Lab 09:
Introduction to RevBayes: Phylogenetic Analysis Using Graphical Models and Markov chain Monte Carlo

By Will Freyman, edited by Carrie Tribble and Ixchel González

1 Before you begin

Please download and install **RevBayes**. RevBayes is a command line program that runs from the terminal, please make sure you can get it running before lab. You can check to see if it is running by navigating to the directory that contains the rb executable file and typing `open rb` in terminal. Another option is to double click in the executable file. Mac user, you may need to give special permission to open this software as in previous labs. Downloads are here: `https://revbayes.github.io/download`

Since we will be using sequence data that downloaded with RevBayes, please run RevBayes in the RevBayes folder you downloaded it in. I recommend to change the name of the folder for just RevBayes

2 Introduction to Graphical Models and RevBayes

Probabilistic graphical models are commonly used in statistics and machine learning, and their use is growing in all fields of science in which mathematical models are used to make inference, particularly computational biology [Jordan, 2004, Airoldi, 2007]. Graphical models are simply a different way to describe the relationships between parameters. Despite the name, graphical models can be expressed either in computer code or by drawing visual representations. By describing our models as graphical models, we are forced to be more conscious and precise about how we think different elements of the model are related to each and depend on each other. The graphical model framework provides a precise formalism for complex, parameter rich models that enable them to be constructed and utilized effectively.

There are a number of scripting languages that specialize in describing graphical models such as Stan, JAGS, and BUGS. However, the field of evolutionary biology requires a number of unique structures like phylogenetic trees and molecular data that make using more general model specification languages extremely difficult to use. The software **RevBayes** [Höhna et al., 2014] provides a new scripting language **Rev** in which users explicitly specify complex phylogenetic models. In RevBayes there are no default "black-box" models like in previous phylogenetic software; the user must fully detail the analysis they want to perform. The hope is that this will lead to improved understanding of phylogenetic models and spur the development of new models.

Both frequentist and Bayesian inference paradigms can be used with probabilistic graphical models, and RevBayes allows for both maximum likelihood and Bayesian MCMC analyses. However, inferences in high-dimensionality parameter space are usually performed in a Bayesian framework, and directed graphical models are a convenient way to represent hierarchical Bayesian models.

RevBayes is developped and mantained by a very active group of biologists and mathematicians. The versatility of the software is its main strength and many resources are available in the website: `https://revbayes.github.io`. Take a look at the tutorial page `https://revbayes.github.io/tutorials/`. This is probably one of the best resources since the tutorials explain a wide variety of analysis, from the graphical model to the implementation and examples of code.

3 Quick introduction to the Rev language

In RevBayes the **Rev** language is used to specify models. Like in R, the commands can either be typed directly into RevBayes interactively, or written in scripts that are then loaded.

Start RevBayes by opening terminal and navigating to the appropriate directory on your computer using `cd`. Now type `open rb` in the terminal window. The command prompt `>` will appear. Type

```
5 + 5
```

and hit enter. Now try

```
x = 5 + 5
x
```

If you want to load a Rev script from an external file type

```
source("my_script.Rev")
```

You'll get an error message because the file `my_script.Rev` doesn't exist. To quit RevBayes type `q()`.

The Rev language is very similar to R, however an important difference is the way variables are assigned. In R we can use either the `=` or `<-` symbols interchangeably. In Rev we use the `=` operator when defining variables that exist in the Rev workspace but are not components of the graphical model. Since graphical model variables come in three types, Rev provides three other assignment operators to specify the type of the variable. The three types are constant, stochastic, and deterministic variables. To declare constant variables (parameters that are fixed throughout the analysis) use

```
mu <- 5
sigma <- 2
```

Stochastic variables are variables that are modeled as drawn from a probability distribution. These represent both the parameters we want to infer values for and the data. To define a stochastic variable:

```
length ~ dnNormal(mean=mu, sd=sigma)
```

What is the value of `length`?

```
length
```

Deterministic variables are simply transformations of other variables:

```
long_length := length * 10
long_length
```

If we draw a new value for `length`, `long_length` will automatically be updated:

```
length ~ dnNormal(mean=mu, sd=sigma)
long_length
```

We'll see more examples of constant, stochastic, and deterministic variables and how they are used in graphical models in the examples that follow.

4 Simple non-phylogenetic example

Let's use Rev on a very trivial example to get started. We'll infer the mean height and standard deviation of a group of students. Clearly, we could analytically calculate the mean and SD, but we'll go totally overboard and set up a graphical probabilistic model and infer the parameters using MCMC.

4.1 Setting up the model

First, let's clear our workspace of all the variables we've been messing around with:

```
clear()
```

Now enter our raw data (the observed heights), a constant vector of values (in feet):

```
data <- [5.5, 5.1, 6.0, 5.9, 6.2, 5.2, 6.1, 6.3]
```

Let's get the number of students:

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

The mean and SD are stochastic variables we want to infer. We will draw them from highly uninformative uniform priors:

```
mu ~ dnUniform(0, 10)
sigma ~ dnUniform(0, 5)
```

We will model our observed heights (the raw data) as being drawn from a normal distribution. To attach our observed data to the stochastic variable we **clamp** the observation to the variable. We'll do this in a loop for each observed height:

```
for (i in 1:n) {
    heights[i] ~ dnNormal(mean=mu, sd=sigma)
    heights[i].clamp(data[i])
}
```

Now we can finalize the model by saving it as a workspace variable:

```
mymodel = model(heights)
```

If you type `mymodel` details about the model you have specified and how the nodes are all connected will be output to the screen. To view details on a single node type

```
str(mu)
```

4.2 Performing an MCMC analysis

Now let's perform inference on our model. To use MCMC we have to declare the moves that the MCMC sampler will take to explore parameter space. There are many moves available in RevBayes, here will will just use a simple slide move. We will save our moves in a vector:

```
moves[1] = mvSlide(mu, delta=0.001)
moves[2] = mvSlide(sigma, delta=0.001)
```

To get details on what the `mvSlide` function does, type `?mvSlide`. Just like in R, you can type `?` to get info on any command.

We will also need to set up some monitors for our MCMC analysis. The monitors will basically watch the MCMC and keep samples from it. We'll set up a log file for the entire model, and also a screen monitor so we can view progress on our screen:

```
monitors[1] = mnModel(filename="output/heights.log", printgen=10)
monitors[2] = mnScreen(mu, sigma, printgen=10)
```

Finally, let's setup our MCMC object and run it. For such a simple model 10000 iterations should be enough:

```
mymcmc = mcmc(mymodel, monitors, moves)
mymcmc.run(generations=10000)
```

After the run we can check to see how efficiently the MCMC explored parameter spacer by looking at the operator summary:

```
mymcmc.operatorSummary()
```

This tells us how many moves were proposed and their acceptance rates.

Exercise 1:

1. Take a look at `heights.log` in Tracer. Do the estimates of `mu` and `sigma` seem right? Did they converge?
2. What does the `delta` argument in the `mvSlide` function do?
3. Set `delta=1.0` for each of the moves, and rerun the analysis. How do the estimates look now? Did they converge? What are your ESS values for `mu` and `sigma`?

There are many moves, distributions, and functions that you can use to build more complex models. You can see a list of all available commands by typing `ls(all=TRUE)`.

Exercise 2:
In the model above we used highly uniformative uniform priors on `mu` and `sigma`. The average height in the United States is 5.4 feet and the SD is about 0.23. Modify the code above to use a more informative prior for `mu`. Don't just set it to be constant, use a different prior distribution that is more informative than the uniform prior.

1. Rerun the analysis using the informative prior. Show me the line of code you used to set an informative prior for `mu`.
2. Did this affect the final estimates of `mu` and `sigma`? Which estimate is closer to the actual mean? What does this tell you about using informative priors?

5 Phylogenetic graphical models

Let's run an actual phylogenetic analysis now. Though you can specify very complex phylogenetic models in RevBayes, we'll cover only the basics today: inferring trees using Jukes-Cantor and GTR. Much of this exercise comes from the excellent tutorials you can find at `http://www.revbayes.com`.

5.1 Loading the data

First clear the workspace again:

```
clear()
```

Let's load an alignment of the cytochrome B gene for a number of primate species:

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

Take a look at some information about the alignment:

```
data
```

We'll need some useful information from the alignment:

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

Now we'll set up a counter for our `moves` vector:

```
mvi = 0
```

5.2 Juke-Cantor substitution model

Remember DNA substitution models are continuous-time Markov chains. Different models are defined by their instantaneous-rate matrix Q. DNA data has 4 discrete states, so we'll set up the Q matrix like this:

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

Check out the Q matrix:

```
Q
```

5.3 Tree prior

Here we will use simple priors that are similar to the defaults used in a MrBayes analysis. We'll use a uniform tree prior that puts equal probability on all unrooted, fully resolved topologies. Remember, `topology` is simply a discrete random variable in our model.

```
topology ~ dnUniformTopology(taxa=taxa)
```

Now we'll add MCMC moves for the topology. The tree rearrangement moves, nearest-neighbor interchange (NNI) and subtree pruning and regrafting (SPR) should be familiar to you:

```
moves[++mvi] = mvNNI(topology, weight=3.0)
moves[++mvi] = mvSPR(topology, weight=3.0)
```

The weight specifies how often the move will be proposed relative to all other moves.

Now create a vector of branch lengths. We'll draw each branch length from an exponential distribution. For a long time this was the default branch length prior in MrBayes (but see Rannala et al. [2012] for why the default is now compound Dirichlet priors). We'll also add scaling moves for each branch length.

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

Simply for convenience, let's add a deterministic node to monitor tree length:

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

Finally, we combine the topology and the branch length vector into a deterministic node that represents our phylogeny:

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

5.4 Phylogenetic continuous-time Markov chain model

Our sequence evolution model is a continuous-time Markov chain (CTMC) over a phylogeny. So we pass the Jukes-Cantor rate matrix `Q` and the `phylogeny` into the phylogenetic CTMC distribution:

```
seq ~ dnPhyloCTMC(tree=phylogeny, Q=Q, type="DNA")
```

And now we clamp our sequence data to the CTMC:

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

We have fully defined our model, so now we wrap it up and declare it complete:

```
mymodel = model(seq)
```

I passed `seq` to the `model` function, but any node in the model could have been used.

5.5 Running the MCMC analysis

We need some monitors:

```
monitors[1] = mnModel(filename="output/primates_JC.log", printgen=10)
monitors[2] = mnFile(filename="output/primates_JC.trees", printgen=10, phylogeny)
monitors[3] = mnScreen(printgen=100, TL)
```

Note the file monitor (`monitors[2]`) saves all the sampled trees into a single file. Set up the MCMC and run it:

```
mymcmc = mcmc(mymodel, monitors, moves)
mymcmc.run(generations=30000)
```

Now, we will analyze the tree sampled by the MCMC. Read in the tree trace:

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

and let's summarize the trees into a single maximum a posteriori (MAP) tree:

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

Open up the tree in FigTree and view the posterior probabilities for each clade. Reroot the tree so that your outgroup is *Galeopterus variegatus*.

Exercise 3:

1. In a text editor, modify the code above to use the GTR model instead of Jukes-Cantor. I describe how to set up the GTR model below. Save the script in a file called `primates_GTR.Rev`. Send me a copy of your working script.
2. Run your script by calling `source("primates_GTR.Rev")` in RevBayes. Open the final MAP tree and properly reroot it. Do the posterior probabilities, topology, and/or branch lengths differ from the Jukes-Cantor tree?
3. Send me both of your MAP tree files.

5.6 The GTR model

We've discussed the General Time Reversible (GTR) substitution model before, which allows all four stationary frequencies and all six exchangeability rates to differ.

To use the GTR model, we define a variable `pi` for the stationary frequencies that are drawn from a flat, uninformative Dirichlet distribution:

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

Since `pi` is a stochastic variable that we want to estimate, we need to set up a MCMC move for it:

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

The mvSimplexElementScale randomly changes one element of the simplex and then rescales the other elements so that they sum to 1 again.

We will also use an uninformative Dirichlet distribution on the six exchangeability rates:

```
er_prior <- v(1,1,1,1,1,1)
er ~ dnDirichlet(er_prior)
moves[++mvi] = mvSimplexElementScale(er)
```

Finally, we'll pass the exchangeability rates and the stationary frequencies into a function to create the rate matrix Q:

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

Please email me the following:

1. The answers to exercises 1-3.
2. Your primates_GTR.Rev script.
3. Your Jukes-Cantor and GTR MAP trees.

References

Edoardo M Airoldi. Getting started in probabilistic graphical models. *PLoS Comput Biol*, 3(12):e252, 2007.

Sebastian Höhna, Tracy A Heath, Bastien Boussau, Michael J Landis, Fredrik Ronquist, and John P Huelsenbeck. Probabilistic graphical model representation in phylogenetics. *Systematic biology*, 63(5):753–771, 2014.

Michael I Jordan. Graphical models. *Statistical Science*, pages 140–155, 2004.

Bruce Rannala, Tianqi Zhu, and Ziheng Yang. Tail paradox, partial identifiability, and influential priors in bayesian branch length inference. *Molecular biology and evolution*, 29 (1):325–335, 2012.