Phylogenetic Inference using RevBayes

The comparative method beyond Brownian motion

Sebastian Höhna

This tutorial is currently under construction/revision.

1 Introduction

Throughout this tutorial, we have exclusively considered undirected Brownian models. However, many other models could be used, and this, both for quantitative traits and for substitution rates or substitution parameters. Right now, there are at least two other models available in RevBayes: the Brownian model with systematic trend and the Ornstein-Uhlenbeck process.

2 Data and files

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

- **primates_cytb.nex**: Alignment of the *cytochrome b* subunit from 23 primates representing 14 of the 16 families (*Indriidae* and *Callitrichidae* are missing).
- **primates_lhtlog.nex**: 2 life-history traits (endocranial volume (ECV), body mass; each for males and females separately) for 23 primate species (taken from the Anage database, De Magalhaes and Costa 2009). The traits have been log-transformed.
- **primates.tree**: A time calibrated phylogeny of the same 23 primates.

3 Ornstein-Uhlenbeck process

First, load the trait data:

```
contData <- readContinuousCharacterData("data/primates_lhtlog.nex")
```

If you type you will see that the continuous character data matrix contains several characters (columns).

```
contData

   Continuous character matrix with 23 taxa and 11 characters
   ==========================================================
   Origination:                 primates_lhtlog.nex
   Number of taxa:              23
   Number of included taxa:     23
   Number of characters:        11
   Number of included characters: 11
   Datatype:                    Continuous
```

Since we only want the body mass (of females) we exclude all but the third character

```
contData.excludeAll()
contData.includeCharacter(3)
```

Next, load the time-tree from file. Remember that we use in this first simple example a fixed tree that we assume is known without uncertainty.

```
treeArray <- readTrees("data/primates.tree")
psi <- treeArray[1]
```

→ You may want to look at this tree before by loading the **`primates.tree`** in FigTree or any other tree visualization software.

As usual, we start be initializing some useful helper variables. For example, we set up a counter variable for the number of moves that we already added to our analysis. This will make it much easier if we extend the model or analysis to include additional moves or to remove some moves.

```
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:

```
logAlpha ~ dnUniform(-10,10)
alpha := 10^logAlpha

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(logAlpha,delta=10,tune=true,weight=2)
moves[++mvi] = mvSlide(logSigma, delta=1.0, tune=true, weight=2.0)
```

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()
```

We will use a uniform prior distribution on the logarithm of the root mass optimal value.

```
logOptim ~ dnUniform(-10,10)
```

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(logOptim,delta=10,tune=true,weight=2)
```

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] := logOptim
```

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 Ornstein-Uhlenbeck process along the tree
for (i in (numNodes-1):(numTips+1) ) {
  logmass[i] ~ dnOrnsteinUhlenbeck( x0=logmass, theta=logOptim, alpha=alpha, sigma=sigma, time=
      psi.branchLength(i) )
  # moves on the Ornstein-Uhlenbeck 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] ~ dnOrnsteinUhlenbeck( x0=logmass, theta=logOptim, alpha=alpha, sigma=sigma, time=
      psi.branchLength(i) )

  # condition OU model on quantitative trait data (second column of the dataset)
  logmass[i].clamp(contData.getTaxon(psi.nodeName(i))[1])
}
```

The model is now entirely specified and we can create a model object containing the entire model graph by providing it with only one of our model variables, *e.g.,***`sigma`**.

```
mymodel = model(sigma)
```

To see what it happing during the MCMC let us make a screen monitor that tracks the rate **`sigma`**.

```
monitors[1] = mnScreen(printgen=10, sigma, alpha, logOptim)
```

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_OU.log", printgen=10, separator = TAB)
monitors[3] = mnExtNewick(filename="output/primates_mass_OU_ext.trees", isNodeParameter=TRUE,
    printgen=10, separator = TAB, tree=psi, logmass)
```

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)
```

To get the annotate tree we use the map tree function.

```
treetrace = readTreeTrace("output/primates_mass_OU_ext.trees", treetype="clock")
map_tree = mapTree(treetrace,"output/primates_mass_OU_ext_MAP.tree")
```

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 **`logOptim`** 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?

4 Branch-rate jump process

We start with specifying a probability for a jump to occur at a given branch.

```
rho <- 0.01
```

Next, we specify the prior distribution on the root value.

```
logRootOptim ~ dnUniform(-10,10)
moves[++mvi] = mvSlide(logRootOptim,delta=10,tune=true,weight=2)
```

Now, we use a loop over all branches to specify the per branch optimal value.

```
for (i in numBranches:1) {
   optimChangeProbability[i] := Probability(1-rho) # + (1-exp(-lambda*psi.branchLength(i)))
   optimMultiplier[i] ~ dnReversibleJumpMixture(1, dnGamma(2,2), optimChangeProbability[i] )
   if ( psi.isRoot( psi.parent(i) ) ) {
      nodeOptim[i] := logRootOptim * optimMultiplier[i]
   } else {
      nodeOptim[i] := nodeOptim[psi.parent(i)] * optimMultiplier[i]
   }
   optimChange[i] := ifelse( optimMultiplier[i] == 1, 0, 1 )
   moves[++mvi] = mvRJSwitch(optimMultiplier[i], weight=1)
   moves[++mvi] = mvScale(optimMultiplier[i], lambda=0.1, tune=true, weight=1)
}
```

Only for monitoring purposes we add a variable that counts the current number of jumps.

```
numOptimChanges := sum( optimChange )
```

You may want to monitor and then visualize the branch-specific jumps

```
monitors[4] = mnExtNewick(filename="output/primates_mass_OU_rate_jumps.trees", isNodeParameter=
    FALSE, printgen=10, separator = TAB, tree=psi, nodeOptim, optimChange)
```

References

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.

Version dated: July 10, 2016