Intro to OUCH

Marguerite A. Butler[1,2]

[1]Department of Biology, University of Hawaii, Honolulu, HI 96822
[2]mbutler@hawaii.edu

August 11, 2016

Contents

Chapter 1

Verification: Computing Phylogenetic GLS "by hand"

Chapter Topics:

- Checking your computations by other means
- Deconstructing `ape` objects

Skills: accessing object elements, constructing similarity matrices, using R matrix math functions, using R linear model functions.

A good practice is to try to verify the software you are using by doing things another way. Phylo GLS is a good one because it is fairly simple mathematically. All we need to do is take our data and, using matrix math, divide by the square root of the phylogenetic covariance matrix. Thus we need to do the following steps:

1. Compute the phylogenetic covariance matrix expected under BM. This is a "similarity" matrix based on the amount of time they share along the phylogeny. Let's call this `tbm`
2. Take the inverse of the matrix, `tbmi`.
3. Find the square root of `tbmi`, a good way to do this is by cholesky decomposition.
4. Multiply our data vectors by `roottbmi` to get our phylogenetically transformed data.
5. Run regression analysis on the transformed data.

Let's make up two variables `X` and `Y`:

```
> X <- c(4.09434, 3.61092, 2.37024, 2.02815, -1.46968)
> Y <- c(4.74493, 3.33220, 3.36730, 2.89037, 2.30259)
> names(X) <- names(Y) <- c("Homo", "Pongo", "Macaca", "Ateles", "Galago")
```

Let's load our primate tree in newick format:

```
> require(ape)     # a comparative methods package written by Emanuel Paradis
> require(nlme)   # non-linear mixed effects model, has gls function
> tree <- read.tree(text="((((Homo:0.21,Pongo:0.21):0.28,Macaca:0.49):
+ 0.13,Ateles:0.62):0.38,Galago:1.00);")
> names(tree)
```

```
[1] "edge"        "Nnode"       "tip.label"   "edge.length"
```

```
> tree$edge
```

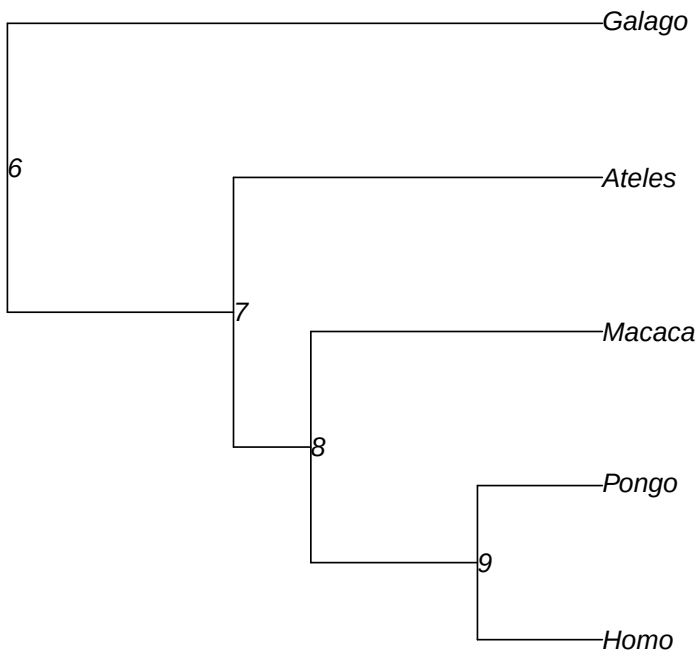
```
     [,1] [,2]
[1,]    6    7
[2,]    7    8
[3,]    8    9
[4,]    9    1
[5,]    9    2
[6,]    8    3
[7,]    7    4
[8,]    6    5
```

```
> tree$tip.label
```

```
[1] "Homo"   "Pongo"  "Macaca" "Ateles" "Galago"
```

Remember that the first column in the edge matrix is the ancestral node, and the second column is the descendant node. If you think of the descendant node as the reference point, then the tips are nodes 1-5, and the internal nodes are therefore 6-9. We can assign the node.labels and plot them on the tree:

```
> tree$node.label <- c(6:9)
> plot(tree, show.node.label=TRUE)
```



Let's make a dataframe so that we can see the tree structure and associated metadata more clearly. We're using the function `with`, which defines a small local environment where we are using the object `tree`.

Let's `cbind` together the `edge` matrix, which describes the ancestor-descendant pairs, with the appropriate branch lengths, and species and tip labels. Note that the edge matrix doesn't have a row for the most basal node in the tree as a descendant, so we have to drop the first node label from our vector (otherwise the label vector will be too long by one).

```
> with( tree, cbind(tree$edge, edge.length,
+ labels=c(node.label[-1], tip.label)))
```

```
     edge.length labels
[1,] "6" "7" "0.38"      "7"
[2,] "7" "8" "0.13"      "8"
[3,] "8" "9" "0.28"      "9"
[4,] "9" "1" "0.21"      "Homo"
```

```
[5,] "9" "2" "0.21"      "Pongo"
[6,] "8" "3" "0.49"      "Macaca"
[7,] "7" "4" "0.62"      "Ateles"
[8,] "6" "5" "1"         "Galago"
```

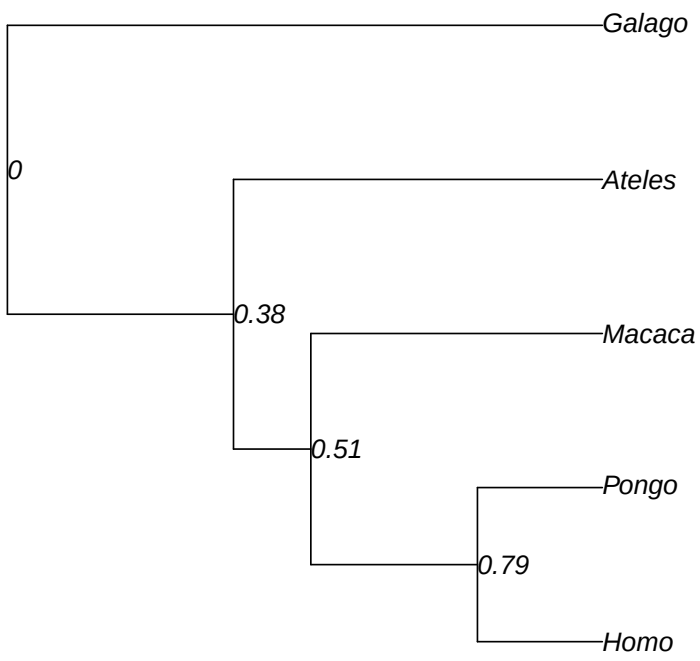
Since we need to calculate a similarity matrix based on time in shared evolutionary history, it would actually be much more convenient to have for each node, the time from the root to the node. Let's call this "times". For a small and simple phylogeny, we can do this by "hand". Looking at the dataframe we just made, we can add up the branches to each node, with the base of the tree at time zero and the tips at time 1, and then remake the matrix (again, we drop the basal node from this vector):

```
> times <- c(0, .38, .38+.13, .38+.13+.28, 1, 1, 1, 1, 1)
> with( tree, cbind(tree$edge, edge.length,
+ labels= c(node.label[-1], tip.label), times=times[-1]))
```

```
             edge.length labels   times
[1,] "6" "7" "0.38"      "7"      "0.38"
[2,] "7" "8" "0.13"      "8"      "0.51"
[3,] "8" "9" "0.28"      "9"      "0.79"
[4,] "9" "1" "0.21"      "Homo"   "1"
[5,] "9" "2" "0.21"      "Pongo"  "1"
[6,] "8" "3" "0.49"      "Macaca" "1"
[7,] "7" "4" "0.62"      "Ateles" "1"
[8,] "6" "5" "1"         "Galago" "1"
```

Looks good so far. Let's plot these "times" on the tree. Replace the node.label with times values and replot:

```
> tree$node.label <- as.character(times[1:4])
> plot(tree, show.node.label=TRUE)
```



Now, looking at the tree, let's make our tbm matrix. First, make an empty matrix with just species names to help us with a mental image of what we're doing:

```
> tbm <- matrix(nrow = 5, ncol = 5)   # could also use length(tree$tip.label)
>                             # instead of 5
> rownames(tbm) <- colnames(tbm) <- tree$tip.label
> tbm
```

```
       Homo Pongo Macaca Ateles Galago
Homo     NA    NA     NA     NA     NA
Pongo    NA    NA     NA     NA     NA
Macaca   NA    NA     NA     NA     NA
Ateles   NA    NA     NA     NA     NA
Galago   NA    NA     NA     NA     NA
```

Now look at the phylogeny and fill in the pairwise similarities by the amount of evolutionary history they share:

```
> tbm[1, ] <- c(1, .79, .51, .38, 0)
> tbm[2, ] <- c(.79, 1, .51, .38, 0)
> tbm[3, ] <- c(.51, .51, 1, .38, 0)
> tbm[4, ] <- c(.38, .38, .38, 1, 0)
> tbm[5, ] <- c(rep(0, 4), 1)
> tbm
```

```
       Homo Pongo Macaca Ateles Galago
Homo   1.00  0.79   0.51   0.38      0
Pongo  0.79  1.00   0.51   0.38      0
Macaca 0.51  0.51   1.00   0.38      0
Ateles 0.38  0.38   0.38   1.00      0
Galago 0.00  0.00   0.00   0.00      1
```

Of course, one could program an automated way to do this, but that would take a lot of time and skill. But this is a verification, so we compute it by hand. Now we are ready to begin transforming our data. First do the inversion and root of tbm using the `solve()` and `chol()` functions, respectively. These are both part of the `base` package.

```
> tbmi <- solve(tbm)
> roottbmi <- chol(tbmi)
```

Now transform our data. For the regression model, we will also have to transform the intercept, so we bind a column of 1's with the X variable (independent variable in this example) and make transformed variables Z and U. Note that matrix multiplication is designated by %*% (Otherwise multiplication is element-by-element, the default in R) :

```
> Z <- roottbmi %*% Y
> U <- roottbmi %*% cbind(1, X)
> colnames(U) <- c("1", "X")
```

(Feel free to look at any of these matrices we're creating). Now we just have to run the regression. Since the intercept term is inside the X matrix now, we do a regression without an intercept (one is added automatically unless you say no). You can use either lm or gls in this case because we don't have to specify a correlation structure.

```
> summary(lm(Z ~ U -1))
```

```
Call:
lm(formula = Z ~ U - 1)
```

```
Residuals:
   Homo   Pongo  Macaca  Ateles  Galago
 1.7362 -0.6636  0.0303 -0.4857  0.4373

Coefficients:
   Estimate Std. Error t value Pr(>|t|)
U1   2.5001     0.7755   3.224   0.0484 *
UX   0.4319     0.2865   1.508   0.2288
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.138 on 3 degrees of freedom
Multiple R-squared:  0.8746,        Adjusted R-squared:  0.791
F-statistic: 10.46 on 2 and 3 DF,  p-value: 0.04442
```

```
> summary(gls(Y ~ X, correlation=corBrownian(phy=tree), data=data.frame(X,Y)))

Generalized least squares fit by REML
  Model: Y ~ X
  Data: data.frame(X, Y)
       AIC      BIC   logLik
  17.48072 14.77656 -5.74036

Correlation Structure: corBrownian
 Formula: ~1
 Parameter estimate(s):
numeric(0)

Coefficients:
                Value Std.Error  t-value p-value
(Intercept) 2.5000672 0.7754516 3.224014  0.0484
X           0.4319328 0.2864904 1.507669  0.2288

 Correlation:
  (Intr)
X -0.437

Standardized residuals:
      Homo      Pongo     Macaca     Ateles     Galago
 0.4187373 -0.6395037 -0.1376075 -0.4269456  0.3844060
attr(,"std")
[1] 1.137666 1.137666 1.137666 1.137666 1.137666
attr(,"label")
```

```
[1] "Standardized residuals"

Residual standard error: 1.137666
Degrees of freedom: 5 total; 3 residual
```

Let's compare that with the phylogenetic GLS, and with the regression from PIC:

```
> pic.X <- pic(X, tree)
> pic.Y <- pic(Y, tree)
> summary(lm( pic.Y ~ pic.X -1))


Call:
lm(formula = pic.Y ~ pic.X - 1)

Residuals:
       0     0.38     0.51     0.79
-0.55351  0.35263  0.03311  1.85770

Coefficients:
      Estimate Std. Error t value Pr(>|t|)
pic.X   0.4319     0.2865   1.508    0.229

Residual standard error: 1.138 on 3 degrees of freedom
Multiple R-squared:  0.4311,        Adjusted R-squared:  0.2414
F-statistic: 2.273 on 1 and 3 DF,  p-value: 0.2288
```

We can also calculate the correlated evolution in the two characters, the fundamental problem that Independent Contrasts were designed for.

```
> cor(pic.Y, pic.X)


[1] -0.5179156
```

Chapter 2

Stochastic Models in Cartoons

Let?s make some graphical animations to illustrate the BM and OU model. We have all the tools to do this in R, to help us understand the differences between the models and how we can exploit their features to explore evolutionary scenarios.

2.1 Brownian motion model

We can describe a Brownian motion process using the following differential equation:

$$dX(t) = \sigma\, dB(t). \tag{2.1}$$

This equation says that the change in value of X in some small interval in time by an amount of change $dB(t)$ (a draw from a Normal distribution) times the BM scaling factor σ (often called the 'rate of evoluiton' - this is a bit confusing, of course because each different model produces different rates of evolution, just be aware).

That is, over a small increment of time, $dX(t)$ is the infinitesimal change in the character X over the infinitesimal interval from time t to time $t + dt$. The term $dB(t)$ is ?white noise?; that is, the random variables $dB(t)$ are independent and identically-distributed normal random variables, each with mean zero and variance dt.

We can mimic this behavior by a simulation in discrete time. If we take the above equation and approximate it as:

$$\Delta X = X_{i+1} - X_i = \sigma\, dB(t) \tag{2.2}$$

$$X_{i+1} = X_i + \sigma\, dB(t) \tag{2.3}$$

By looking at the difference equation above, we can see that it is possible to build a simulation by simply taking the old value of X_i and adding a random amount of change

to it in order to get the next value (X_{i+1}), scaled (multiplied) by a constant value σ. If $\sigma = 1$, then it is in its most simplest form and we can code this simply as:

```
>   x[i+1] <- x[i] + devs[i]
```

Where `devs` is a random deviate. So let's do it! Start with 100 random draws.

```
> nsteps = 100       # number of steps in our simulation
> devs =rnorm(nsteps)    # 100 draws from a normal distribution
```

Create a vector for our phenotype `x`, and fill it by starting from 0, and adding a random deviate to `x` at each generation.

```
> x <- c(0:100)  # initialize x (create a vector of the appropriate length)
> for (i in 1:nsteps)  # a loop which will run ngens times
+ {
+    x[i+1] <- x[i] + devs[i]   # add random draw to old value of phenotype
+ }
```

```
> x   # take a look at the values of x, this is how x changes each generation
```

```
  [1]  0.000000  1.727290  2.840496  1.736700  1.265043  1.669334  2.818793
  [8]  4.606562  3.956888  4.560257  3.965367  4.302249  5.630431  5.648792
 [15]  4.939006  4.721786  5.336900  6.354243  7.413156  6.954242  7.365918
 [22]  9.516711  9.909210  9.424690 10.394081 10.230222  8.825981  9.676764
 [29] 11.115194 11.474203 13.412849 13.647906 14.037570 16.181312 15.535814
 [36] 15.185207 15.550010 16.219923 17.373443 17.806264 18.219638 17.066978
 [43] 17.931207 19.709855 22.167118 21.532103 21.226264 21.080697 21.728541
 [50] 21.781476 22.931787 23.864783 21.910217 21.611462 22.950472 22.870891
 [57] 22.998618 22.173252 22.087638 23.313955 22.094916 22.384510 21.945206
 [64] 21.949402 22.064481 23.466016 23.369648 24.826797 26.019992 27.048051
 [71] 28.331002 28.337722 27.977713 29.749190 29.454352 28.496931 29.664016
 [78] 29.401881 27.404863 27.994271 29.416431 29.964027 28.631371 30.885919
 [85] 31.062889 31.234908 31.466356 33.248181 33.048893 33.341874 33.409294
 [92] 34.142084 34.230421 33.635506 34.901866 35.851001 34.214798 35.246873
 [99] 36.073665 33.533991 32.425709
```

We can plot this single random walk. The function `plot()` creates a new plot window (and plots data, except that we told it to plot nothing by the parameter `type="n"` ? we did this in order to plot the simulation slowly). The actual simulations are added one generation at a time by the command `lines()`.

```
> plot(1:length(devs), devs, type = "n", col="red",
+ ylim = c(-max(devs)*30, max(devs)*30),
+ xlab="Time", ylab="Value", main="BM Simulation")
> x <- c(0:100)
> for (i in 1:nsteps)
+ {
+    x[i+1] <- x[i] + devs[i]
+    lines(i:(i+1), x[i:(i+1)], col="red")
+ }
```

We can add the Brownian scaling parameter σ by modifying the line of code specifying the BM:

```
> x[i+1] <- x[i] + sigma * devs[i]
```

In order to do 100 random walks (lineages), we need to place an outer loop, once for each random walk:

```
> bm.plot <- function( sigma=1, ngens=100, nlineages=100, yylim=c(-50, 50) )
+ {
+     plot(1:length(devs), devs, type = "n", col="red",
+     ylim = c(-max(devs)*30, max(devs)*30),
+     xlab="Time", ylab="Value", main="BM Simulation")
+
+     for (i in 1:nlineages)       # number of lineages to simulate
+     {
+         x <- c(0:100)
+         devs =rnorm(ngens)    # draws from a normal distribution each gen
+         for (i in 1:ngens)
+         {
+            x[i+1] <- x[i] + sigma*devs[i]   # BM equation
+                # step through time, increasing x a little bit each time
+            lines(i:(i+1), x[i:(i+1)], col="red")   # plot line segment
+         }
+     }
+ }
```

Now run the simulation:

```
> bm.plot()
```

Now that we have created a function, we can play with it by varying the parameters to see what they mean. For example try: `bm.plot( sigma=4 )` see the difference?

Compare with:

```
> bm.plot( sigma=1, ngens=300 )
```

For the following exercises, it will be helpful to collect the final values of each lineage. To do this, we need to (1) collect the final values, and (2) return them from the function. We can do this by adding in the lines for `xfinal` below:

```
> bm.plot <- function( sigma=1, ngens=100, nlineages=100, yylim=c(-50, 50) )
+ {
+     plot(1:length(devs), devs, type = "n", col="red", ylim = yylim,
+      xlab="Time", ylab="Value", main="BM Simulation")
+      xfinal <- c(1:nlineages)  ### initialze final values
+
+      for (j in 1:nlineages) # number of lineages to simulate
+      {
+          x <- c(0:ngens)
+          devs =rnorm(ngens) # draws from a normal distribution, one per generation
+         for (i in 1:ngens)
+         {
+             x[i+1] <- x[i] + sigma*devs[i]     # BM equation
+             # step through time, increasing x a little bit each time
+             lines(i:(i+1), x[i:(i+1)], col="red")  # plot line segment
+          }
+          xfinal[j] <- x[ngens+1]     ### collect the last value in each lineage
+      }
+     return(xfinal)     ### return the final values
+ }
```

2.2 R tricks – feel free to skip

The loop is easier to understand in terms of a stochastic process, but actually we can write this code much more compactly:

Adding up a series of BM steps using the cumulative sum function:

```
>     sigma=1
>     cumsum(rnorm(nsteps, sd=sigma))
```

Plotting all the line segments at once:

```
>     y <- c(0, cumsum(rnorm(nsteps, sd=sigma)) )
>     lines(0:nsteps, y)
```

Or even more compactly:

```
>    sigma=1
>    lines(0:nsteps, c(0, cumsum(rnorm(nsteps, sd=sigma))))
```

And doing all of the lineages using lapply:

```
> bm.plot <- function( sigma=1, nsteps=100, nlineages=100, yylim=c(-50, 50))
+ {
+ # Set up plotting environment
+         plot(0, 0, type = "n", xlab = "Time", ylab = "Trait",
+                 xlim=c(0, nsteps), ylim=yylim)
+
+ # Draw random deviates and plot
+         lapply( 1:nlineages, function(x)
+           lines(0:nsteps, c(0, cumsum(rnorm(nsteps, sd=sigma)))))
+ }
```

If you want to show the simulations to screen, then you may actually prefer to do the slower for-loops, as the lapply is too fast.

2.3 BM Exercises

1. Go back to `bm.sim.slow` and modify it to an OU. Recall the OU equation:

$$dX(t) = \alpha\,(\theta - X(t))\,dt + \sigma\,dB(t). \tag{2.4}$$

   Hint: You will need to make two new parameters.

2. Introduce a branch to either the BM or OU simulation. You will need to simulate a single lineage for half the time, then two lineages for the rest of the time.

2.4 The OU Process

Recall that we described a Brownian motion process using the following equation:

$$dX(t) = \sigma\,dB(t). \tag{2.5}$$

The simplest stochastic model that incorporates selection is the Ornstein-Uhlenbeck process Uhlenbeck and Ornstein (1930):

$$dX(t) = \alpha\ (\theta\ - X(t))dt + \sigma\, dB(t). \tag{2.6}$$

Equation 2.6 adds two new parameters to our changing phenotype. The parameter ? measures the strength of selection. When ? = 0, the deterministic part of the OU model drops out and 2.6 collapses to the familiar BM model of pure drift.

The OU process was first described in 1930. Russ Lande showed that the evolution of a species? mean phenotype could be described by an OU process, representing stabilizing selection (Lande, 1976), and was discussed by Felsenstein (1988). Thomas Hansen first recognized its usefulness in phylogenetic comparative analysis (Hansen, 1997) to describe adaptive evolution along a phylogeny. Butler and King developed this application further (Butler and King, 2004) and expanded upon the use of the OU model to represent alternative models of adaptive and non-adaptive evolution, as well as a rigorous model-comparison framework.

We can again write a simple simulation by copying the form of the equation, with parameters for α and θ:

```
>           x[i+1] = alpha*(theta-x[i])+x[i] + sigma*devs[i]
```

OK! Let?s simulate the OU process:

```
> ou.plot <- function( alpha=0.005, theta=0, sigma=1, ngens=100, nwalks=30,  yylim=c(-
+ {
+      plot(1:length(devs), devs, type = "n", col="red", ylim = yylim,
+      xlab="Time", ylab="Value", main="OU Simulation")
+      xfinal <- c(1:nwalks)
+
+      for (j in 1:nwalks)   ### number of lineages to simulate
+      {
+          x <- c(0:ngens)
+          devs =rnorm(ngens)
+          for (i in 1:ngens)
+          {
+              x[(i+1)] = alpha*(theta-x[i])+x[i] + sigma*devs[i]
+              lines(i:(i+1), x[i:(i+1)], col="red") ### plot line segment
+          }
+          xfinal[j] <- x[ngens+1]    ### collect last value at each lineage
+      }
+      return(xfinal)    ### return the final values
+ }
```

You can run the function by typing:

```
> ou.plot( theta=30 )

 [1]  5.858906 11.593035  5.570329 16.736155 13.635151 10.245383 13.805306
 [8] 18.498839  5.008840 10.928479  7.975263  6.158527  7.652482 18.515890
[15]  9.504363  7.826320 21.983405 20.160912 15.451889 29.106171 13.068116
[22]  3.750659 15.140060 11.167322 18.781672  8.828107  1.674351 13.131380
[29] 14.674974  1.228152
```

2.5 OU Exercises

1. What is the effect of bigger or smaller α? Does the simulation always reach the optimum value you set? What else could you change?
2. What is the effect of changes in θ?
3. What is the difference between BM simulations and OU simulations with similar σ? Try your own or do a BM simulation with $\sigma = 1$, and OU with α=1,$\alpha = 0.01$, and $\theta = 0$.

2.6 Using the OU to model adaptive evolution

Simpson (1953) painted a vivid picture of adaptive evolution in macroevolutionary time. He imagined lineages evolving through time within an adaptive zone. On rare occasions, the lineage may enter a new adaptive zone, which would result in morphological adaptations that reflect their new functional demands. In some cases, entering a new adaptive zone may even result in special bursts of evolution, perhaps adaptive radiation. However, we are getting ahead of ourselves.

We can use the OU to model adaptive evolution. Not only does it have a term for selection, but also for adaptive optima. These adaptive optima can be used to represent selective regimes, or lifestyles, or different suites of ecological pressures. While we limit each branch of the phylogeny to having only one optima, we can assign different optima to different branches, according to our biological hypothesis. For example, we may have a lizard lineage that is ancestrally insectivorous, but they may have evolved herbivory along some of its branches (Fig. 2.1)

Note that we don?t assign a unique optimum to every branch, that would result in too many parameters and not enough data to estimate them unambiguously. Rather, we are postulating a limited number of adaptive regimes ? the simplest model that can explain most of the data.

Download the function `OU.sim.branch` and load it into your R workspace by the following command (make sure you download it to your working directory):

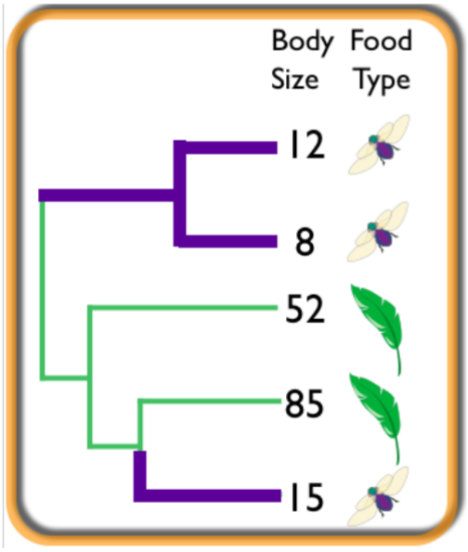


Figure 2.1: Hypothetical adaptive regimes. We group branches leading to herbivory in a separate adaptive regime relative to those leading to insectivory.

```
> source("OU.sim.branch.R")
```

If you?d like to see the function, just type it?s name:

```
> OU.sim.branch
```

and the function will dump to screen. In particular look at the default parameters in the function call and do the exercises below.

2.7 OU Exercises 2

Is the OU process, as simple as it is, flexible enough to suit our needs? Let?s try some simulations, this time with a branch along the phylogeny, and see if we can simulate changes in the phenotype.

1. Can you make the phenotype diverge in different branches of the phylogeny?
2. Food for thought. Does the simulation always reach the optimum? What are the possible explanations?

2.8 Making movies

In order to make a movie of the plot, you will need to save a series of plots as separate graphics files, similar to the ”flip-books” you played with as a child. You need to make a plot with the first lineage, then the first two lineages, then the first three lineages, and so on.

So it would make sense to make a matrix to store the lineages, then plot through cumulatively:

```
> nsteps=100
> nlineages=30
> sigma=1
> sims <- sapply(1:nlineages, function(x) c(0, cumsum(rnorm(nsteps, sd=sigma))))
> yylim <- c(-30, 30)
> png(filename="movies/Rplot%03d.png")
>               # turn on png graphical device (write to file)
> for (i in 2:nlineages)
+ {
```

```
+         plot(0, 0, type = "n", xlab = "Time", ylab = "Trait",
+         xlim=c(0, nsteps), ylim=yylim)
+         apply( sims[,1:i], 2 , function(x) lines(0:nsteps, x, col="red"))
+ }
> dev.off()      # turn off png
```

Then in a terminal, move into the `movies` directory. If you have imagemagik installed:

convert -delay 10 Rplot.png Rplot.gif

To make a .mov file, you can use Quicktime Pro (but you have to pay for the Pro upgrade). In R version 2.8 there is a new package named `animation` which calls ImageMagick from R. It was sort of touch-and-go on my Mac under R 2.7.

2.9 RGL graphics

The 3D animations that I showed were produced using the package `rgl`. Unfortuantely, there is a bug that is currently being fixed right now so I cannot demonstrate it for you. It is a bug on the mac platform.

You can see the graph gallery at http://rgl.neoscientists.org/gallery.shtml. I have also included my source code in the webdav. Under `ou2drgl.R`.

Chapter 3

Introduction to OU Models

Goals:

- Approaches for adaptive evolution (ouch, slouch, others)
- Model-based vs statistical approaches

Concepts:

- Model comparison tools in R
- Process-based models

3.1 The OU Model for Comparative Analysis

Recall that we described a Brownian motion process using the following equation:

$$dX(t) = \sigma\, dB(t). \tag{3.1}$$

If we imagine the phenotype X as changing through time t, this equation says that in a small increment of time, the change will be proportional to the parameter σ. Here, $dB(t)$ is a sample from a Brownian (white noise) process.

A small step towards reality is the OU Process:

$$dX(t) = \alpha\,(\theta - X(t))\, dt + \sigma\, dB(t). \tag{3.2}$$

Eq. 3.2 expresses the amount of change in character X over the course of a small increment of time: specifically, $dX(t)$ is the infinitesimal change in the character X over the

infinitesimal interval from time t to time $t + dt$. The term $dB(t)$ is "white noise"; that is, the random variables $dB(t)$ are independent and identically-distributed normal random variables, each with mean zero and variance dt. The parameter α measures the strength of selection. When $\alpha = 0$, the deterministic part of the OU model drops out and (3.2) collapses to the familiar BM model of pure drift,

3.2 Introduction to Likelihood

3.3 `ouch`

See `ouch` lecture.

Good starting points:

?bimac help page for *Bimaculatus* character displacement dataset

example(bimac) example of bimac analysis

?anolis.ssd help page for *Anolis* sexual size dimorphism dataset

`ouch` is a package designed to test adaptive hypotheses using variations of the OU process, including BM. OUCH implements a model that fits an alpha and sigma parameters to the entire phylogeny, but allows the user to specify which branches belong to different selective regimes. The location of the optima are also fit.

3.3.1 The Data

The data in OUCH are most easily assembled as a data frame. Load the built in example from ouch and then print it to the screen (I only printed the head of the dataset here):

```
> require(ouch)
> data(bimac)
> bimac
```

```
  node species size ancestor time OU.1   OU.3 OU.4  OU.LP
1    1    <NA>   NA       NA    0   ns medium  anc medium
2    2    <NA>   NA        1   12   ns medium  anc medium
3    3    <NA>   NA        2   32   ns medium  anc  small
4    4    <NA>   NA        3   34   ns medium  anc  small
5    5    <NA>   NA        4   36   ns medium  anc  small
6    6    <NA>   NA        3   36   ns medium  anc  small
```

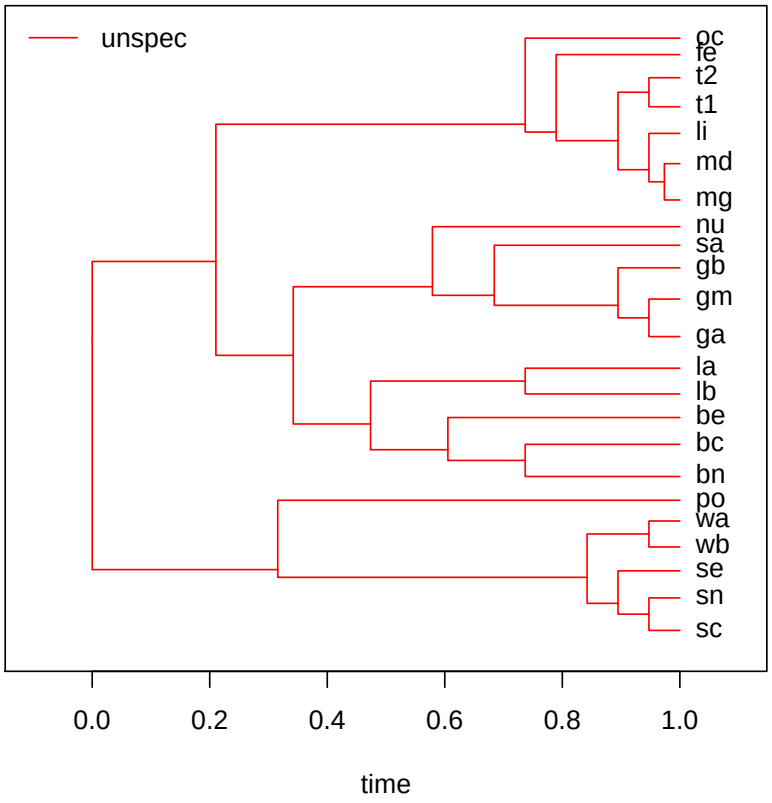
NOTE: a very important detail about `ouch` is that it matches trees with data and regimes using the node labels stored in the rownames of the objects you pass to the `ouch` functions. So it is important to make sure that your dataframes and vectors are appropriately named. The dataframe bimac already has the correct row names, but we do so here just to illustrate.

```
> rownames(bimac) <- bimac$node
```

`ouch` was designed around a rectangular data model, so although the tree object is not a dataframe internally, it still helps us to build the data as a dataframe before making the `ouchtree` objects. The central organizing element is the `node`: it has a node number (usually an integer but it is actually a unique character string), an `ancestor` to which it is joined by a branch, a `time` since the root of the tree, and optional `label` such as a species name. The hypotheses which we use are assigned by "painting" particular regimes on branches. It is convenient to represent each model or hypothesis as a column on the dataframe, with the regime assigned to the node (that is, it is assigned to the branch connecting the node to its ancestor).

Make an `ouchtree` object using the `ouchtree` constructor. `with` is a very nice function to create a small local environment so that you can use a dataframe's elements directly without using the `bimac$` prefix. It is similar to an `attach` but it is temporary – only lasting as long as the call itself. I like it much better than `attach` because I sometimes forget what I've attached and run into problems later. Also, with `attach`, you are actually working with a copy of the original dataframe object, so updating values is trickly. With `with`, it is more clear what's going on, and I don't tend to make those mistakes.

```
> tree <- with(bimac, ouchtree(node,ancestor,time/max(time),species))
> plot(tree)
```



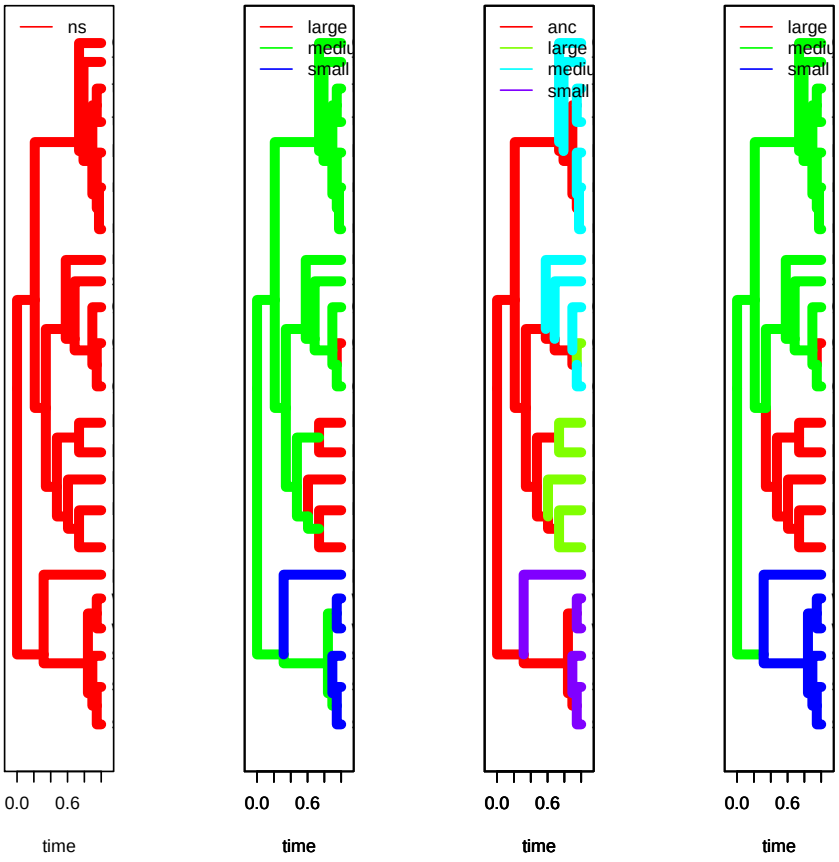
`ouch` fits the OU model Eq. 3.2 along each branch of the phylogeny. While α and σ are held constant across the entire tree, the optima along each branch θ are allowed to vary. Users can then "paint" various combinations of optima on the tree to reflect various biological scenarios.

For example, the dataset `bimac` was used to test the hypothesis of character displacement using an interspecific daaset of body sizes and sympatry/allopatry Butler and King (2004). The analysis tested several different models, which are included with `bimac`. They are: "OU.1" or global optimum, "OU.3" or small, medium, and large regimes depending on the body size of the observed species (terminal branches only, internal branches painted "medium", "OU.4" or the same as "OU.3" but with internal branches given their own unique regime called "ancestral", and "OU.LP" based on a linear parsimony reconstruction of the colonization events (i.e., that as species came into sympatry, they diverged in body size).

3.3.2 Plotting ouchtrees

You can plot the regime paintings on the tree, and set options such as line widths for prettier plots. ouch has a very nice feature which allows plotting of the alternative models on one plot.

```
> plot(tree, regimes=bimac[c("OU.1", "OU.3", "OU.4", "OU.LP")], lwd=6)
```



Remember that you can pass a single vector or a data frame to the regimes parameter, but it must have the appropriate row names or names in the case of a vector. The regimes are not part of the ouchtree object, because they represent our hypothesis of evolution along the tree, rather than the tree itself. It is part of the original dataframe from which we derived the tree, so remember to refer to bimac when passing the regimes to the plot function.

3.3.3 Fitting models

There are two main model fitting functions in `ouch`, `brown`, which fits Brownian motion models, and `hansen`, which fits OU models to comparative data. The call to `brown` is particularly simple, as it takes only the data and the tree:

```
> brown(log(bimac['size']),tree)
```

```
call:
brown(data = log(bimac["size"]), tree = tree)
   nodes ancestors     times labels     size
1      1      <NA> 0.0000000   <NA>       NA
2      2         1 0.3157895   <NA>       NA
3      3         2 0.8421053   <NA>       NA
4      4         3 0.8947368   <NA>       NA
5      5         4 0.9473684   <NA>       NA
6      6         3 0.9473684   <NA>       NA
7      7         1 0.2105263   <NA>       NA
8      8         7 0.3421053   <NA>       NA
9      9         8 0.4736842   <NA>       NA
10    10         9 0.6052632   <NA>       NA
11    11        10 0.7368421   <NA>       NA
12    12         9 0.7368421   <NA>       NA
13    13         8 0.5789474   <NA>       NA
14    14        13 0.6842105   <NA>       NA
15    15        14 0.8947368   <NA>       NA
16    16        15 0.9473684   <NA>       NA
17    17         7 0.7368421   <NA>       NA
18    18        17 0.7894737   <NA>       NA
19    19        18 0.8947368   <NA>       NA
20    20        19 0.9473684   <NA>       NA
21    21        20 0.9736842   <NA>       NA
22    22        19 0.9473684   <NA>       NA
23    23         2 1.0000000     po 2.602690
24    24         4 1.0000000     se 2.660260
25    25         5 1.0000000     sc 2.660260
26    26         5 1.0000000     sn 2.653242
27    27         6 1.0000000     wb 2.674149
28    28         6 1.0000000     wa 2.701361
29    29        10 1.0000000     be 3.161247
30    30        11 1.0000000     bn 3.299534
31    31        11 1.0000000     bc 3.328627
32    32        12 1.0000000     lb 3.353407
```

```
33    33        12 1.0000000     la 3.360375
34    34        13 1.0000000     nu 3.049273
35    35        14 1.0000000     sa 2.906901
36    36        15 1.0000000     gb 2.980619
37    37        16 1.0000000     ga 2.933857
38    38        16 1.0000000     gm 2.975530
39    39        17 1.0000000     oc 3.104587
40    40        18 1.0000000     fe 3.346389
41    41        20 1.0000000     li 2.928524
42    42        21 1.0000000     mg 2.939162
43    43        21 1.0000000     md 2.990720
44    44        22 1.0000000     t1 3.058707
45    45        22 1.0000000     t2 3.068053

sigma squared:
           [,1]
[1,] 0.04311003

theta:
NULL
   loglik  deviance       aic     aic.c       sic       dof
 17.33129 -34.66257 -30.66257 -30.06257 -28.39158   2.00000
```

What is returned is an object of class `browntree`. It contains all input including the function call, the tree and data), as well as the parameter estimate for σ and the model fit statistics including: the log-likelihood, the deviance ($-2 * log(L)$), the information criteria AIC, AIC_c (corrected for small sample size), and SIC, and the model degrees of freedom.

It is a good practice to save this, as it encapsulates the analysis. From this, we can rerun the model fit.

```
> h1 <- brown(log(bimac['size']),tree)
```

`hansen` models are slightly more complex. In addition to σ, we are now fitting α, the strength of selection, and all of the optima θ specified by our model. This maximum-likelihood search now requires an initial guesses. If you have no idea, a good starting guess is 1. If you want to be sure, you can intiate searches with different starting guesses. You can also specify alternative optimization algorithms and increase or decrease the relative tolerance, which is the stringency by which convergence is assessed. Typically, the default is roughly `reltol=1e-8`, and the limit of machine precision is in the neighborhood of `reltol=1e-15`.

```
> h2 <- hansen(log(bimac['size']),tree,bimac['OU.1'],sqrt.alpha=1,sigma=1)
> h3 <- hansen(log(bimac['size']),tree,bimac['OU.3'],sqrt.alpha=1,sigma=1)
```

```
> h4 <- hansen(log(bimac['size']),tree,bimac['OU.4'],sqrt.alpha=1,sigma=1)
> h5 <- hansen(log(bimac['size']),tree,bimac['OU.LP'],sqrt.alpha=1,sigma=1,reltol=1e-5
```

3.3.4 hansentree and ouchtree methods

We can see the model results by typing `h5`, which will execute the `print` method for this class. You could also use the `attributes` function, but this will dump too much information. `ouchtree` objects and the classes derived from them contain information that is used in internal calculations of the algorithms, not of general interest to users.

Additional accessor functions include:

```
> coef(h5)    # the coefficients of the fitted model


$sqrt.alpha
[1] 1.61658

$sigma
[1] 0.2249274

$theta
$theta$size
   large   medium    small
3.355087 3.040729 2.565249


$alpha.matrix
        [,1]
[1,] 2.61333

$sigma.sq.matrix
           [,1]
[1,] 0.05059232


> logLik(h5)   # the log-likelihood value


[1] 24.81823


> summary(h5)   # everything in the print method except the tree+data
```

We can now generate a table of our model fits:

```
> unlist(summary(h5)[c('aic', 'aic.c', 'sic', 'dof')])  # just the model fit statistics

      aic     aic.c       sic       dof
-39.63645 -36.10704 -33.95898   5.00000

>                                    # on a single line
> h <- list(h1, h2, h3, h4, h5)   # store all our fitted models in a list
> names(h) <- c("BM", "OU.1", "OU.3", "OU.4", "OU.LP")
> sapply( h, function(x) unlist(summary(x)[c('aic', 'aic.c', 'sic', 'dof')]) )

             BM      OU.1      OU.3      OU.4     OU.LP
aic   -30.66257 -25.39364 -29.15573 -35.22319 -39.63645
aic.c -30.06257 -24.13048 -25.62631 -29.97319 -36.10704
sic   -28.39158 -21.98715 -23.47826 -28.41022 -33.95898
dof     2.00000   3.00000   5.00000   6.00000   5.00000

>                                    # table with all models
```

By storing the model fits in a list, we can use apply methods to get the statistics from all the models at once. `sapply` returns a matrix if possible.

```
> h.ic <- sapply( h, function(x) unlist(summary(x)[c('aic', 'aic.c', 'sic', 'dof')]) )
> print( h.ic, digits = 3)

         BM  OU.1  OU.3  OU.4 OU.LP
aic   -30.7 -25.4 -29.2 -35.2 -39.6
aic.c -30.1 -24.1 -25.6 -30.0 -36.1
sic   -28.4 -22.0 -23.5 -28.4 -34.0
dof     2.0   3.0   5.0   6.0   5.0
```

Simulation and bootstrap methods: `simulate` generates random deviates or sets of simulated tip data based on the fitted model. The input is a fitted model `hansentree` or `browntree`, and the output is a list of dataframes, each comparable to the original data. These can then be used to refit the model.

```
> h5.sim <- simulate(object = h5, nsim=10)   # saves 10 sets of simulated data
>                                    #  based on OU.LP
```

`update` refits the model, with one or more parameters changed.

```
> summary( update( object = h5, data = h5.sim[[1]] ) )   # fit the first dataset
```

```
$call
hansen(data = data, tree = object, regimes = regimes, sqrt.alpha = sqrt.alpha,
    sigma = sigma)

$conv.code
[1] 0

$optimizer.message
NULL

$alpha
         [,1]
[1,] 4.203232

$sigma.squared
          [,1]
[1,] 0.0650292

$optima
$optima$size
   large   medium    small
3.233845 2.937525 2.620466


$loglik
[1] 25.67344

$deviance
[1] -51.34687

$aic
[1] -41.34687

$aic.c
[1] -37.81746

$sic
[1] -35.6694

$dof
[1] 5
```

```
> h5.sim.fit <- lapply( h5.sim, function(x) update(h5, x))  # fit all 10 simulations
```

`bootstrap` is a convenience function for generating parametric bootstraps of the parameter estimates. It takes the fitted model, performs the simulations, refits, and outputs a dataframe of parameter estimates.

```
> bootstrap( object = h5, nboot=10)
```

```
       alpha sigma.squared optima.size.large optima.size.medium
1   1.814456    0.06311766          3.317520           3.101595
2  30.322008    0.23435004          3.291273           3.165092
3   4.719009    0.05345055          3.345851           3.062709
4   5.887305    0.07052806          3.365086           3.050895
5   4.991195    0.05823173          3.373929           3.063185
6   6.466649    0.12097660          3.199421           3.032996
7   7.364605    0.08811966          3.283106           3.065028
8   3.670616    0.04141470          3.332683           2.991690
9   5.939537    0.09620429          3.315323           2.932346
10  5.386454    0.05030167          3.447769           3.083609
   optima.size.small   loglik       aic     aic.c       sic dof
1           2.396842 19.83475 -29.66951 -26.14010 -23.99204   5
2           2.650185 31.28303 -52.56606 -49.03665 -46.88859   5
3           2.532390 28.92630 -47.85260 -44.32318 -42.17513   5
4           2.681861 27.72923 -45.45846 -41.92905 -39.78099   5
5           2.647378 28.43304 -46.86607 -43.33666 -41.18860   5
6           2.583068 22.39840 -34.79681 -31.26739 -29.11934   5
7           2.619564 27.28990 -44.57980 -41.05039 -38.90233   5
8           2.620460 29.73296 -49.46592 -45.93651 -43.78845   5
9           2.541401 24.23785 -38.47570 -34.94629 -32.79823   5
10          2.635190 30.79962 -51.59924 -48.06982 -45.92177   5
```

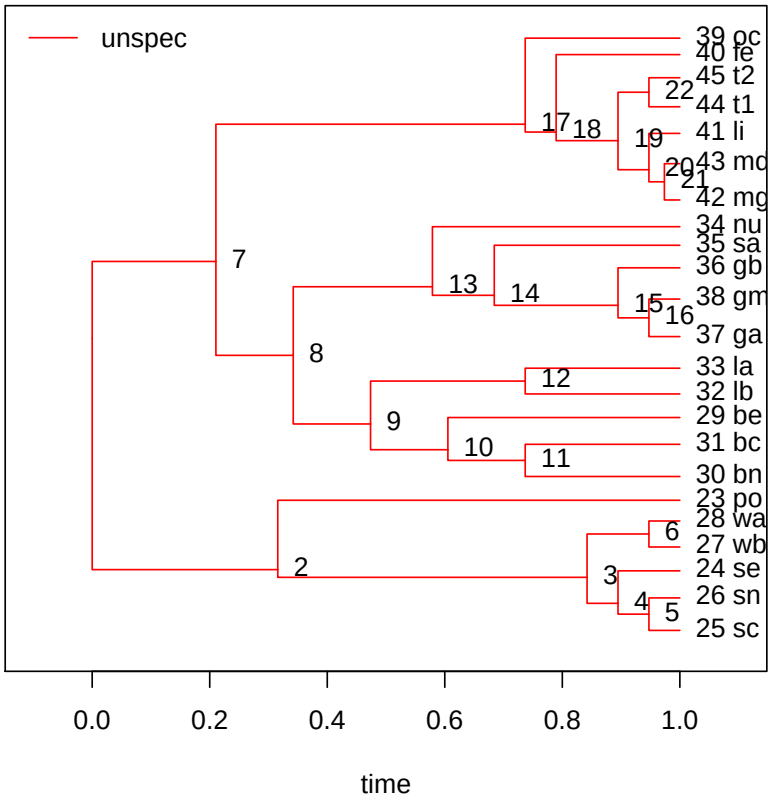
3.3.5 painting regimes on trees

A new function in `ouch` is `paint`. Previously, it was up to users to set up regimes manually by editing spreadsheets. `paint` helps with this task by specifying the regimes on particular species, subtrees, or particular branches.

There are two parameters to `paint`, `subtrees`, which paints the entire subtree which descends from the node, and `branch`, which paints the branch connecting the node to it's ancestor. For either, you specify the node label (remember it's a character and needs to be quoted), and set it equal to the name of the regime you want to specify.

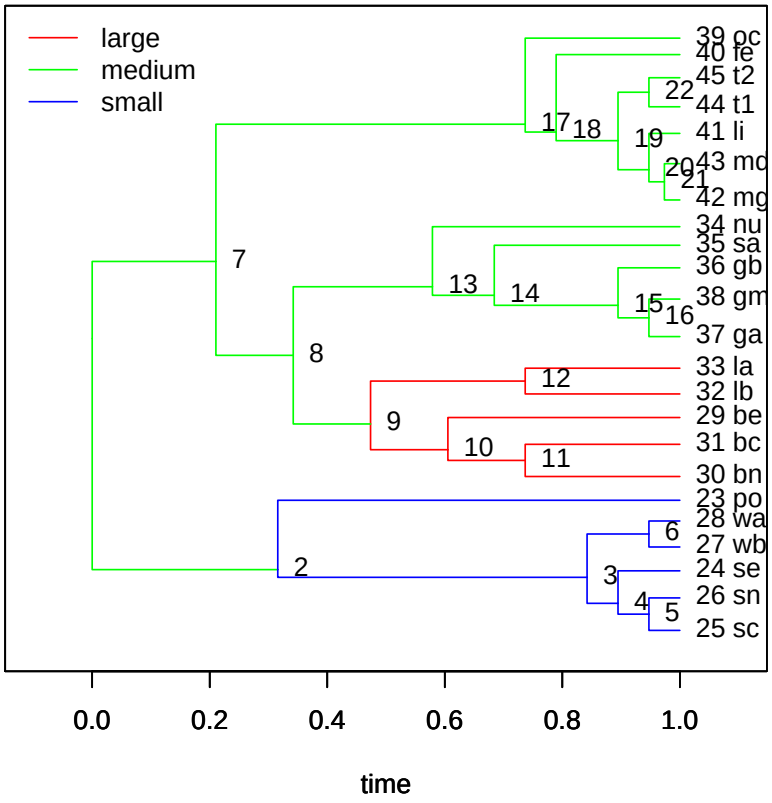
Let's try it on the `bimac` tree and try to recreate the OU.LP regime:

```
> plot(tree, node.names=T)
```



Paint the subtrees first, take a look:

```
> ou.lp <- paint( tree, subtree=c("1"="medium","9"="large","2"="small") )
> plot(tree, regimes=ou.lp, node.names=T)
```

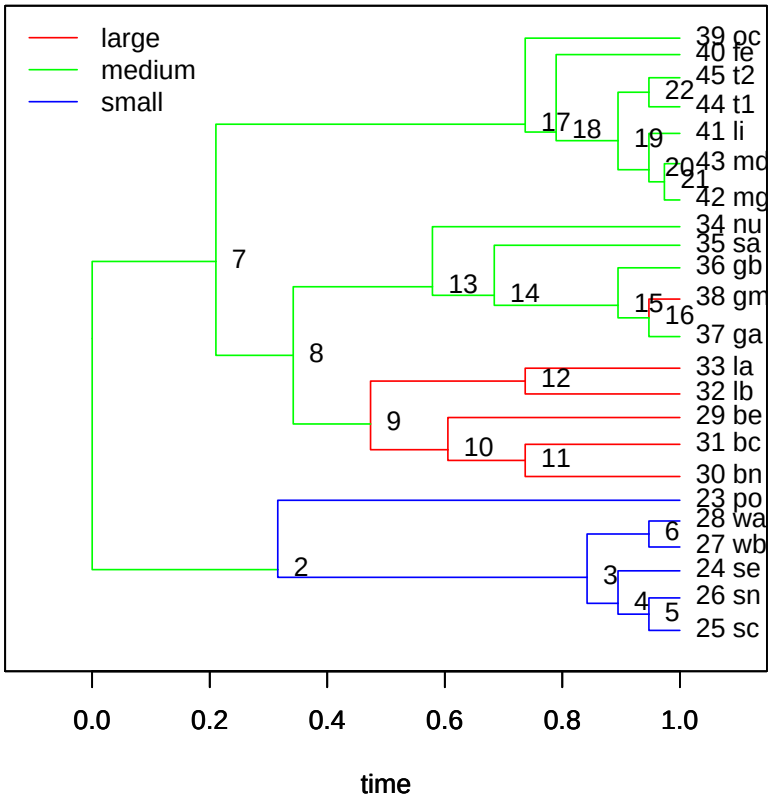


But there was an independent switch from medium to large at species "gm", or node "38", and the node connecting "9" to its ancestor:

```
> ou.lp <- paint( tree, subtree=c("1"="medium","9"="large","2"="small"),
+ branch=c("38"="large","2"="medium"))
```

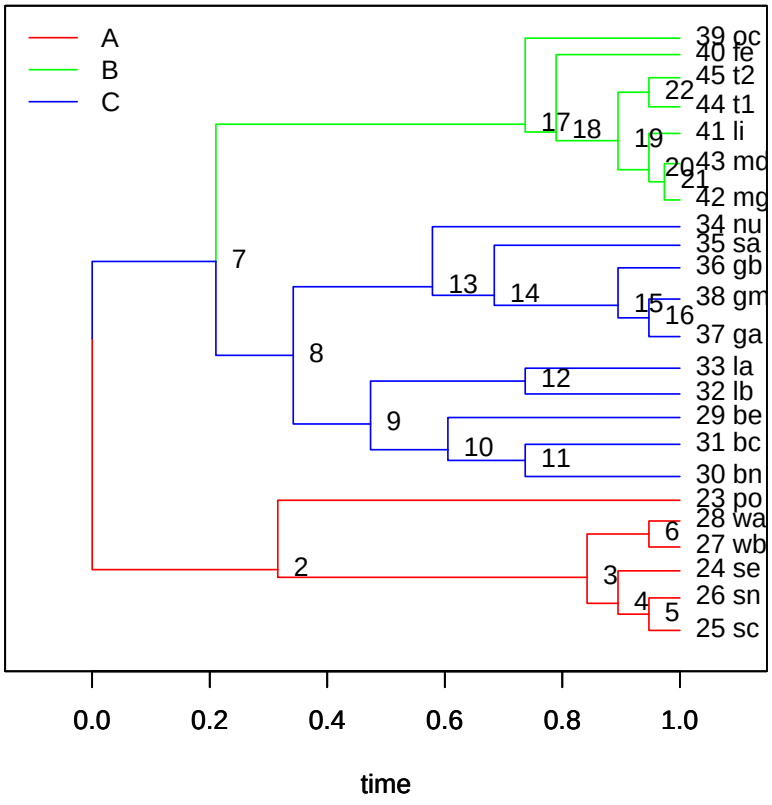
Compare it to the original "OU.LP" from above.

```
> plot(tree, regimes=ou.lp, node.names=T)
```



We can create alternative paintings of the regimes to test against the data. Suppose we wanted to add a "clade specific" hypothesis that diverged in a similar time period (this is a completely made-up hypothesis, just for example):

```
> ou.clades <- paint( tree, subtree=c("1"="A","7"="B", "8"="C"),
+ branch=c("8"="C", "7"="C", "1"="A"))
> plot(tree, regimes=ou.clades, node.names=T)
```



Run the model:

```
> h6 <- hansen(log(bimac['size']),tree, regimes=ou.clades,sqrt.alpha=1,sigma=1)
```

Rebuild our table and compare models:

```
> h <- append(h, h6)          # append (add on) new model results to our list h
> names(h)[length(h)] <- "OU.clades"    # add the name of the new model
> names(h)

[1] "BM"        "OU.1"      "OU.3"      "OU.4"      "OU.LP"     "OU.clades"

> h.ic <- sapply( h, function(x) unlist(summary(x)[c('aic', 'aic.c', 'sic', 'dof')]) )
> print( h.ic, digits = 3)

          BM  OU.1  OU.3  OU.4 OU.LP OU.clades
aic   -30.7 -25.4 -29.2 -35.2 -39.6     -30.7
```

```
aic.c -30.1 -24.1 -25.6 -30.0 -36.1     -27.1
sic   -28.4 -22.0 -23.5 -28.4 -34.0     -25.0
dof     2.0   3.0   5.0   6.0   5.0       5.0
```

Chapter 4

Bivariate `ouch`

The `ouch` package has been completely rewritten by Aaron King to implement a bivariate model, as well as the new S4 class system described previously.

Correlated evolution is a major feature of evolutionary theory, and of great interest among comparative biologists. However, there have been few attempts to develop a bivariate OU model for comparative analysis. NOTE: We are about to submit a paper on this, please cite us (and wait for us to publish first!).

4.0.1 The Bivariate model

The Hansen model describes the evolutionary processes operative on a single quantitative character Hansen (1997); Butler and King (2004). In the case of two characters, we will accordingly have two equations.

$$dX_1(t) = \alpha_1 \left(\theta_1 - X_1(t)\right) dt + \sigma_1 \, dB_1(t). \tag{4.1}$$
$$dX_2(t) = \alpha_2 \left(\theta_2 - X_2(t)\right) dt + \sigma_2 \, dB_2(t). \tag{4.2}$$

The above system of eqs. 4.3 can be written in matrix form with vectors in the place of dX, X, and $dB(t)$, and square matrices in place of α and σ. The θ are already vector valued in the univariate case with a single value per adaptive regime. Here we simply have separate θ vectors for each character.

4.0.2 No Correlations

When evolution is uncorrelated, the α and σ matrices are diagonal:

$$\alpha = \begin{pmatrix} \alpha_{11} & 0 \\ 0 & \alpha_{22} \end{pmatrix} \qquad \sigma = \begin{pmatrix} \sigma_{11} & 0 \\ 0 & \sigma_{22} \end{pmatrix}.$$

This form implies that neither character influences the evolution of the other (i.e., they are evolving independently of one another, and we have a simple duplication of the univariate case).

4.1 Correlated Evolution

We can readily see, however, that the matrices may have off-diagonal elements. These evolutionary correlations can enter in as off-diagonal terms in either the α or the σ terms. In particular, they will have the following form:

$$\alpha = \begin{pmatrix} \alpha_{11} & \alpha_{21} \\ \alpha_{12} & \alpha_{22} \end{pmatrix} \qquad \sigma = \begin{pmatrix} \sigma_{11} & 0 \\ \sigma_{12} & \sigma_{22} \end{pmatrix}.$$

Where $\alpha_{12} = \alpha_{21}$, and without loss of generality, σ is lower-triangular.

Written out as separate equations, the model has the following form:

$$dX_1(t) = \alpha_{11}\,(\theta_1 - X_1(t))\,dt + \alpha_{12}\,(\theta_2 - X_2(t))\,dt + \sigma_{11}\,dB_1(t).$$
$$dX_2(t) = \alpha_{22}\,(\theta_2 - X_2(t))\,dt + \alpha_{12}\,(\theta_1 - X_1(t))\,dt + \sigma_{22}\,dB_2(t) + \sigma_{12}\,dB_1(t).$$

4.2 Implementation in ouch

To illustrate, we reanalyzed the evolution of sexual size dimorphism in association with habitat specialization in *Anolis* lizards (Butler and King, 2004), reformulated as evolution in male and female body size.

Load the data (tree+quantitive data) and regimes:

```
> require(ouch)
> regimes <- read.csv('Rdata/regimes.csv',row.names=1)    # regimes
> ssd <- read.csv('Rdata/ssd.data.csv',row.names=1)    # tree + body size data
> otree <- with(ssd,ouchtree(nodes,ancestors,times,labels))
> xdata <- log(ssd[c('fSVL','mSVL')])
> names(xdata) <- paste('log',names(xdata),sep='.')
> nreg <- length(regimes)
```

ouch now requires you to specify an initial guess for the alpha and sigma matrices. Univariate models are specified by providing a single value. **The bivariate model is specified by providing multiple values for these guesses. The three element vector will be transformed into a symmetric 2x2 matrix for alpha and a lower-triangular matrix for sigma.**

```
> alpha.guess <- c(1, 0, 1)
> sigma.guess <- c(1, 1, 1)
```

Fit the model for the first regime:

```
> tic <- Sys.time()
> hansen(
+        data=xdata,
+        tree=otree,
+        regimes=regimes[5],
+        sqrt.alpha=alpha.guess,
+        sigma=sigma.guess,
+        method="Nelder-Mead",
+        maxit=3000,
+        reltol=1e-12
+        )
```

```
call:
hansen(data = xdata, tree = otree, regimes = regimes[5], sqrt.alpha = alpha.guess,
    sigma = sigma.guess, method = "Nelder-Mead", maxit = 3000,
    reltol = 1e-12)
    nodes ancestors times     labels         TW.6       TW.6.1 log.fSVL
1       1      <NA>  0.00                    twig         twig       NA
2       2         3  0.62             trunk-crown  trunk-crown       NA
3       3         4  0.37             trunk-crown  trunk-crown       NA
4       4         5  0.14             crown-giant  crown-giant       NA
5       5         1  0.08                    twig         twig       NA
6       6         8  0.65           trunk-ground trunk-ground       NA
7       7         8  0.61           trunk-ground trunk-ground       NA
8       8        10  0.54           trunk-ground trunk-ground       NA
9       9        10  0.65             grass-bush   grass-bush       NA
10     10        12  0.50           trunk-ground trunk-ground       NA
11     11        12  0.79            trunk-crown  trunk-crown       NA
12     12        14  0.43           trunk-ground trunk-ground       NA
13     13        14  0.63                  trunk        trunk       NA
14     14        23  0.29           trunk-ground trunk-ground       NA
15     15        16  0.78           trunk-ground trunk-ground       NA
```

16	16	17	0.57		trunk-ground	trunk-ground	NA
17	17	22	0.36		trunk-ground	trunk-ground	NA
18	18	19	0.77		trunk-crown	trunk-crown	NA
19	19	20	0.72		trunk-crown	trunk-crown	NA
20	20	21	0.51		trunk-crown	trunk-crown	NA
21	21	22	0.46		trunk-ground	trunk-ground	NA
22	22	23	0.27		trunk-ground	trunk-ground	NA
23	23	28	0.14		trunk-ground	trunk-ground	NA
24	24	124	0.60		trunk-crown	trunk-crown	NA
124	124	25	0.47		trunk-crown	trunk-crown	NA
25	25	26	0.41		trunk-crown	trunk-crown	NA
26	26	27	0.33		twig	twig	NA
27	27	28	0.23		twig	twig	NA
28	28	34	0.11		twig	twig	NA
29	29	145	0.40		crown-giant	crown-giant	NA
145	145	34	0.28		crown-giant	crown-giant	NA
30	30	31	0.38		grass-bush	grass-bush	NA
31	31	34	0.22		twig	twig	NA
32	32	33	0.77		trunk-ground	trunk-ground	NA
33	33	161	0.72		trunk-ground	trunk-ground	NA
161	161	34	0.44		trunk-ground	trunk-ground	NA
34	34	1	0.08		twig	twig	NA
36	36	2	1.00	A_aliniger	trunk-crown	trunk-crown	3.815512
62	62	24	1.00	A_allisoni	trunk-crown	trunk-crown	4.100989
54	54	17	1.00	A_allogus	trunk-ground	trunk-ground	3.749504
64	64	134	1.00	A_alutaceu	grass-bush	grass-bush	3.487375
134	134	27	0.54		grass-bush	grass-bush	NA
60	60	110	1.00	A_angustic	twig	twig	3.653252
110	110	26	0.40		twig	twig	NA
49	49	13	1.00	A_breviros	trunk	trunk	3.693867
37	37	2	1.00	A_chlorocy	trunk-crown	trunk-crown	3.992681
38	38	3	1.00	A_coelesti	trunk-crown	trunk-crown	4.000034
44	44	7	1.00	A_cooki	trunk-ground	trunk-ground	3.728100
43	43	7	1.00	A_cristate	trunk-ground	trunk-ground	3.797734
66	66	29	1.00	A_cuvieri	crown-giant	crown-giant	4.782479
70	70	32	1.00	A_cybotes	trunk-ground	trunk-ground	3.927896
50	50	13	1.00	A_distichu	trunk	trunk	3.797734
39	39	413	1.00	A_equestri	crown-giant	crown-giant	5.045359
413	413	4	0.90		crown-giant	crown-giant	NA
48	48	11	1.00	A_evermann	trunk-crown	trunk-crown	3.958907
56	56	18	1.00	A_garmani	crown-giant	crown-giant	4.412798
57	57	19	1.00	A_grahami	trunk-crown	trunk-crown	3.784190
42	42	6	1.00	A_gundlach	trunk-ground	trunk-ground	3.811097

```
35     35       403  1.00 A_henderso   grass-bush   grass-bush 3.693867
403   403         4  0.58              grass-bush   grass-bush       NA
53     53        16  1.00 A_homolech trunk-ground trunk-ground 3.706228
69     69       150  1.00 A_insolitu         twig         twig 3.676301
150   150        31  0.65                    twig         twig       NA
46     46         9  1.00    A_krugi   grass-bush   grass-bush 3.671225
59     59       101  1.00 A_lineatop trunk-ground trunk-ground 3.788725
101   101        21  0.62            trunk-ground trunk-ground       NA
61     61       113  1.00 A_loysiana        trunk        trunk 3.575151
113   113        25  0.59                    trunk        trunk       NA
40     40         5  1.00 A_occultus         twig         twig 3.668677
68     68        30  1.00  A_olssoni   grass-bush   grass-bush 3.703768
55     55        18  1.00 A_opalinus  trunk-crown  trunk-crown 3.701302
52     52        15  1.00 A_ophiolep   grass-bush   grass-bush 3.440418
41     41         6  1.00 A_poncensi   grass-bush   grass-bush 3.678829
63     63        24  1.00 A_porcatus  trunk-crown  trunk-crown 3.998201
45     45         9  1.00 A_pulchell   grass-bush   grass-bush 3.610918
65     65        29  1.00 A_ricordii  crown-giant  crown-giant 4.933754
51     51        15  1.00   A_sagrei trunk-ground trunk-ground 3.688879
67     67        30  1.00 A_semiline   grass-bush   grass-bush 3.616309
71     71        32  1.00  A_shrevei trunk-ground trunk-ground 3.832980
47     47        11  1.00 A_stratulu  trunk-crown  trunk-crown 3.686376
58     58        20  1.00 A_valencie         twig         twig 4.226834
72     72        33  1.00 A_whiteman trunk-ground trunk-ground 3.873282
    log.mSVL
1         NA
2         NA
3         NA
4         NA
5         NA
6         NA
7         NA
8         NA
9         NA
10        NA
11        NA
12        NA
13        NA
14        NA
15        NA
16        NA
17        NA
18        NA
```

```
19        NA
20        NA
21        NA
22        NA
23        NA
24        NA
124       NA
25        NA
26        NA
27        NA
28        NA
29        NA
145       NA
30        NA
31        NA
32        NA
33        NA
161       NA
34        NA
36  4.063885
62  4.382027
54  4.030695
64  3.569533
134       NA
60  3.788725
110       NA
49  3.864931
37  4.276666
38  4.247066
44  4.085976
43  4.195697
66  4.877485
70  4.188138
50  3.935740
39  5.129899
413       NA
48  4.258446
56  4.700480
57  4.182050
42  4.171306
35  3.869116
403       NA
53  3.958907
```

```
69  3.737670
150       NA
46  3.906005
59  4.177459
101       NA
61  3.706228
113       NA
40  3.663562
68  3.802208
55  3.901973
52  3.600048
41  3.819908
63  4.261270
45  3.848018
65  5.023222
51  3.977811
67  3.728100
71  4.001864
47  3.843744
58  4.374498
72  4.082609

alpha:
          [,1]      [,2]
[1,] 3.2885637 0.2651125
[2,] 0.2651125 5.0304925

sigma squared:
          [,1]      [,2]
[1,] 0.1164814 0.1606944
[2,] 0.1606944 0.2412312

theta:
$log.fSVL
 crown-giant   grass-bush        trunk  trunk-crown trunk-ground         twig
    4.966605     3.605634     3.678655     3.912812     3.772731     3.798625

$log.mSVL
 crown-giant   grass-bush        trunk  trunk-crown trunk-ground         twig
    5.003842     3.763422     3.832975     4.171052     4.076491     3.874264

    loglik   deviance        aic      aic.c        sic        dof
  82.05319 -164.10638 -128.10638 -116.10638  -86.15318   18.00000
```

```
> toc <- Sys.time()
> print(toc-tic)

Time difference of 3.393276 secs
```

Fitting the Hansen model is much more complex than the Brownian motion, because the α enters non-linearly into the likelihood function. With more variables, the complexity increases, with a less well-behaved likelihood surface than the univariate case. There are frequently convergence issues.

Using the subplex method from package `subplex` helps. Other things to try include increasing the tolerance, increasing the number of maximum iterations allowed to reach convergence, and drawing initial guesses for alpha and sigma at random (and discarding the bad guesses). This of course increases computer time.

```
> tic <- Sys.time()
> h.subplex <- hansen(
+         data=xdata,
+         tree=otree,
+         regimes=regimes[1],
+         sqrt.alpha=alpha.guess,
+         sigma=sigma.guess,
+         method="subplex",
+         maxit=20000,
+         reltol=1e-12
+         )
> toc <- Sys.time()
> print(toc-tic)

Time difference of 20.40885 secs

> summary(h.subplex)

$call
hansen(data = xdata, tree = otree, regimes = regimes[1], sqrt.alpha = alpha.guess,
    sigma = sigma.guess, method = "subplex", maxit = 20000, reltol = 1e-12)

$conv.code
[1] 0

$optimizer.message
NULL
```

```
$alpha
          [,1]      [,2]
[1,] 3.0345716 0.7042767
[2,] 0.7042767 4.3493285

$sigma.squared
          [,1]      [,2]
[1,] 0.1232642 0.1585030
[2,] 0.1585030 0.2201612

$optima
$optima$log.fSVL
   ancestral  crown-giant   grass-bush        trunk  trunk-crown trunk-ground
    4.688811     4.957811     3.588247     3.657690     3.888971     3.758807
        twig
    3.777041

$optima$log.mSVL
   ancestral  crown-giant   grass-bush        trunk  trunk-crown trunk-ground
    6.001250     4.959877     3.710174     3.775684     4.111575     4.050079
        twig
    3.807817


$loglik
[1] 82.4903

$deviance
[1] -164.9806

$aic
[1] -124.9806

$aic.c
[1] -109.7079

$sic
[1] -78.36594

$dof
[1] 20
```

The Brownian motion model is fit:

```
> brown.fit <- brown(
+                 data=xdata,
+                 tree=otree
+               )
> summary(brown.fit)


$call
brown(data = xdata, tree = otree)

$sigma.squared
          [,1]      [,2]
[1,] 0.1523957 0.1564986
[2,] 0.1564986 0.1746020

$theta
$theta$log.fSVL
[1] 3.954663

$theta$log.mSVL
[1] 4.118798


$loglik
[1] 22.56325

$deviance
[1] -45.12651

$aic
[1] -35.12651

$aic.c
[1] -34.26937

$sic
[1] -23.47284

$dof
[1] 5
```

4.3 Exercises

1. Run the Hansen model on the remaining regimes. Can you use an apply method to run them all at once?
2. Plot the multiple regime hypotheses.
3. Compare results.

4.4 Variations of the OU Model — Brian?

Instead of assuming a constant σ across the entire tree, O'Meara et al. (2006) developed a Brownian motion model that allows two or more σ values. This can be interpreted as having different rates of evolution in different regions of the tree.

Other possibilities exist. The difficulty for the future will be twofold: (1) Ensuring that the complexity of the model is reasonable given the information content of the data (i.e., are the parameter estimates and likelihoods well-behaved?). (2) Thinking hard about the best evolutionary and biological interpretations of the models.

Bibliography

Butler, M. A. and A. A. King. 2004. Phylogenetic comparative analysis: a modeling approach for adaptive evolution. American Naturalist, **164**:683–695. URL http://www.journals.uchicago.edu/AN/journal/issues/v164n6/40201/40201.html.

Felsenstein, J. 1988. Phylogenies and quantitative characters. Annual Review of Ecology and Systematics, **19**:445–471. URL http://links.jstor.org/sici?sici=0066-4162%281988%2919%3C445%3APAQC%3E2.0.CO%3B2-I.

Hansen, T. F. 1997. Stabilizing selection and the comparative analysis of adaptation. Evolution, **51**:1341–1351.

Lande, R. 1976. Natural selection and random genetic drift in phenotypic evolution. Evolution, **30**:314–334.

O'Meara, B. C., C. Ane, M. J. Sanderson, and P. C. Wainwright. 2006. Testing for different rates of continuous trait evolution using likelihood. Evolution, **60**:922–933.

Simpson, G. G. 1953. The Major Features of Evolution. Columbia University Press.

Uhlenbeck, G. E. and L. S. Ornstein. 1930. On the theory of the brownian motion. Physical Review, **36**:823–841.