only rejection-sampling is available for cladogenic histories, hence `proposal="rejection"`. Finally, we apply high `weight` values to the proposals since the larger the state space is, the more character history proposals needed to effectively integrate over the space of sample paths.

Let's create the character history moves as follows: conservative character history updates for paths and nodes, with `lambda=0.05`

```
moves[++mvi] = mvCharacterHistory(ctmc=m, qmap=q_range, tree=psi, lambda=0.05,
    type="Biogeo", graph="node", proposal="rejection", weight=n_nodes*5)
moves[++mvi] = mvCharacterHistory(ctmc=m, qmap=q_range, tree=psi, lambda=0.05,
```

```
type="Biogeo", graph="branch", proposal="rejection", weight=n_nodes)
```

and the same proposals for moderately-sized character history updates, with `lambda=0.2`

```
moves[++mvi] = mvCharacterHistory(ctmc=m, qmap=q_range, tree=psi, lambda=0.2,
                                    type="Biogeo", graph="node", proposal="rejection", weight=n_nodes*5)
moves[++mvi] = mvCharacterHistory(ctmc=m, qmap=q_range, tree=psi, lambda=0.2,
                                    type="Biogeo", graph="branch", proposal="rejection", weight=n_nodes)
```

and radical updates

```
moves[++mvi] = mvCharacterHistory(ctmc=m, qmap=q_range, tree=psi, lambda=1.0,
                                    type="Biogeo", graph="node", proposal="rejection", weight=n_nodes*5)
moves[++mvi] = mvCharacterHistory(ctmc=m, qmap=q_range, tree=psi, lambda=1.0,
                                    type="Biogeo", graph="branch", proposal="rejection", weight=n_nodes)
```

Next, create the model object,

```
my_model = model(m)
```

and monitors for our simple parameters.

```
monitors[1] = mnScreen(clock_bg, r_gain, r_loss, dp, subset_sympatry, allopatry,
                       widespread_sympatry, printgen=100)
monitors[2] = mnFile(clock_bg, r_gain, r_loss, dp, subset_sympatry, allopatry,
                     widespread_sympatry, filename=out_fn+".params.txt", printgen=10)
```

Like any parameter, we can sample the augmented range histories from the MCMC to approximate the posterior distribution of range histories. This is statistically equivalent to generating ancestral state reconstructions from a posterior distribution via stochastic mapping. We will extract these reconstructions using special monitors designed for the `dnPhyloDACTMC` distribution.

Next, we will create `mnCharHistoryNewick` monitors to record the sampled character history states for each node in the tree. This monitor has two `style` options: `counts` reports the number of gains and losses per branch in a tab-delimited Tracer-readable format; `events` reports richer information of what happens along a branch, anagenetically and cladogenetically, using an extended Newick format. How to read these file formats will be discussed in more detail in Section 21.1.4.

```
monitors[3] = mnCharHistoryNewick(filename=out_fn+".events.txt", ctmc=m, tree=psi,
    printgen=100, style="events")
monitors[4] = mnCharHistoryNewick(filename=out_fn+".counts.txt", ctmc=m, tree=psi,
    printgen=100, style="counts")
```

As our last monitor, `mnCharHistoryNhx` records character history values throughout the MCMC analysis, then stores some simple posterior summary statistics as a Nexus file. These summary statistics could be computed from the previously mentioned monitor output files, but `mnCharHistoryNhx` provides a simple way to produce Phylowood-compatible files. We will also discuss this file's format in more detail later in the tutorial.

```
monitors[5] = mnCharHistoryNhx(filename=out_fn+".phw.txt", ctmc=m, tree=psi, atlas=atlas
    , samplegen=100, maxgen=nge, burnin=0.5)
```

Finally, create the MCMC object

```
my_mcmc = mcmc(my_model, monitors, moves)
```

and run

```
my_mcmc.run(generations=nge)
```

The posterior surface over character histories is highly multimodal, so this analysis takes a long time to mix. If you're following this tutorial in a lab setting, you should proceed using the pre-analysed output files provided in `./example_output`.

Analysis output

We'll focus some attention on node 42, the most recent common ancestor of chimps and macaques, designated as the Old World most recent common ancestor (MRCA).

Biogeographic event counts from `mnCharHistoryNewick`

Recording stochastic mappings in a Tracer-compatible format requires some summarization. This monitor generates a tab-delimited file where the number of events of each type for each branch is recorded.

Open `./output/bg_3.counts.txt` in a text editor.

Iter	Posterior	Likelihood	Prior	t_s0	t_s1	t_c0	t_c1	t_c2	t_c3	b0_s0			
	b0_s1	b0_c	...										
0	-6518.54	-6519.25	0.706823	977	839	22	0	0	38	28	0	...	
10	-583.147	-585.333	2.186240	66	37	3	7	9	3	2	1	2	...
20	-296.540	-297.340	0.799755	20	25	15	1	3	3	0	1	0	...
30	-276.284	-277.257	0.972918	17	24	14	0	4	4	0	1	0	...
40	-266.569	-266.948	0.379624	18	23	15	1	3	3	0	1	3	...
...													

For example, `b0_s1` gives the number of areas that are gained for the branch leading to the node indexed 0. Referencing FigTree, we see this corresponds to chimpanzees. `b0_c` gives the cladogenic event type that gives rise to the chimp lineage, where narrow sympatry, widespread sympatry, subset sympatry, and allopatry are recorded as 0, 1, 2, and 3, respectively. The columns `t_s0` and `t_s1` give the sum of events over all branches. `t_c0`, `t_c1`, `t_c2`, and `t_c3` give the total number of narrow sympatric, widespread sympatric, subset sympatric, and allopatric cladogenic events over the entire tree.

Biogeographic event histories from `mnCharHistoryNewick`

For more detailed data exploration, this analysis also provides annotated Newick strings with the complete character mappings for the tree. (This file format is a bit unwieldy and under revision.)

→ Open `./output/bg_3.events.txt` in a text editor.

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

```
[&index=42;nd=0000000000000101000000000;pa=0000000000000100000000000;ch0
=0000000000000101000000000;ch1=0000000000000001000000000;cs=s;bn=41;ev={{t:0.138033,
a:39.4458,s:1,i:15}}]
```

Having consulted FigTree earlier, we know node 42 is the Old World MRCA. The branch began with the range `pa=0000000000000100000000000` and terminated in the state `nd=0000000000000101000000000`. Since this node is not a tip node, it represents a speciation event, so the daughter ranges are also given, `ch0=0000000000000101000000000` and `ch1=0000000000000001000000000`. The cladogenic state for this speciation event was subset sympatric, `cs=s`, rather than sympatric (wide or narrow; `w` or `n`) or allopatric (`a`).

Anagenic dispersal and extinction events occurring along the lineage leading to node 42 are recorded in `ev`, where each event has a time (relative to the absolute branch length), absolute age, state (into), and character index (`t`, `a`, `s`, `i`, resp.). Potentially, `ev` contains multiple events. For this posterior sample, the MRCA of chimps and macaques dispersed into East Africa 39.4458 million years ago, which is consistent with the ranges recorded by `pa` and `nd`.

Although we have stochastic mappings under the posterior distribution, the remaining challenge is to

summarize them into something useful. The Python script `bg_parse.py` is provided to manipulate this data format. Below are a few examples of interesting features of the posterior.

- Open a Python console and read in the events.

```
> cd RevBayes_scripts  
> python  
  
...  
  
>>> from bg_parse import *  
>>> dd=get_events(fn="../output/bg_3.events.txt")
```

By default, `get_events()` extracts a dictionary-of-dictionaries from the posterior event samples. A dictionary is very much like a vector, except values are selected by keys, so this can be thought of as similar to a two-dimensional matrix. The first key corresponds to the node (branch) whose MCMC samples you'd like to retrieve, while the second dictionary's keys are the columns correspond to MCMC states and values specific to that node. For example, say we are interested in the last five cladogenic states sampled for node 42, Old World MRCA,

```
>>> dd[42].keys()
['ch1', 'iteration', 'bn', 'nd', 'ch0', 'prior', 'posterior', 'pa', 'cs', 'ev',
 'likelihood']
>>> dd[42]['cs'][-5:]
['widespread_sympatry', 'widespread_sympatry', 'widespread_sympatry',
 'widespread_sympatry', 'widespread_sympatry']
```

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

More data-exploration functions are found in `bg_parse.py`.

Exercises

- (Under construction.)

Bibliography

- Landis, M. J., N. J. Matzke, B. R. Moore, and J. P. Huelsenbeck. 2013. Bayesian analysis of biogeography when the number of areas is large. *Systematic Biology* 62:789–804.
- Nielsen, R. 2002. Mapping mutations on phylogenies. *Systematic Biology* 51:729–739.
- Ree, R. H. and S. A. Smith. 2008. Maximum likelihood inference of geographic range evolution by dispersal, local extinction, and cladogenesis. *Systematic Biology* 57:4–14.
- Robinson, D. M., D. T. Jones, H. Kishino, N. Goldman, and J. L. Thorne. 2003. Protein evolution with dependence among codons due to tertiary structure. *Molecular Biology and Evolution* 20:1692–1704.

Part X

Phylogenetic Comparative Method

Chapter 22

Phylogenetic Comparative Analyses: Continuous Trait Evolution

Introduction

The subject of the comparative method is the analysis of trait evolution at the macroevolutionary scale. In a comparative context, many different questions can be addressed: tempo and mode of evolution, correlated evolution of multiple quantitative traits, trends and bursts, changes in evolutionary mode correlated with major key innovations in some groups, etc (for a good introduction see [Harvey and Pagel 1991](#)).

In order to correctly formalize comparative questions, the underlying phylogeny should always be explicitly accounted for. This point is clearly illustrated, in particular, by the independent contrasts method ([Felsenstein 1985](#); [Huelsenbeck and Rannala 2003](#)). Practically speaking, the phylogeny and the divergence times are usually first estimated using a separate phylogenetic reconstruction software. In a second step, this time-calibrated phylogeny is used as an input to the comparative method. Doing this, however, raises a certain number of methodological problems:

- the uncertainty about the phylogeny (and about divergence times) is ignored
- the traits themselves may have something to say about the phylogeny
- the rate of substitution, and more generally the parameters of the substitution process, can also be seen as quantitative traits, amenable to a comparative analysis.

All these points are not easily formalized in the context of the step-wise approach mentioned above. Instead, what all this suggests is that phylogenetic reconstruction, molecular dating and the comparative method should all be considered jointly, in the context of one single overarching probabilistic model.

Thanks to its modular structure, RevBayes represents a natural framework for attempting this integration. The aim of the present tutorial is to guide you through a series of examples where this integration is achieved, step by step. It can also be considered as an example of the more general perspective of *integrative modeling*, which can be recruited in many other contexts.

Data and files

We provide several data files which we will use in this tutorial. You may want to use your own data instead. In the **data** folder, you will find the following files

- **primates_cytb.nex**: Alignment of the *cytochrome b* subunit from 23 primates representing 14 of the 16 families (*Indriidae* and *Callitrichidae* are missing).
- **primates_lhtlog.nex**: 2 life-history traits (endocranial volume (ECV), body mass; each for males and females separately) for 23 primate species (taken from the Anage database, [De Magalhaes and Costa 2009](#)). The traits have been log-transformed.
- **primates.tree**: A time calibrated phylogeny of the same 23 primates.

Univariate Brownian evolution of quantitative traits

As a first preliminary exercise, we wish to reconstruct the evolution of body mass in primates and, in particular, estimate the body mass of their last common ancestor. For this, we will assume that the

logarithm of body mass follows a simple univariate Brownian motion along the phylogeny. In a first step, we will ignore phylogenetic uncertainty: thus, we will assume that the Brownian process describing body mass evolution runs along a fixed time-calibrated phylogeny (with fixed divergence times), such as specified in the file `primates.tree`.

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

The model and the priors

A univariate Brownian motion $x(t)$ is parameterized by its starting value at the root of the phylogeny $x(0)$ and a rate parameter σ . This rate parameter tunes the amplitude of the variation per unit of time. Specifically, along a given time interval $(0, T)$, the value of X at time T is normally distributed, with mean $x(0)$ and variance $\sigma^2 T$:

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

Concerning σ , we can formalize the idea that we are ignorant about the *scale* (the order of magnitude) of this parameter by using a log-uniform prior:

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

Concerning the initial value $x(0)$ of the Brownian process at the root of the phylogeny. Alternatively, you may want to specify a normal distribution as the prior distribution on the root value if you have some prior information.

Finally, the tree topology ψ is, as mentioned above, fixed to some externally given phylogeny. The entire model is now specified: tree ψ , variance σ and Brownian process $x(t)$:

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

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.

Programming the model in RevBayes

The problem of continuous trait evolution —just as for discrete trait evolution— along a phylogeny is that we do not know the values of the traits at the internal nodes. That means, that we need to treat the states at the internal nodes as additional parameters of the model. For discrete characters we use the sum-product (a.k.a. pruning) algorithm ([Felsenstein 1981](#)) to analytically integrate over all possible states at the internal nodes. For continuous characters (traits) similar methods have been proposed. In RevBayes you

have three main ways of specifying this model and running an analysis on it. The three approaches are: (1) phylogenetic independent contrasts using the reduced likelihood (REML), (2) Brownian motion using a phylogenetic covariance matrix, and (3) a full Brownian motion model using data augmentation. Each of these approaches has there advantages and disadvantages as will be explained below. Nevertheless, all approaches give the same results in terms of rate estimation.

Phylogenetic Independent Contrasts using the reduced likelihood (REML)

The reduced or restricted maximum likelihood (REML) method computes the probability of observing the continuous character at the tips by an analytical solution to integrate over the internal states ([Felsenstein 1985](#)). This analytical solution is very fast to compute and thus can be applied to large phylogenies and/or many independent characters. However, the REML method looses the information about the location of the root state and thus you cannot infer which state the root or other internal nodes have.

You do not need to understand the algorithm but we provide a sketch of the idea behind REML to give you some insights. REML compute the values at the internal nodes as the phylogenetic contrasts $x_k = x_i - x_j$ where x_i and x_j are the values of the child nodes in the phylogeny. Then, we can compute the probability of observing the contrast x_k using the probability density of a normal distribution with mean $\mu = 0$ and standard deviation $\sigma = \sqrt{\nu_i + \delta_i + \nu_j + \delta_j}$ where ν_i and ν_j are the (scaled) branch lengths leading to node i and j respectively. δ is the additional uncertainty that is propagated through the phylogeny and is compute by $\delta_k = ((\nu_i + \delta_i) * (\nu_j + \delta_j)) / (\nu_i + \delta_i + \nu_j + \delta_j)$. These computations are done for you in the `RevBayes` distribution called `dnPhyloBrownianREML`.

In the directory `RevBayes_scripts/` you will find a script called `primatesMass_BM_REML.Rev`. This script implements the univariate Brownian model described above. Instead of re-typing the content of script entirely in the context of an interactive `RevBayes` session, you can instead run the script directly:

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

This script essentially reformulates what has been explained in the last subsection and serves as an example solution for you. For the later section you need to adjust the script.

Let us go through the script step by step in the `Rev` language. First, load the trait data:

```
contData <- readContinuousCharacterData("data/primates_lhtlog.nex")
```

If you type you will see that the continuous character data matrix contains several characters (columns).

```
contData

Continuous character matrix with 23 taxa and 11 characters
=====
Origination:          primates_lhtlog.nex
Number of taxa:        23
Number of included taxa: 23
Number of characters:   11
Number of included characters: 11
Datatype:              Continuous
```

Since we only want the body mass (of females) we exclude all but the third character

```
contData.excludeAll()
contData.includeCharacter(3)
```

Next, load the time-tree from file. Remember that we use in this first simple example a fixed tree that we assume is known without uncertainty.

```
treeArray <- readTrees("data/primates.tree")
psi <- treeArray[1]
```

- You may want to look at this tree before by loading the **primates.tree** in FigTree or any other tree visualization software.

As usual, we start by initializing some useful helper variables. For example, we set up a counter variable for the number of moves that we already added to our analysis. This will make it much easier if we extend the model or analysis to include additional moves or to remove some moves.

```
mvi = 0
```

Then, we define the overall rate parameter σ which we assign a (truncated) log-uniform prior. Note that it is more efficient in Bayesian inference to specify a uniform prior and then to transform the parameter which we will use here:

```
logSigma ~ dnUniform(-5,5)
sigma := 10^logSigma
```

Using this approach we have specified a prior probability distribution on **sigma** between 10^{-5} to 10^5 which should be broad enough to include all reasonable values.

Since the rate of trait evolution **logSigma** is a stochastic variable and we want to estimate it, we need to add a sliding move on it. Remember that the sliding move proposes new values drawn from a window with width **delta** and is centered around the current values; thus it slides through the parameter space together with the current parameter value.

```
moves[++mvi] = mvSlide(logSigma, delta=1.0, tune=true, weight=2.0)
```

Next, define a random variable from the univariate Brownian-Phylo-REML process, which we will call **logmass**. We need to provide the tree variable **psi**, some branch-specific rate multiplier parameter which we simply set to 1, the shared rate for this site **sigma** and the number of sites (number of continuous traits) that we use.

```
logmass ~ dnPhyloBrownianREML(psi, branchRates=1.0, siteRates=sigma, nSites=1)
```

Now, condition the Brownian model on empirically observed values for body mass in the extant taxa.

```
logmass.clamp( contData )
```

The model is now entirely specified and we can create a model object containing the entire model graph by providing it with only one of our model variables, *e.g.*, **sigma**.

```
mymodel = model(sigma)
```

To see what is happening during the MCMC let us make a screen monitor that tracks the rate **sigma**.

```
monitors[1] = mnScreen(printgen=10, sigma)
```

Additionally, we'll use a file monitor that does the same thing, but directly stores the values into a file.

```
monitors[2] = mnFile(filename="output/primates_mass_REML.log", printgen=10, separator = TAB, sigma)
```

We can finally create a mcmc, and run it for a good 100 000 cycles after we did a burnin phase of 10 000 iterations:

```
mymcmc = mcmc(mymodel, monitors, moves)
mymcmc.burnin(generations=10000,tuningInterval=500)
mymcmc.run(100000)
```

Exercises

- Run the model.
- using **Tracer**, visualize the posterior distribution on the rate parameter **sigma**
- calculate the 95% credible interval for the rate of evolution of the log of body mass (σ)

Phylogenetic covariance matrix

The second method that we will use creates a phylogenetic covariance matrix. The phylogenetic covariance matrix method integrates over the states at the internal nodes as well but uses instead a multivariate normal distribution. The key advantage is that this method provides information about the root state since it models the root state as an additional parameter of the model. The disadvantage is that it is very computationally intensive. That means, that the phylogenetic covariance matrix approach may take long for very large data sets (at least in its current implementation).

- Copy the file `primatesMass_BM_REML.Rev`, name it for example `primatesMass_BM_Cov.Rev` and start editing it.

In the previous example, the REML approach, we did not specify a parameter for the state at the root. In this exercise, we need this additional parameter. Let us use a uniform prior distribution on the logarithm of the root mass. A uniform prior between -100 and 100 should be diffuse enough. Just image how big or small an individual needs to be if it has body mass smaller than $\exp(-100)$ or larger than $\exp(100)$.

```
rootlogmass ~ dnUniform(-100,100)
```

Next, we'll specify a sliding move that proposes new values for the `rootlogmass` randomly drawn from a window centered around the current value.

```
moves[++mvi] = mvSlide(rootlogmass,delta=10,tune=true,weight=2)
```

Finally, we need to substitute the `dnPhyloBrownianREML` by `dnPhyloBrownianMVN` to use the phylogenetic covariance matrix approach. Again, we provide the tree variable `psi`, some branch-specific rate multiplier parameter which we simply set to 1, the shared rate for this site `sigma` and the number of sites (number of continuous traits) that we use.

```
logmass ~ dnPhyloBrownianMVN(psi, branchRates=1.0, siteRates=sigma, rootStates=rootlogmass,
nSites=1)
```

This will automatically connect the parameters of the model together.

Additionally, we now have the extra parameter `rootlogmass` which we want to monitor. Thus we need to replace the file monitor and use instead.

```
monitors[2] = mnFile(filename="output/primates_mass_Cov.log", printgen=10, separator = TAB,
sigma, rootlogmass)
```

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

Exercises

- Run the analysis.
- Using **Tracer**, visualize the posterior distribution on the rate parameter **sigma** and the **rootlogmass**
- How does the posterior distribution of **sigma** looks compared with the first analysis?
- Calculate the 95% credible interval for the rate of evolution of the log of body mass (σ) and the **rootlogmass**

Data augmentation

The third method to we will use is a data augmentation method. The data augmentation method uses explicitly the states at the internal nodes. We will specify the full model in **Rev**. That means that we will create a random variable for each node of the tree using a for loop. Each trait is then simply assign a normal distribution, as we described above in the model description. The advantage of this method is that you estimate the values at the internal nodes directly and that you have full control about modifying any part of the model. The disadvantage is that the MCMC algorithm will be harder if you want to jointly estimate the phylogeny, although the likelihood computation is very fast. The problem is the mixing of the MCMC because proposing new trees involves proposing new good values for the states at the internal nodes.

→ Copy the file **primatesMass_BM_REML.Rev**, name it for example **primatesMass_BM_DA.Rev** and start editing it.

]As in the previous example we will use a uniform prior distribution on the logarithm of the root mass.

```
rootlogmass ~ dnUniform(-100,100)
```

Again, we'll specify a sliding move that proposes new values for the **rootlogmass** randomly drawn from a window centered around the current value.

```
moves[++mvi] = mvSlide(rootlogmass,delta=10,tune=true,weight=2)
```

In order to create the random variables for the internal states we need to know the number of nodes and the number of tips. We will store these as some helper variables.

```
numNodes = psi.nnodes()
numTips = psi.ntips()
```

Now we are ready to specify the Brownian motion model for each branch. That is, we simply specify a new normal distributed random variable for each node with mean being equal to the value of the parent variable and the standard deviation being equal to the product of the square root of the branch length and our rate parameter **sigma**. We store all the variables in the vector **logmass**. Then we are able to access the value at the parent node using the index of the parent node, which we can obtain from the tree using the function **psi.parent(i)**. Similarly, since the variance depends on the branch length we retrieve the branch length of node with index **i** using the function **psi.branchLength(i)**.

First we need to copy (create a reference to) the **rootlogmass**

```
logmass[numNodes] := rootlogmass
```

Let us start by creating the random variables for the internal nodes. Remember that the variance is equal to **sigma**-squared times the branch length, and we need to compute the square root of it to obtain the standard deviation.

```
# univariate Brownian process along the tree
# parameterized by sigma
for (i in (numNodes-1):(numTips+1) ) {
    logmass[i] ~ dnNormal( logmass[psi.parent(i)], sd=sigma*sqrt(psi.branchLength(i)) )
    # moves on the Brownian process
    moves[++mvi] = mvSlide( logmass[i], delta=10, tune=true ,weight=2)
}
```

You may have noticed that we specified in the loop a move for each internal **logmass**. This is because we want to use the MCMC algorithm to integrate over the uncertainty in the states.

Next, we repeat the same loop but now for the tip nodes. Instead of applying a move to each tip node we will clamp the nodes. The nodes will be clamped with the data that we read in before.

```
for (i in numTips:1) {
    logmass[i] ~ dnNormal( logmass[psi.parent(i)], sd=sigma*sqrt(psi.branchLength(i)) )

    # condition Brownian model on quantitative trait data (second column of the dataset)
    logmass[i].clamp(contData.getTaxon(psi.nodeName(i))[1])
}
```

Here we do not need a **dnPhyloBrownianREML** or **dnPhyloBrownianMVN** distribution anymore because we explicitly instantiated the model. Note that the approach we have taken using the loop is the tree-plate approach described in (?)

Since we have several additional parameters —the states at the internal nodes— we will use a model monitor to write to file instead.

```
monitors[2] = mnModel(filename="output/primates_mass_DA.log", printgen=10, separator = TAB)
monitors[3] = mnExtNewick(filename="output/primates_mass_DA_ext.trees", isNodeParameter=TRUE,
    printgen=10, separator = TAB, tree=psi, logmass)
```

To get the annotate tree we use the map tree function.

```
treetrace = readTreeTrace("output/primates_mass_DA_ext.trees", treetype="clock")
map_tree = mapTree(treetrace,"output/primates_mass_DA_ext_MAP.tree")
```

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

Exercises

- Run the analysis.
- Using **Tracer**, visualize the posterior distribution on the rate parameter **sigma** and the **rootlogmass** and the internal states.
- How does the posterior distribution of **sigma** looks compared with the first and second analysis?
- Calculate the 95% credible interval for the rate of evolution of the log of body mass (σ) and the **rootlogmass**. Have they changed?

Brief intermediate summary

In the above exercises you have analyzed the log-transformed body mass using a Brownian motion (BM) along a phylogeny. We assumed the phylogeny to be known without error and were primarily interested in the rate how fast the body mass evolves. The exercises showed you three different ways how to approach the BM model, each having its advantages and disadvantages. You will need to decide for your analysis which approach is most appropriate. Our intention was mainly to show you some of the flexibility in RevBayes how to specify the same model in many different ways, exposing sometimes more and sometimes

less of the internal model graph structure. As an extra exercise you could start thinking about how to extend the basic BM.

Multiple independently evolving traits

The Brownian motion can easily be applied to multiple traits. The exactly same procedure and model will be applied as above and thus we will skip the repetitive description. The interesting questions that you can ask about multiple traits is —amongst others— if rates between traits vary.

As an example we use now the first four characters of the data matrix, which are: 1) female log endocranial volume (ECV), 2) male log ECV, 3) female log body mass and 4) male log body mass.

```
contData <- readContinuousCharacterData("data/primates_lhtlog.nex")
contData.excludeCharacter(5:11)
```

In the first simple example we assume that all site rates are equal, thus we use a rate multiplier we we call **perSiteRates**.

```
perSiteRates <- [1,1,1,1]
```

Furthermore, we have now four characters and hence we will estimate a root state for each one of them. We use the same prior probability for every character, just as above for the single character example.

```
for (i in 1:4) {
  rootlogmass[i] ~ dnUniform(-100,100)
  moves[+mvil] = mvSlide(rootlogmass[i],delta=10,tune=true,weight=2)
}
```

Finally, we bring together all the parameter to specify our Brownian motion model along the tree. Note that we specify here the site rates as the product of the global rate **sigma** time the sites specific rates (which are, however all equal in this first exercie).

```
logmass ~ dnPhyloBrownianMVN(psi, branchRates=1.0, siteRates=sigma*perSiteRates, rootStates=
  rootlogmass, nSites=4)
```

After this you need to create the model using the **model** function, create your monitors, create the MCMC and finally run the MCMC.

Exercises

- Run the analysis.
- Using **Tracer**, visualize the posterior distribution on the rate parameter **sigma** and the four **rootlogmass** values.
- Compare the results to the previous single trait analyses?

Independent site rates

```
perSiteRates ~ dnDirichlet([1,1,1,1])
moves[++mvi] = mvSimplexElementScale(perSiteRates,alpha=10,tune=true,weight=4)
```

Exercises

- Run the analysis.
- Using **Tracer**, visualize the posterior distribution on the per site rate parameter **perSiteRates**.
- How are the estimate per site rates compared to each other?

Partially shared site rates

We hope that your results in the previous analysis showed that the rates for females and males for both the ECV and the body mass are very similar. This motivates us to specify an explicit model where the rates between ECV and body mass evolution are different but shared between females and males. We will model the difference using the **siteRateDiff** parameter which is drawn from a Beta(1,1) distribution, which is essentially a uniform distribution between 0 and 1. We still choose the Beta distribution because it is easier to specify more informative priors.

```
siteRateDiff ~ dnBeta(1,1)
```

Next, we specify a sliding move on the **siteRateDiff**.

```
moves[++mvi] = mvSlide(siteRateDiff,delta=10,tune=true,weight=2)
```

To obtain the rates we use **siteRateDiff** as the rate for the ECV evolution in females and males and **1-siteRateDiff** as the rate for the body mass evolution in females and males. Note, however, that we need a small workaround here. Rev is a strictly typed language, which you may have noticed. The per site rates need to be positive real number (negative rates obviously don't make sense). However, the difference between two number is not guaranteed to be positive, only if we know that the first term is larger than the second. This is exactly the case for **1-siteRateDiff** but RevBayes doesn't know this so we need to tell it by using the absolute value **abs** function.

```
perSiteRates[1] := siteRateDiff
perSiteRates[2] := siteRateDiff
perSiteRates[3] := abs(1.0 - siteRateDiff) # specify the type here
perSiteRates[4] := abs(1.0 - siteRateDiff)
```

You could have specified many other prior distribution on the relative rate (or ratio) such as a uniform prior distribution and then use logit or hyperbolic tangent transformation. The only important factor is that the rates are somehow normalized because we use the global, shared rate **sigma**. Using the beta distribution makes the rates automatically normalized because the two different rates sum to one. Note that in the previous all four rates summed to 1.0 and here the two different rates sum to 1.0. This has the effect that we should estimate the global rate **sigma** to be half of what you estimated previously. You may want to check this once you ran the analysis and loaded the output into **Tracer**.

The remaining functions are the same as in the previous example.

Exercises

- Run the analysis.
- Using **Tracer**, visualize the posterior distribution on the per site rate parameter **perSiteRates**.
- Does the value of equal rate (**siteRateDiff**=0.5) fall into the 95% credible interval?
- If the rate does not fall into the 95% credible interval, should we reject the hypothesis that the underlying rates are equal?

Model selection

Our previous analysis explored the parameters of the rate of evolution. You will have seen that the rate of evolution was smaller for the ECV compared with the rate of evolution of body size. However, we did not perform a statistical test to reject the hypothesis if the underlying rates between the different characters are significantly different. Therefore, we will perform a marginal likelihood estimation for all three model: 1) the equal rates, 2) the independent rates, and 3) the shared rates.

- You need to adopt the three scripts of your previous analysis.

Previously you created an `mymcmc` object using the `mcmc` function. Now, you need to replace this object with the power posterior MCMC object.

```
pow_p = powerPosterior(mymodel, moves, monitors, "output/pow_p_BM_equal.out", cats=100,
sampleFreq=10)
```

Next, you need to run the burnin and afterwards run the power posterior sampler. This will create 101 output files (100 stepping stones and one for the posterior) logging the values into the monitors that you have specified before (we recommend you not to use a screen monitor). You could look into each of these files to check that you obtain sufficiently many samples for each stone.

```
pow_p.burnin(generations=10000, tuningInterval=250)
pow_p.run(generations=10000)
```

Use stepping-stone sampling to calculate marginal likelihoods.

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

RevBayes will print to the screen the estimated marginal likelihood. Write down this value and repeat the exercise for the other two models.

Exercises

- Run the three different marginal likelihood estimations and write down the marginal likelihood.
- Compute the log-Bayes factors by taking the difference of the alternative model (independent rates & shared rates) and the null hypothesis (equal rates).
- Is any of the log-Bayes factors larger than 1.16 for substantial support of the alternative hypothesis (or 2.3 for strong support)?

Estimating the phylogeny from continuous character data

In the motivation we argued that the phylogeny should be estimated together with the parameters of the evolutionary process. Only for simplicity we used fixed trees in the previous exercises. Instead of using a tree fixed to a pre-defined value, the tree should now be moved during the MCMC. Implementing this joint model in **RevBayes** is just a matter of adding the following features to the model defined in the previous section (after duplicating the script). You may want to use the REML approach for computational efficient (*i.e.*, good MCMC mixing) although the phylogenetic covariance matrix and the data augmentation are generally possible too.

We need to get some useful variables from the data so that we will be able to specify the tree prior below. These variables are the number of tips, the number of nodes and the names of the species which we all can query from the continuous character data object.

```
numTips = contData.ntaxa()
names = contData.names()
numNodes = numTips * 2 - 1
```

Instead of having a fixed tree as in the previous, we should now define a *random* tree. We use a birth death prior with prior distributions on the **diversification** rate and **turnover** rate. The **diversification** rate corresponds to the rate of growth of the tree and the **turnover** rate corresponds to the rate how quickly a species is replaced by a new one. This parametrization has the advantage that we can use meaningful prior distributions from the fossil record.

```
diversification ~ dnLognormal(0,1)
turnover ~ dnGamma(4,4)
```

Then, we compute deterministically the **speciation** and **extinction** rate from the **diversification** rate and **turnover** rate.

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

Instead, you could also use directly the speciation and extinction rates, *e.g.*, endowed with some diffuse exponential prior.

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

The phylogeny that we used are obviously not a complete sample of all the species and you should take the incomplete sampling into account. We will simply use an empirical estimate of the fraction of species which we included in this study. For more information about incomplete taxon sampling see **?** and **?**.

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

Now we are able to specify of tree variable **psi** which is drawn from a constant rate birth-death process. We will condition the age of the tree to be 75 million years old which is approximately the crown age of primates, although this estimate is still debated. We only condition here on the crown age for simplicity because we do not use any other fossil calibration.

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

Note that, here, we do not have included any fossil information: we are merely doing *relative* dating.

The first moves on the tree which we specify are moves that change the node ages. The first move randomly picks a subtree and rescales it, and the second move randomly pick a node and uniformly proposes a new node age between its parent age and oldest child's age.

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

We also need moves on the tree topology to estimate the phylogeny. The two moves which you use are the nearest-neighbor interchange (NNI) and the fixed-nodeheight-prune-and-regraft (FNPR) (?).

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

We are essentially done now. We only need to add a new monitor for the tree so that we can monitor and build the maximum a posteriori tree later.

```
monitors[3] = mnFile(filename="output/primates_mass_multiBM_tree.trees", printgen=100,
separator = TAB, psi)
```

→ A short analysis of 50 000 iterations will be sufficient.

Exercises

- Run the analysis.
- Create the maximum a posteriori (MAP) tree **readTreeTrace** and **mapTree**.
- Look at the estimated tree and compare it to the tree you used before.
- How does the posterior distribution of **sigma** looks compared with the first and second analysis?

- Compute the posterior probabilities of each tree using the `treetrace.summarize()` command.
- What is the posterior probability of the best tree?

Including information about the tree from molecular data

Starting from the model implemented in the last section, we now want to account for phylogenetic uncertainty using information contained in molecular data. As first pointed out by [Huelsenbeck and Rannala \(2003\)](#), this can easily be done in a Bayesian framework, through the use of a joint model combining sequence data and quantitative traits. Specifically:

- two data sets are loaded: one for sequence data and one for quantitative traits
- a tree is defined under birth-death process
- a Brownian model is defined over the tree (just as described in the previous section)
- the Brownian model is conditioned on the quantitative trait data
- a substitution model is defined over the same tree (GTR+Gamma)
- the substitution model is conditioned on the molecular sequence data.

Programming the model in RevBayes

First, we create a substitution model, just like what you probably did in previous tutorial (*e.g.*, RB_CTMC_Tutorial). In a first step, we will use a GTR+Gamma model. Start by loading the sequence data matrix specified in `data/primates_cytb.nex`.

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

We can use a flat Dirichlet prior density on the exchangeability rates `er` and the the base frequencies `pi`.

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

Now add the simplex scale move one each the exchangeability rates **er** and the stationary frequencies **pi** to the moves vector:

```
moves[++mvi] = mvSimplexElementScale(er)
moves[++mvi] = mvSimplexElementScale(pi)
```

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

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

The next part of the substitution process is the rate variation among sites. We will model this using the commonly applied 4 discrete gamma categories which only have a single parameter **alpha**. Let us specify the rate of **alpha** to 0.05 (thus the mean will be 20.0).

```
alpha_prior <- 0.05
```

Then create a stochastic node called **alpha** with an exponential prior:

```
alpha ~ dnExponential(alpha_prior)
```

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

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

The random variable that controls the rate variation is the stochastic node **alpha**. We will apply a simple scale move to this parameter.

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

This finishes the substitution process part of the model.

Then next part of the model is the clock model. Here we need a clock model because we work on a time tree. We use our exponentiated uniform distribution to specify a flat prior distribution which is scale invariant.

```
logClockRate ~ dnUniform(-5,5)
clockRate := 10^logClockRate
```

We will apply a sliding window move to the `logClockRate`.

```
moves[++mvi] = mvSlide(logClockRate, delta=0.1, tune=true, weight=2.0)
```

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

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

Finally we need to attach the molecular sequence data to our model.

```
seq.clamp(seqData)
```

You may have wondered how the continuous trait model and the molecular sequence model are combined. This happened automatically when you used the same tree parameter `psi` for both models. Since both models share now the same parameter, they will be used for a joint analyses. The model function thus will construct the joint model graph for the continuous trait model as well as the molecular sequence model.

After you ran the MCMC analyses, read in the tree trace. We will assume that you used a tree monitor and called the file `output/primates_joint.trees`. Change the name if you used a different one.

```
treetrace = readTreeTrace("output/primates_joint.trees", "clock")
```

Then build the maximum a posteriori tree.

```
map = mapTree( file="primates_joint.tree", treetrace )
```

Write this model and make sure that it runs when you give it to RevBayes.

Exercises

- Run the analysis.
- Using `Tracer`, visualize the posterior distribution on the rate parameter `sigma` and compare it to the previous analyses.

- Look at the MAP tree which you estimated now.

Bibliography

- De Magalhaes, J. and J. Costa. 2009. A database of vertebrate longevity records and their relation to other life-history traits. *Journal of evolutionary biology* 22:1770–1774.
- Felsenstein, J. 1981. Evolutionary trees from DNA sequences: a maximum likelihood approach. *Journal of Molecular Evolution* 17:368–376.
- Felsenstein, J. 1985. Confidence limits on phylogenies: an approach using the bootstrap. *Evolution* 39:783–791.
- Harvey, P. H. and M. D. Pagel. 1991. The comparative method in evolutionary biology vol. 239. Oxford university press Oxford.
- Huelsenbeck, J. P. and B. Rannala. 2003. Detecting correlation between characters in a comparative analysis with uncertain phylogeny. *Evolution* 57:1237–1247.