Lab 08: Phylogenetics in R: Trait Evolution in a phylogenetic context

Jennifer Lin (GSI: Ixchel Gonzalez-Ramirez)

3/11/2020

Integrative Biology 200

Principles of Phylogenetics

University of California, Berkeley

Introduction

In the last lab we learnt the basics of working with phylogenetic data in R, and mapped traits on the tips of a phylogeny. Today we will take a step further. When we study traits, we often want to reconstruct the history of the trait based on observations. The process of inferring the states of a trait in internal nodes from observations made on tips of a given phylogeny is called Ancestral State Reconstruction.

First of all make sure you have installed and loaded the libraries we will need

```
#installing packages, commented since you should already have them

#install.packages("ape")

library(ape)
```

Part I: Maximum Likelihood Ancestral State Reconstruction of Discrete Characters

In R we have used a number of functions. Almost every command you have run is a function that takes certain arguments as the inputs and produces some output, such as a value, a datatable, or a plot. We can also define our own functions. Here's an example of a silly function: Taking the mean of a vector with some element NA will produce NA as the answer unless you specify to remove NA values when calculating the mean. To get around that, let's create our own function that removes the NA elements automatically.

```
mean_na <- function(vec) { #we create a function called "mean_na", that takes as inpu
t a vector

  m <- mean(vec, na.rm = T) # the function produces an object, m, that calculates the
mean ommitting NAs

  return(m) #when the function is used, the value m should be returned/printed

}
```

You can test out our new function:

```
test <- c(1,3,4,NA, 8, 1)

mean(test)
```

```
## [1] NA
```

```
mean_na(test)
```

```
## [1] 3.4
```

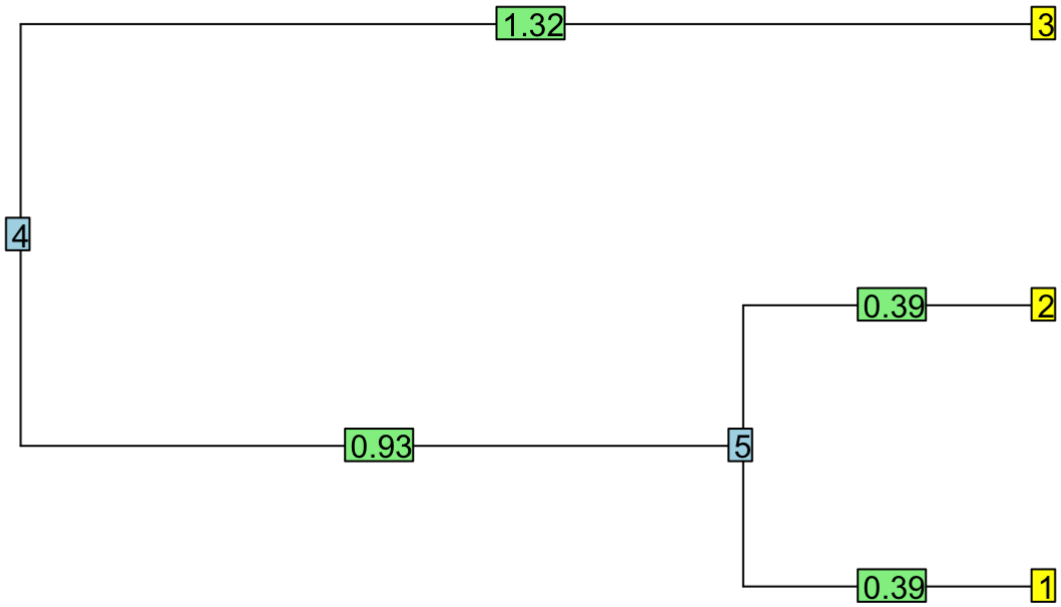
> Today's lecture describes a continuous-time Markov model for a binary character that has two rates of change: alpha is the rate of transitioning from state 0 to 1, and beta is the rate of transitioning from state 1 to 0. See: http://ib.berkeley.edu/courses/ib200/lect/lect17.pdf (http://ib.berkeley.edu/courses/ib200/lect/lect17.pdf).
>
> **Write two functions to calculate the probability of each possible transition over a branch of length t. Each function should take as input alpha, beta, and t.** As a reminder, alpha and beta are the instantaneous rates of change - your function should calculate the probabilites. Insert your code below.

```
#insert your code here
zeroToOne <- function(alpha, beta, t) {
  p <- (alpha/(alpha+beta)) * (1-exp(-(alpha+beta)*t))
  return(p)
}
oneToZero <- function(alpha, beta, t) {
  p <- (beta/(alpha+beta)) * (1-exp(-(alpha+beta)*t))
  return(p)
}
```

Great! Now, let's use those functions to estimate the probabilities of states on a simple tree.

```
t = read.tree(text="((A:0.39,B:0.39):0.93,C:1.32);")

plot(t, show.tip.label=FALSE, main = "Test Tree")

nodelabels()

tiplabels()

edgelabels(t$edge.length)
```

Test Tree



Assume alpha = 2 and beta = 3. Now estimate the probabilities that the tree has the following states:

Node 1 = 0
Node 2 = 0
Node 3 = 1
Node 4 = 1
Node 5 = 0

Hint: the probability of the tree is the product of the probability of each branch. Insert your code below.

```
#insert your code here

# Node_4=1 -> t=1.32 -> Node_3=1
P43 <- 1-oneToZero(alpha = 2, beta = 3, t = 1.32)

# Node_4=1 -> t=0.93 -> Node_5=0
P45 <- oneToZero(alpha = 2, beta = 3, t = 0.93)

# Node_5=0 -> t=0.39 -> Node_2=0
P52 <- 1-zeroToOne(alpha = 2, beta = 3, t = 0.39)

# Node_5=0 -> t=0.39 -> Node_1=0
P51 <- 1-zeroToOne(alpha = 2, beta = 3, t = 0.39)

probabilityOfTheTree <- P43 * P45 * P52 * P51

print(probabilityOfTheTree)
```

```
## [1] 0.1027863
```

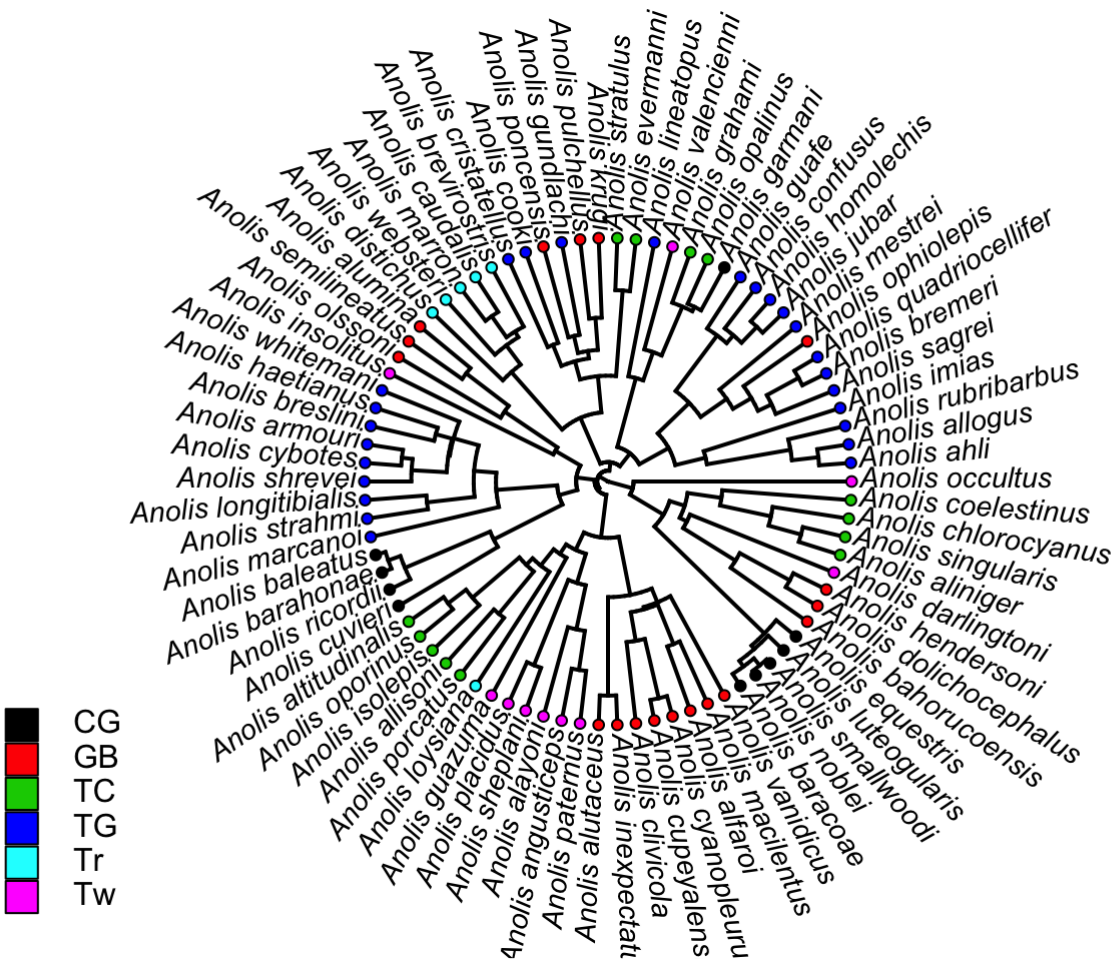
It would be a pain if we had to do this manually every time! Thankfully, there are packages with functions that can do these calculations for us. We'll use an example from Liam J. Revell's Phytools package. Phytools is one of the most commonly used R packages for phylogenetic comparative methods, and the Phytools blog can be incredibly helpful: http://blog.phytools.org/ (http://blog.phytools.org/)

We will get some sample data from the Phytools package. The anoletree dataset. If you are curios to learn more, type ?anoletree in your console.

```
library(phytools)

data(anoletree)

x <- getStates(anoletree, "tips")

tree <- anoletree
```

Plot the data on the tree!

```
plotTree(tree, type = "fan", fsize = 0.9, ftype = "i")

cols <- setNames(palette()[1:length(unique(x))], sort(unique(x)))

tiplabels(pie = to.matrix(x, sort(unique(x))), piecol = cols, cex = 0.2)

add.simmap.legend(colors = cols, prompt = FALSE,
                  x = 0.9 * par()$usr[1], y = -max(nodeHeights(tree)))
```



Here, we will fit a single-rate continuous time Markov model to the data and estimate the ancestral nodes of our tree. In the above exercise, we asked you to calculate the probabilities using a two-rate model. Since there are 6 character states, the instantaneous rate matrix looks like this:

```
   -   a   a   a   a   a
   a   -   a   a   a   a
   a   a   -   a   a   a
   a   a   a   -   a   a
   a   a   a   a   -   a
   a   a   a   a   a   -
```

where the diagonals are -5a

Fit the model using the following code. “ER” means equal rates! Then calculate the log likelihood of the model.

```
fitSR <- rerootingMethod(tree, x, model = "ER")

fitSR$loglik #the likelihood of the model
```

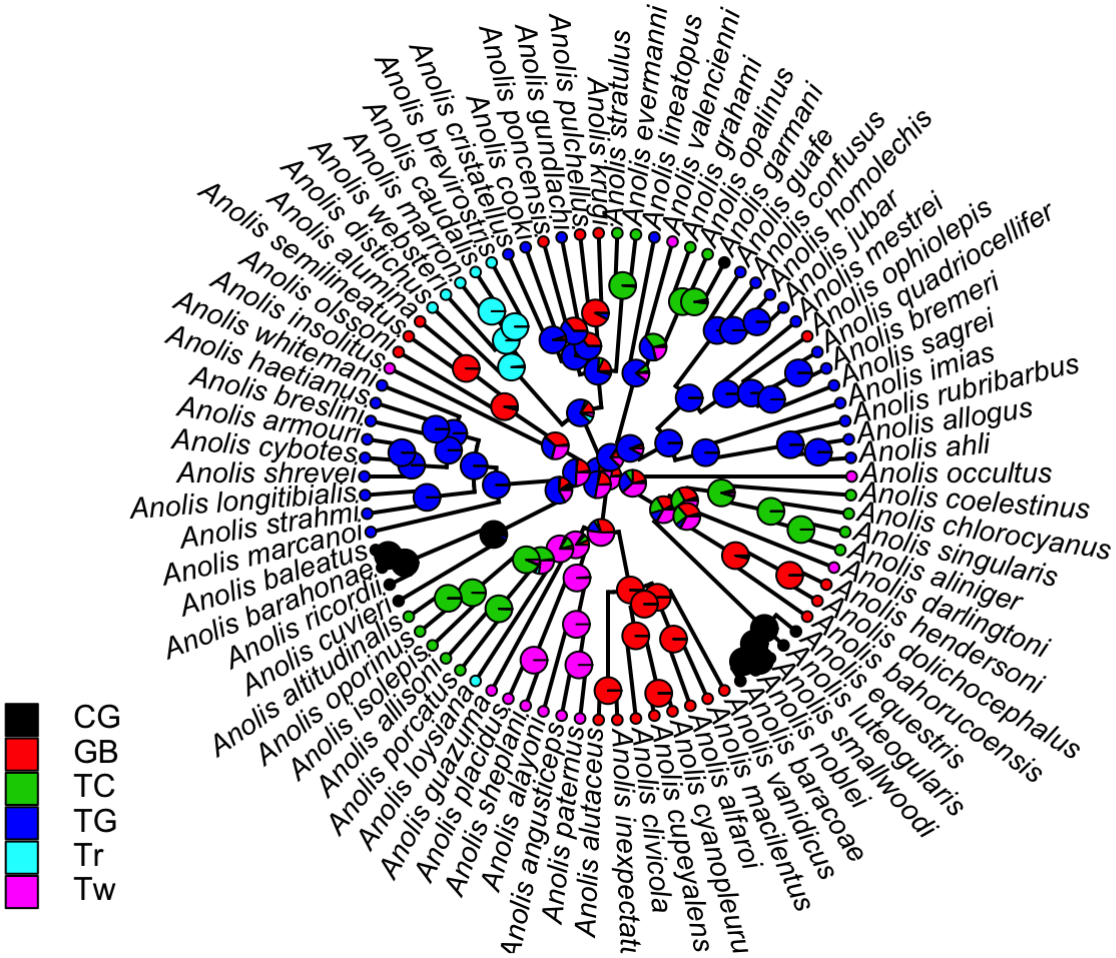
```
## [1] -79.8378
```

```
fitSR$Q #Qmatrix
```

```
##             CG          GB          TC          TG          Tr          Tw
## CG -0.11570723  0.02314145  0.02314145  0.02314145  0.02314145  0.02314145
## GB  0.02314145 -0.11570723  0.02314145  0.02314145  0.02314145  0.02314145
## TC  0.02314145  0.02314145 -0.11570723  0.02314145  0.02314145  0.02314145
## TG  0.02314145  0.02314145  0.02314145 -0.11570723  0.02314145  0.02314145
## Tr  0.02314145  0.02314145  0.02314145  0.02314145 -0.11570723  0.02314145
## Tw  0.02314145  0.02314145  0.02314145  0.02314145  0.02314145 -0.11570723
```

We can plot the results of our ancestral state reconstruction, now showing the ancestral states at the nodes.

```
plotTree(tree, type = "fan", fsize = 0.9, ftype = "i")

nodelabels(node = as.numeric(rownames(fitSR$marginal.anc)),
           pie = fitSR$marginal.anc, piecol = cols, cex = 0.5)

tiplabels(pie = to.matrix(x, sort(unique(x))), piecol = cols, cex = 0.2)

add.simmap.legend(colors = cols, prompt = FALSE,
                  x = 0.9 * par()$usr[1], y = -max(nodeHeights(tree)))
```



Now let's fit a symmetrical-rates model, that allows different (but symmetrical) rates for each character state transition. For a symmetrical-rate 6 state model, the transition rate matrix looks like:

```
   -   a   b   c   d   e
   a   -   f   g   h   i
   b   f   -   j   k   l
   c   g   j   -   m   n
   d   h   k   m   -   o
   e   i   l   n   o   -
```

where the diagonal elements are defined as -1 times the sum of the other row elements, for example, row 1's diagonal element is -(a + b + c + d + e).

We fit the symmetrical rates model using the same code as above, but specifying "SYM" for symmetrical instead of "ER" for equal rates. Also calculate the log likelihood of this model.

```
fitSYM <- rerootingMethod(tree, x, model = "SYM")
```

```
## Warning in log(comp[1:M + N]): NaNs produced
```

```
fitSYM$loglik #the likelihood of the model
```

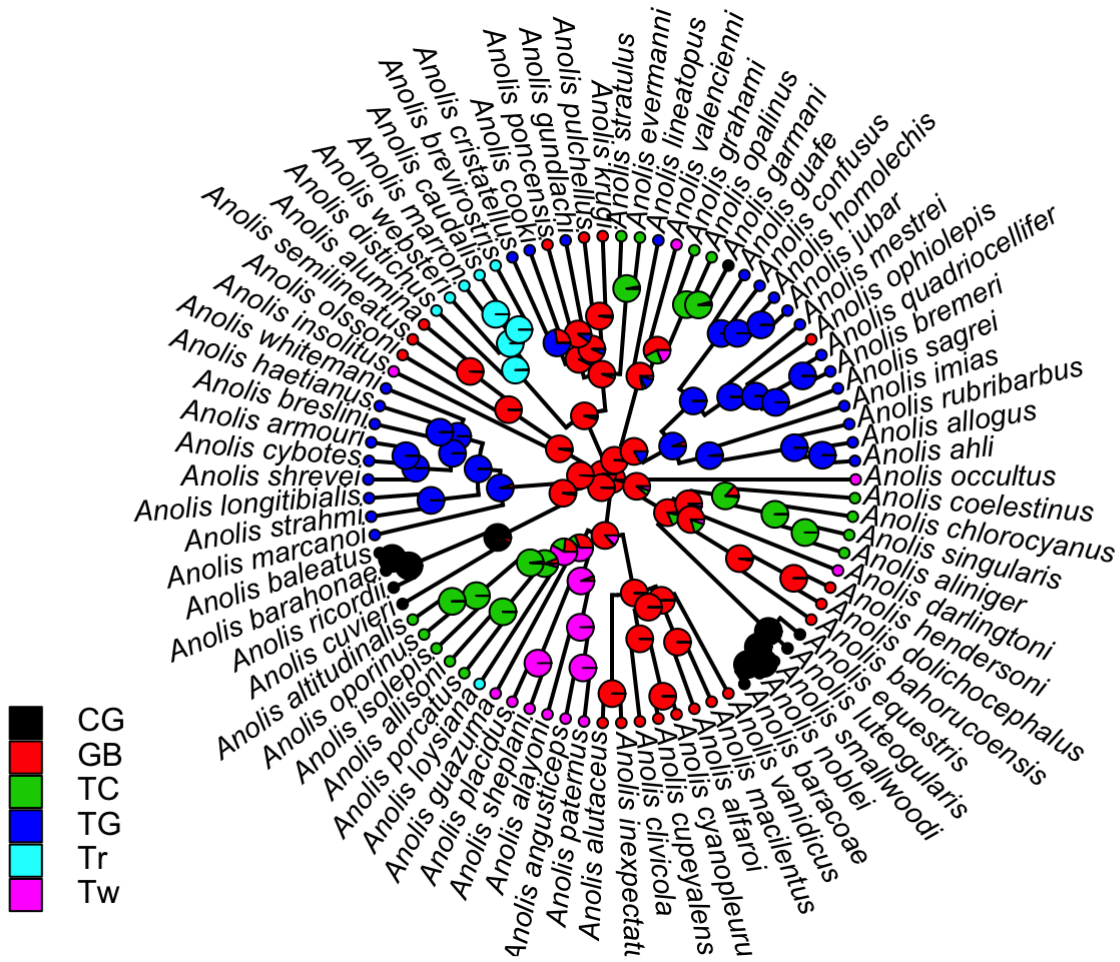
```
## [1] -73.65844
```

```
fitSYM$Q #prints Q matrix
```

```
##             CG          GB          TC          TG          Tr          Tw
## CG -0.05727210  0.02402051  0.03325159  0.00000000  0.00000000  0.00000000
## GB  0.02402051 -0.20033501  0.04238095  0.06009249  0.01341304  0.06042801
## TC  0.03325159  0.04238095 -0.13586016  0.00000000  0.02427571  0.03595191
## TG  0.00000000  0.06009249  0.00000000 -0.06009249  0.00000000  0.00000000
## Tr  0.00000000  0.01341304  0.02427571  0.00000000 -0.03768876  0.00000000
## Tw  0.00000000  0.06042801  0.03595191  0.00000000  0.00000000 -0.09637992
```

Plot the tree again, but show the ancestral states inferred using the symmetrical-rates model. Send me screen shots of both reconstructed ancestral states. Even though the single-rate model is simply a special case of the symmetrical-rates model, are the ancestral state reconstructions the same?

```
#insert your code here
plotTree(tree, type = "fan", fsize = 0.9, ftype = "i")

nodelabels(node = as.numeric(rownames(fitSYM$marginal.anc)),
           pie = fitSYM$marginal.anc, piecol = cols, cex = 0.5)

tiplabels(pie = to.matrix(x, sort(unique(x))), piecol = cols, cex = 0.2)

add.simmap.legend(colors = cols, prompt = FALSE,
                  x = 0.9 * par()$usr[1], y = -max(nodeHeights(tree)))
```




```
# I think for this case, the ancestral state reconstructions are quite similar though the single-rate model is simply a special case of the symmetrical-rates model.
```

Which model fits the data better? Calculate the likelihood ratio test: D = 2 * (loglike of alternative model - loglike of null model). There are 15 parameters in the alternative (SYM) model and 1 parameter in the SR model, so we have 15 - 1 = 14 degrees of freedom. Look up D in a chi-squared distribution table and report the p-value. Is the SYM model supported over the SR model?

```
#insert your code here
D = 2 * (fitSYM$loglik - fitSR$loglik)
print(D)
```

```
## [1] 12.35872
```

```
print(qchisq(.95, df=14))
```

```
## [1] 23.68479
```

```
if (D > qchisq(.95, df=14)){
  print("Accept the alternative model. The SYM model fits the data better.")
}else{
  print("Reject the alternative model. The SYM model doesn't fit the data better.")
}
```

```
## [1] "Reject the alternative model. The SYM model doesn't fit the data better."
```

Part 2: Correlated Evolution of Discrete Traits

Pagel (1994) described an elegant model to test for correlated evolution of discrete traits. Here, we can model two binary traits (with states 0 and 1) as one combined multistate trait (with states 00, 01, 10, 11), illustrated in the table below.

```
table <- data.frame(Trait1 = c(0,1,1,0,1), Trait2 = c(1,1,0,0,1),
                    Combined = c("01","11","10","00","11"))

rownames(table) <- c("OTU1", "OTU2", "OTU3", "OTU4", "OTU5")

#kable(table, align = "c", format = "latex") %>%
#  kable_styling() %>%
#  add_header_above(c(" " = 1, "Binary Traits" = 2, "Multistate" = 1))
```

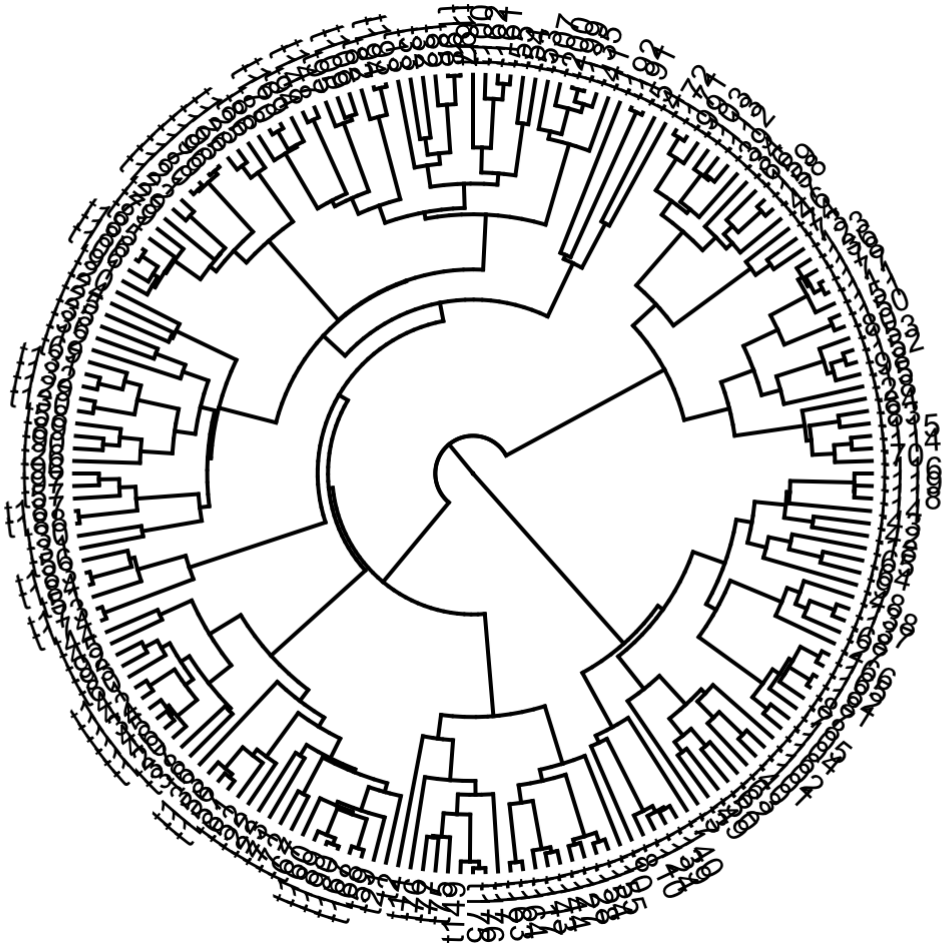
We can then perform an ancestral state reconstruction and estimate the rates of transition between all four ‘states’. Figure 1 illustrates the basic set-up of the model. As you can see, there are separate parameters for transitioning between all 4 states. Let’s think about a biological example. Suppose we are interested in testing for a correlation between two binary traits: presence/ absence of tubers and presence/ absence of rhizomes. Are plants that have tubers more likely to also have rhizomes? We can rephrase this question as, does the rate of evolving rhizomes depend on if the plant already has tubers? If the traits are correlated, we expect that the rate of evolving tubers (trait 1) depends on the presence of rhizomes (trait 2).

We can compare the statistical fit of the fully correlated model (Fig. 1) to the statistical fit of an alternative model. In the alternative model, we force some rates to be equal such that the rate of transitioning between states of one character is the same, regardless of the state of the other character. In our biological example above, we constrain the rates of evolving tubers with and without rhizomes to be the same.

Let’s simulate some fake data and give it a try.

First, let’s simulate a tree with 200 tips.

```
tree <- pbtree(n = 200, scale = 1)

plotTree(tree, type = "fan")
```



Now, let's simulate two traits evolving independently. We do this by building a rate matrix and simulating the evolution of a binary trait, twice. Keep in mind that even though we use the same rate matrix, the traits are still evolving independently.

```
Q <- matrix(c(-1,1,1,-1),2,2)

rownames(Q) <- colnames(Q) <- letters[1:2]

Q
```

```
##    a  b
## a -1  1
## b  1 -1
```

```
binary1 <- sim.history(tree,Q)
```

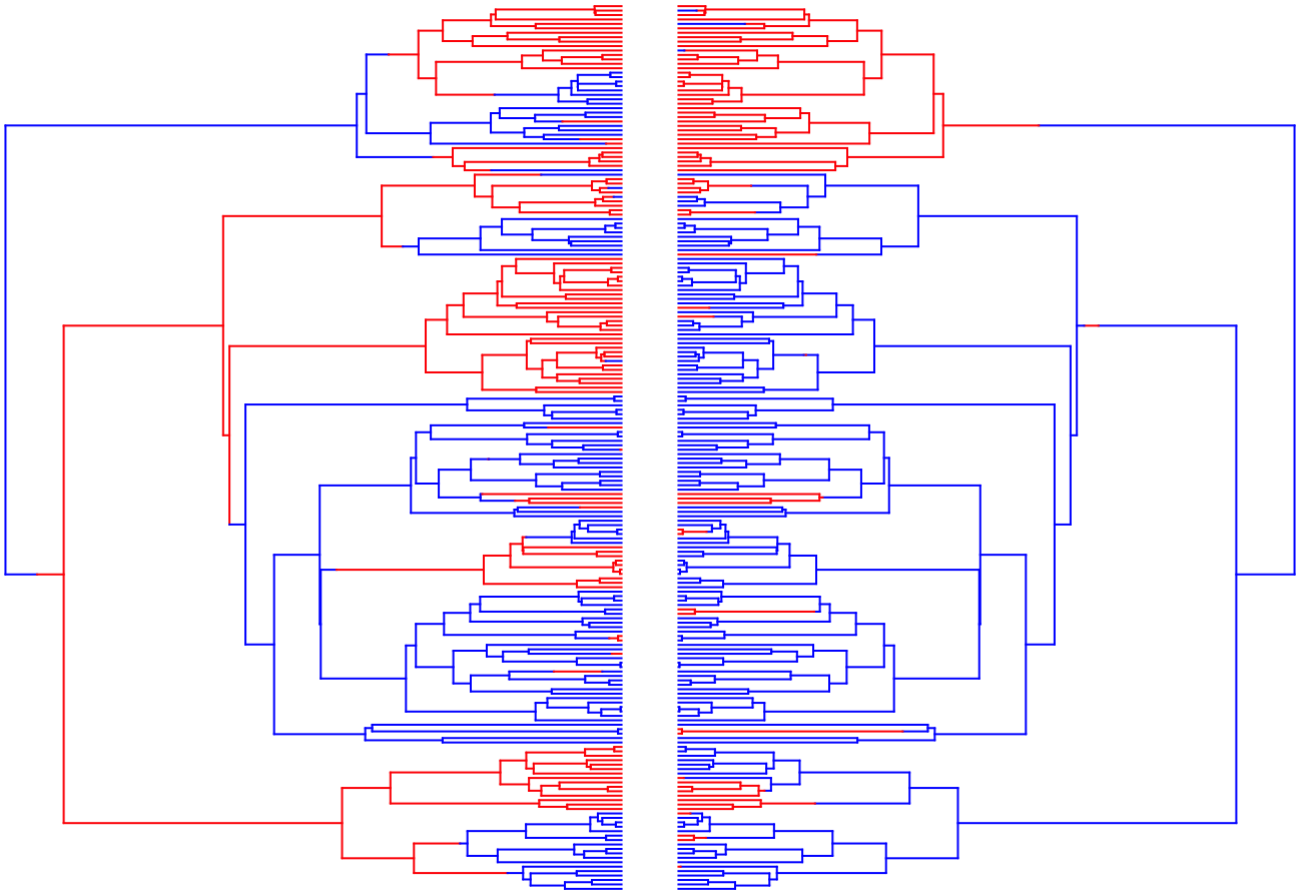
```
## Done simulation(s).
```

```
binary2 <- sim.history(tree,Q)
```

```
## Done simulation(s).
```

Let's plot the results of our simulation to see if they look correlated or not. Remember, we simulated the data separately, so they really shouldn't be correlated!

```
par(mfrow=c(1,2));

plotSimmap(binary1, setNames(c("blue","red"),
                             letters[1:2]), ftype="off", lwd=1)

plotSimmap(binary2, setNames(c("blue","red"),
                             letters[1:2]), ftype="off", lwd=1, direction="leftwards"
)
```



And now, let's fit Pagel's test for correlated evolution to the data:

```
x <- binary1$states

y <- binary2$states

fit <- fitPagel(tree, x, y)

fit
```

```
##
## Pagel's binary character correlation test:
##
## Assumes "ARD" substitution model for both characters
##
## Independent model rate matrix:
##            a|a        a|b        b|a        b|b
## a|a -1.5389827  0.6969307  0.8420520  0.0000000
## a|b  0.5535226 -1.3955746  0.0000000  0.8420520
## b|a  0.7584101  0.0000000 -1.4553408  0.6969307
## b|b  0.0000000  0.7584101  0.5535226 -1.3119327
##
## Dependent (x & y) model rate matrix:
##            a|a        a|b        b|a        b|b
## a|a -1.2536515  0.6861455  0.5675061  0.0000000
## a|b  0.0000000 -2.9258384  0.0000000  2.9258384
## b|a  0.7051912  0.0000000 -1.3284776  0.6232864
## b|b  0.0000000  0.4200947  0.9177814 -1.3378762
##
## Model fit:
##             log-likelihood      AIC
## independent      -133.2950 274.5901
## dependent        -129.8438 275.6876
##
## Hypothesis test result:
##   likelihood-ratio:  6.902465
##   p-value:  0.1411331
##
## Model fitting method used was fitMk
```

Take a close look at this output. Does your test indicate that traits are correlated or not? Which model was favored, and was the difference statistically significant?

```
print("According to the log-likelihood value,")
```

```
## [1] "According to the log-likelihood value,"
```

```
print(paste0("     log-likelihood of independent model: ",fit$independent.logL[1]))
```

```
## [1] "     log-likelihood of independent model: -133.295026264535"
```

```
print(paste0("     log-likelihood of dependent model: ",fit$dependent.logL[1]))
```

```
## [1] "     log-likelihood of dependent model: -129.843793577734"
```

```
print("The higher the log-likelihood is, the better the model is.")
```

```
## [1] "The higher the log-likelihood is, the better the model is."
```

```
print("Thus, DEPENDENT MODEL is favored, which indicates that traits are dependent/co
rrelated")
```

```
## [1] "Thus, DEPENDENT MODEL is favored, which indicates that traits are dependent/c
orrelated"
```

```
print("")
```

```
## [1] ""
```

```
print("The P-value should be at least smaller than 0.05 to make the difference statis
tically significant.")
```

```
## [1] "The P-value should be at least smaller than 0.05 to make the difference stati
stically significant."
```

```
if(fit$P[1]<= 0.05){
  print(paste0("In this case, the P-value is ", fit$P[1], ", which means the differen
ce is statistically significant."))
}else{
  print(paste0("In this case, the P-value is ", fit$P[1], ", which means the differen
ce is not statistically significant."))
}
```

```
## [1] "In this case, the P-value is 0.141133134086588, which means the difference is
not statistically significant."
```

Part 3: Simulating Continuous Character Evolution Under Brownian Motion

We often model the evolution of continuous characters using Brownian Motion. Here, we will model the evolution of a continuous character using Brownian Motion. Then, we will show how to use Brownian Motion to study the evolution of continuous characters.

First, we will initialize our simulations by setting the length of time for the simulation (t). This can also be thought of as the number of generations.

```
t <- 0:100
```

Next, let's initialize the instantaneous rate of change. This determines the relative size of jumps that tend to occur. When sig2 (sigma squared) is small, character state changes tend to be smaller. when its big (maximum of 1), individual state changes tend to be larger.

```
sig2 <- 0.01
```

Now, simulate a set of random deviates. In other words, this is the series of character state changes through time (t). Look at the values of x to get an idea of this.

```
x <- rnorm(n = length(t) - 1, sd = sqrt(sig2))

x
```

```
##   [1] -0.156649397 -0.010879417  0.115587581  0.009883856 -0.010729456
##   [6]  0.160723883 -0.037195725 -0.003452580  0.033710356 -0.019562504
##  [11]  0.074319565  0.032656242  0.152735086 -0.129397183 -0.143016675
##  [16]  0.092130569  0.048204028 -0.163603448 -0.210838294  0.179859409
##  [21]  0.037364644 -0.019656210 -0.102263044 -0.043773128 -0.012464361
##  [26]  0.039674495  0.211500448  0.003033438 -0.082904857 -0.037572492
##  [31] -0.048787105 -0.055546216 -0.219703679 -0.089732829  0.061113180
##  [36]  0.117671001 -0.071960105 -0.025086287  0.054546406 -0.027929702
##  [41]  0.081841613 -0.117027558  0.047272694 -0.025457540 -0.002509667
##  [46] -0.057331446 -0.003208147 -0.005733075 -0.201173030  0.124083922
##  [51]  0.089106138  0.103668374  0.120900562  0.238616138  0.025436426
##  [56]  0.045108467 -0.014721627  0.119146301 -0.023991875  0.051459984
##  [61]  0.069660174  0.140017493 -0.009960580  0.196463330  0.107394431
##  [66]  0.223564021 -0.052462402 -0.267772762 -0.095510849 -0.160091397
##  [71] -0.163616653 -0.081962570  0.062116269  0.088548128 -0.032262702
##  [76]  0.098424314  0.068133885  0.072200686 -0.186914923 -0.057790122
##  [81] -0.005991680 -0.028480533 -0.046333054  0.157338452  0.164315258
##  [86] -0.303542665  0.014031882 -0.106082252  0.153784058  0.023345709
##  [91]  0.001323102  0.082683128  0.147468350 -0.049746788  0.234247708
##  [96] -0.139271732  0.028605013  0.067971013  0.089040643  0.028774479
```

The character state starts at 0 then changes by the amount (x). Imagine this is the size of some trait, like leaf length. when x is negative, it means it got smaller, and when positive, it got bigger.

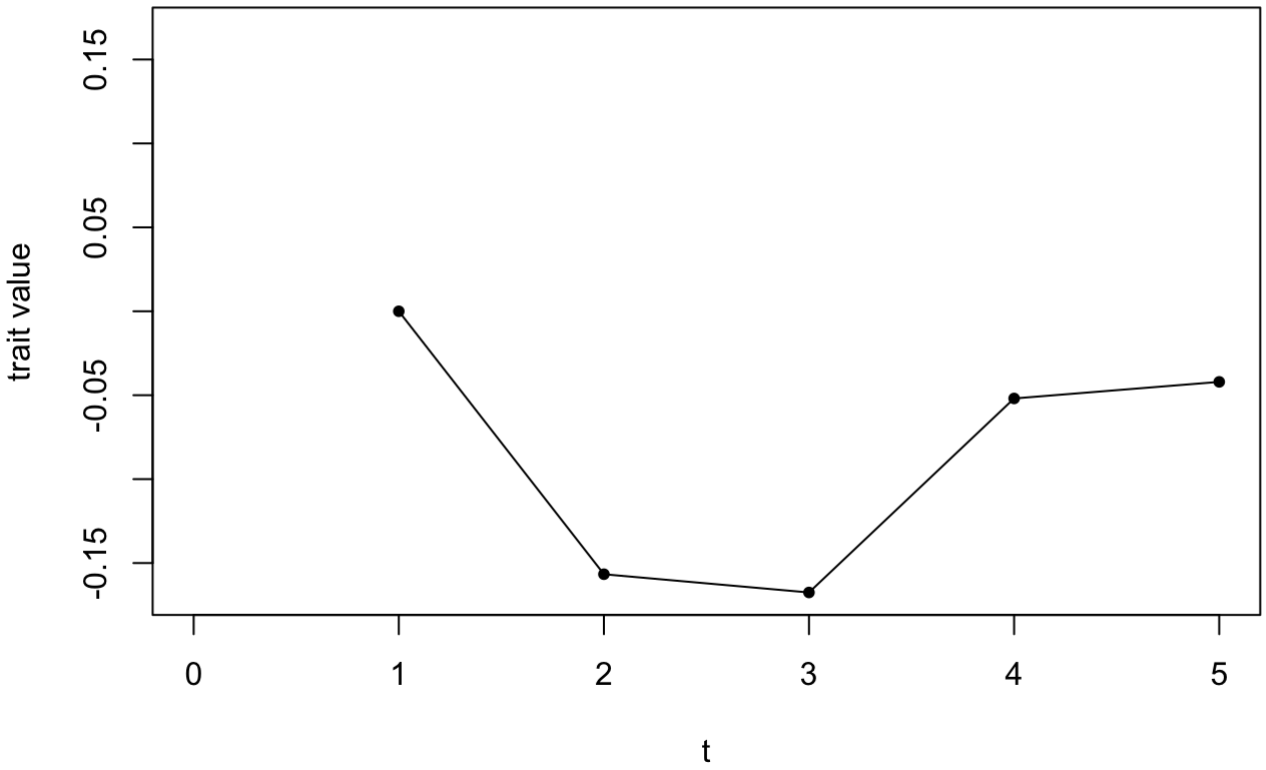
Now compute their cumulative sum. We want the cumulative sum because the state a character is in at any time (t) is the sum of all its past transitions. In other words, the character started at “0”, got bigger by this much, smaller by that much, bigger again, etc. You add those all up to see the state at the end! If we plot all of the cumulative sums over the values of t, we can track the changes in the trait over time.

We can build up to this by starting to generate trait values over time. Each trait value at time t is calculated by adding a change in trait value given in the X vector to the trait value at t-1.

```
stepwise_values <- numeric()

stepwise_values[1] <- 0

stepwise_values[2] <- stepwise_values[1] + x[1]

stepwise_values[3] <- stepwise_values[2] + x[2]

stepwise_values[4] <- stepwise_values[3] + x[3]

stepwise_values[5] <- stepwise_values[4] + x[4]
```

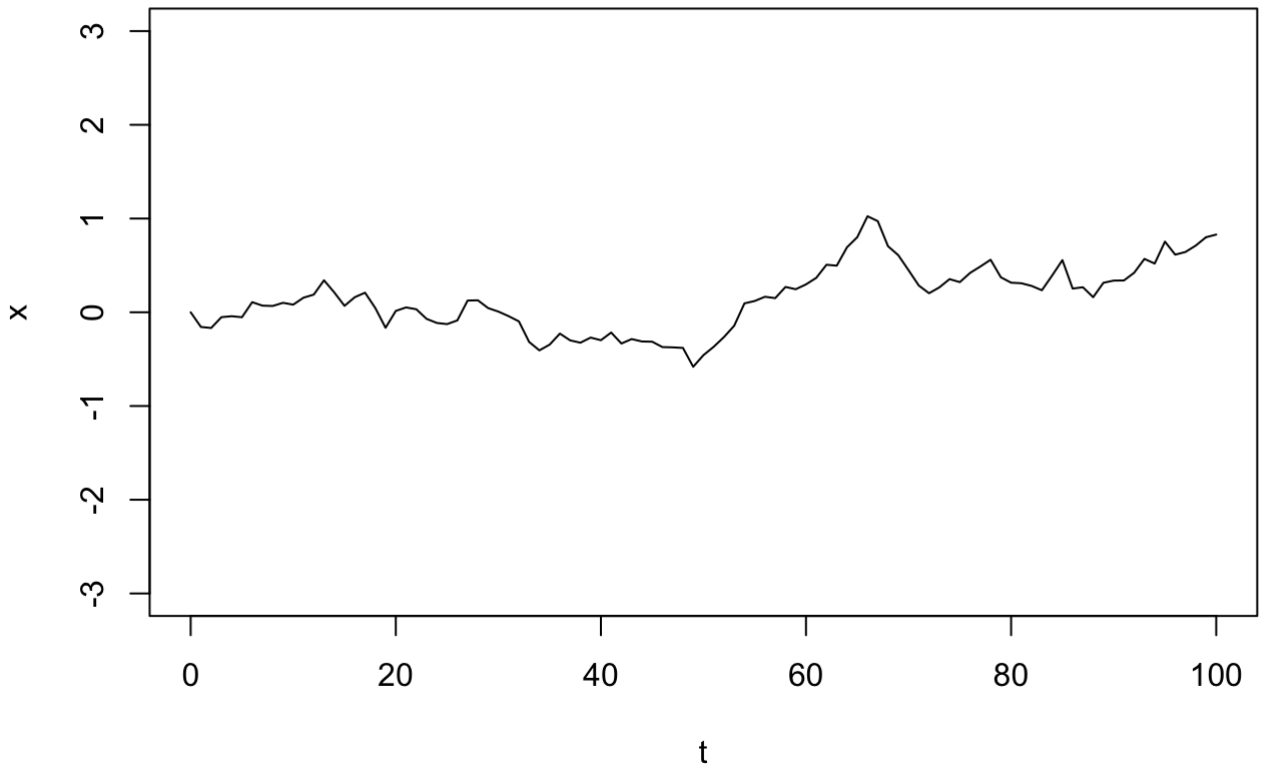
Now that we’ve generated the trait values for the first 5 points in time, we can plot those points to see how the trait values vary as time progresses.

```
abs_max <- max(abs(stepwise_values))

plot(1, stepwise_values[1],
     xlim = c(0,5), ylim = c(-abs_max,abs_max),
     xlab = "t", ylab = "trait value",
     pch = 20, type = "o")

points(c(2:5), stepwise_values[2:5], pch = 20)

lines(1:5, stepwise_values)
```



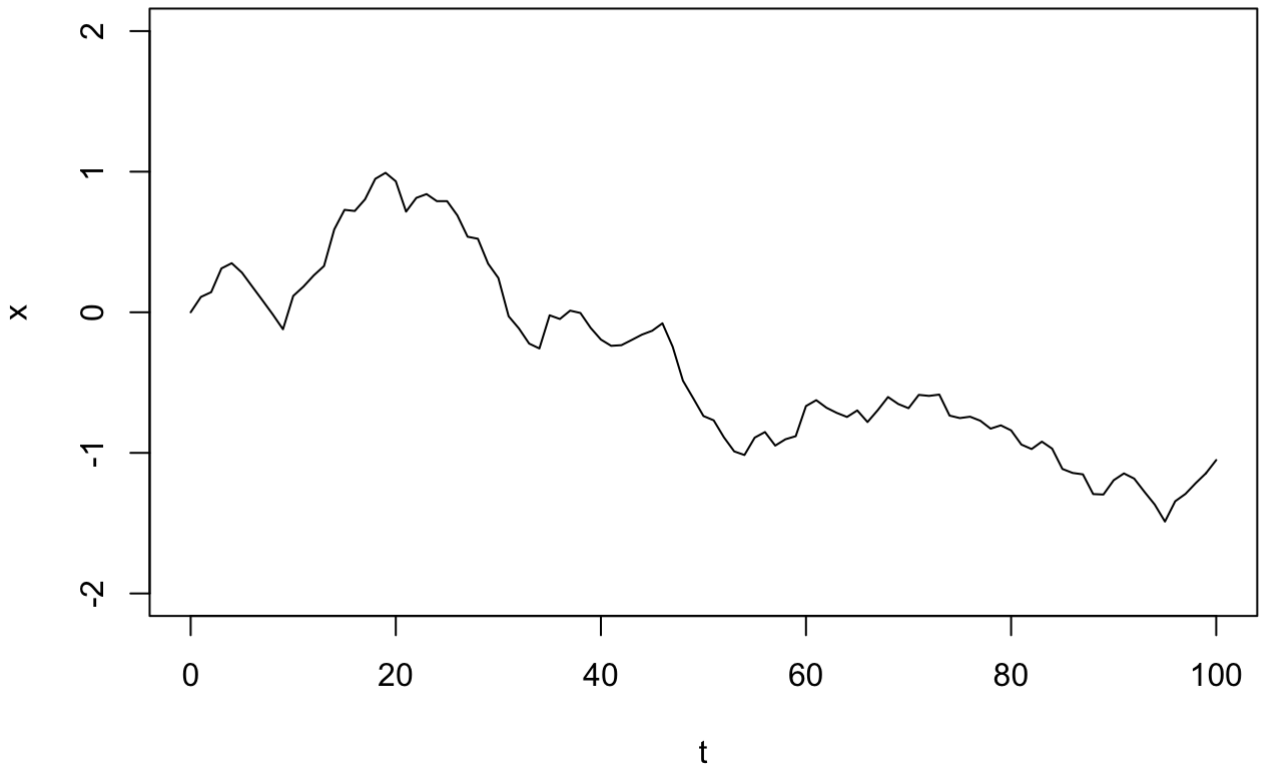
We can do this for all of our values of X using the cumsum() function.

```
x <- c(0, cumsum(x))

plot(t, x, type = "l", ylim = c(-3, 3))
```



Repeat these simulations several times by rerunning the previous code, writen in the following chunk. You'll be drawing new random deviates, so the simulations should look slightly different from each other even though we haven't changed the parameters at all.

```
x <- rnorm(n = length(t) - 1, sd = sqrt(sig2))

x <- c(0, cumsum(x))

plot(t, x, type = "l", ylim = c(-2, 2))
```



The next section of code runs several independent simulations and plot all at once. This can be taught as several lineages evolving independently at the same time.

First, we generate a matrix of simulations, with each simulation as a row and each unit time (t) as a column.

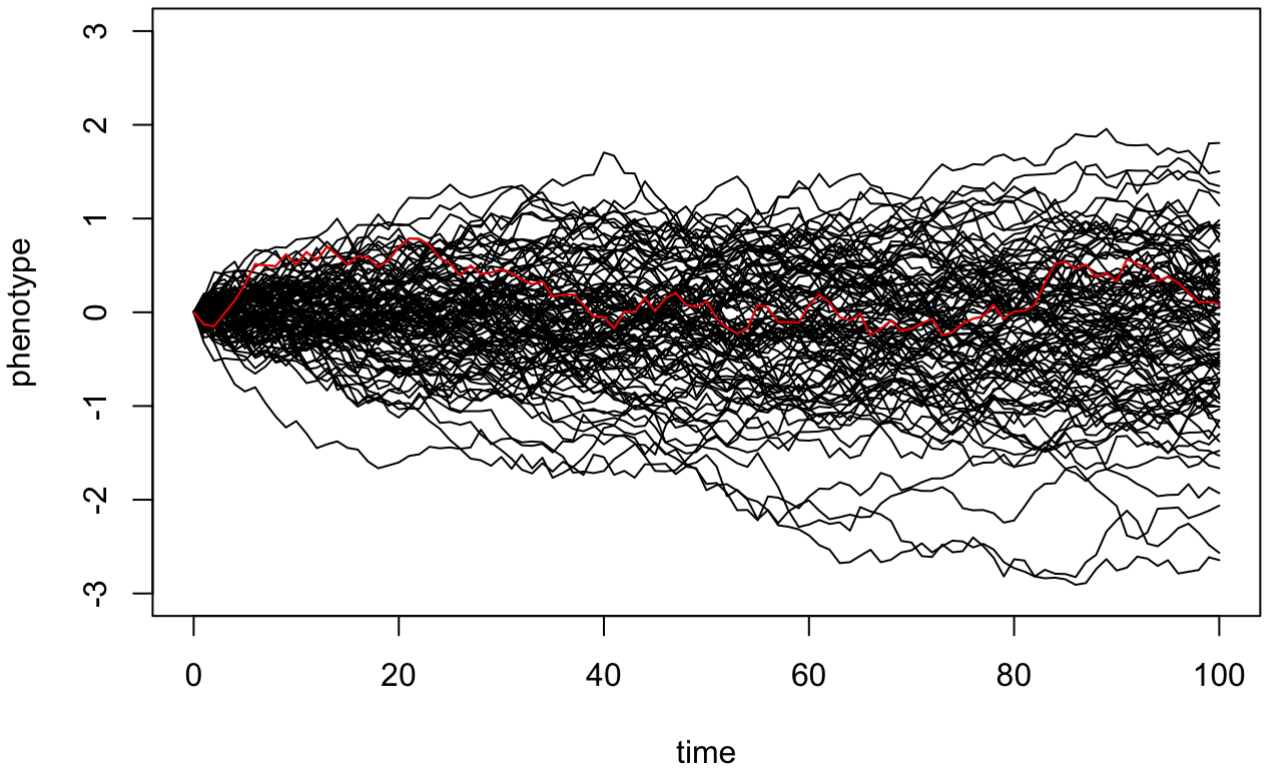
```
t <- 0:100

sig2 <- 0.01

nsim <- 100

X <- matrix(rnorm(n = nsim * (length(t) - 1), sd = sqrt(sig2)), nsim, length(t) - 1)

sim_matrix <- cbind(rep(0, nsim), t(apply(X, 1, cumsum))) #matrix of simulations
```

Then, we plot the first simulation in red and the rest of the simulations in black.

```
plot(t, sim_matrix[1, ], xlab = "time", ylab = "phenotype", ylim = c(-3, 3), type =
"l")

apply(sim_matrix[2:nsim, ], 1, function(x, t) lines(t, x), t = t)
```

```
## NULL
```

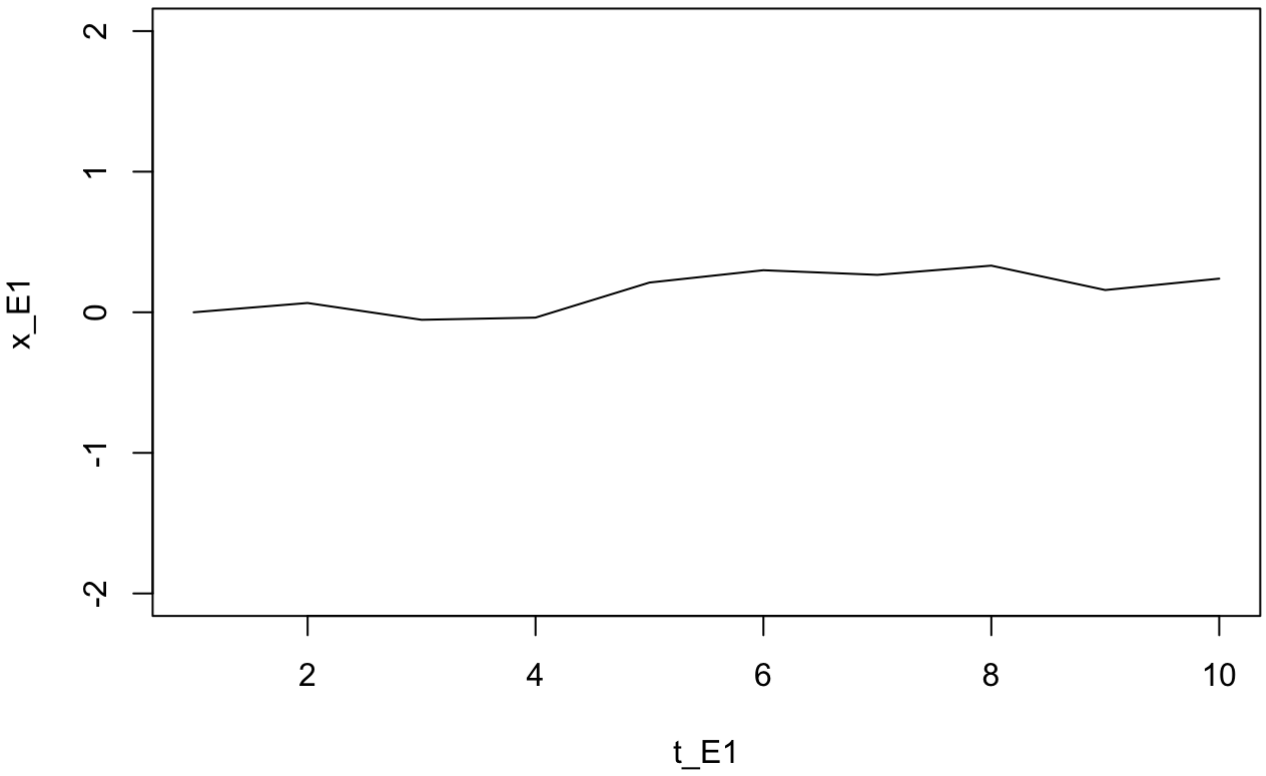
```
lines(t, sim_matrix[1, ], xlab = "time", ylab = "phenotype", ylim = c(-3, 3), col =
"red")
```



Try this a couple of times, again changing sig2 and t. You might need to change the y axis to fit your character state range! (change the numbers in 'ylim = c(-3, 3)' to whatever you want. Include one example of your changed code in a code block below, along with the answers to the following questions:

1) How does the size of t affect how close the final character state (phenotype) is to the initial character state (which is always 0)?

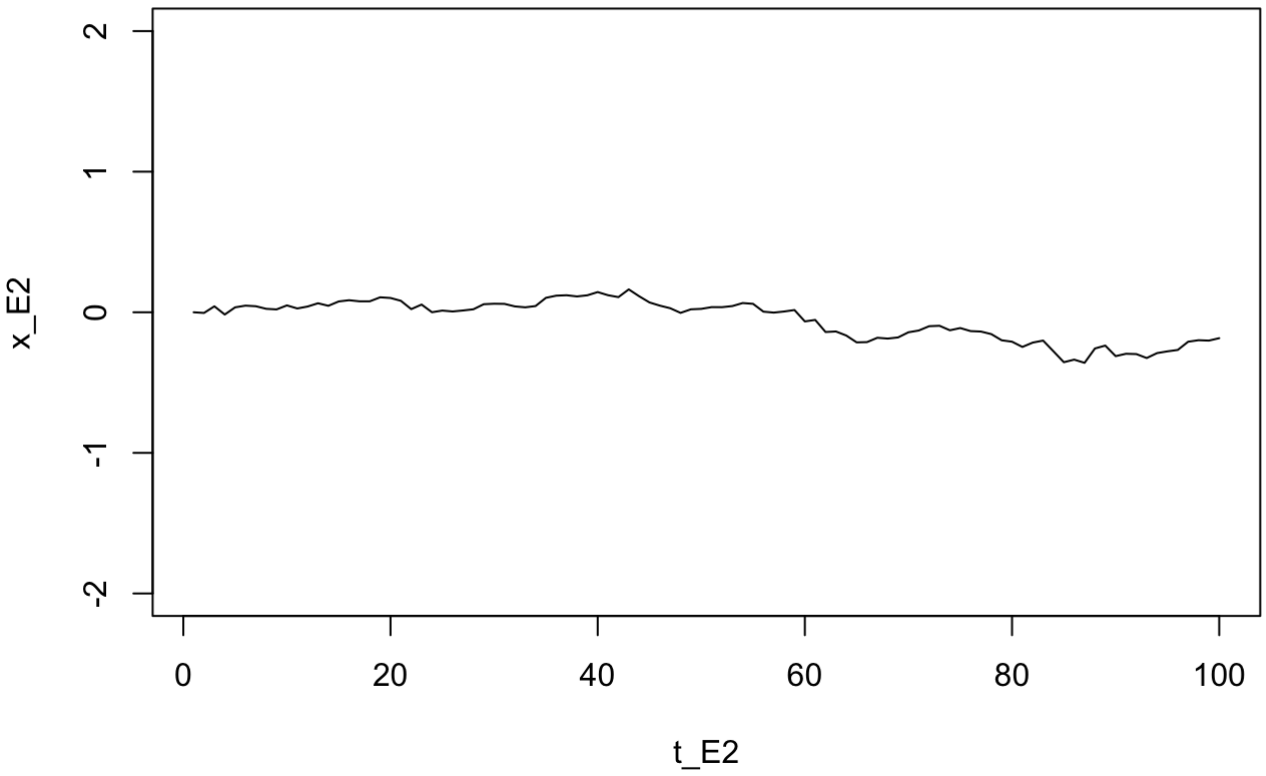
```
#insert your code here
# I change the t from 1:100 to 1:10 in order to compare to the previous simulation.
t_E1 <- 1:10
sig2_E1 <- 0.01
x_E1 <- rnorm(n = length(t_E1) - 1, sd = sqrt(sig2_E1))
x_E1 <- c(0, cumsum(x_E1))
plot(t_E1, x_E1, type = "l", ylim = c(-2, 2))
```




```
# We can see that the SMALLER the size of t is, the CLOSER the final character state
 is to the initial character state.
# This is resonable because the trait has less time to change.
```

2) How does the size of sig2 affect how close the final character state (phenotype) is to the initial character state?

```
#insert your code here
# I change the sig2 from 0.01 to 0.001 in order to compare to the previous simulatio
n.
t_E2 <- 1:100
sig2_E2 <- 0.001
x_E2 <- rnorm(n = length(t_E2) - 1, sd = sqrt(sig2_E2))
x_E2 <- c(0, cumsum(x_E2))
plot(t_E2, x_E2, type = "l", ylim = c(-2, 2))
```



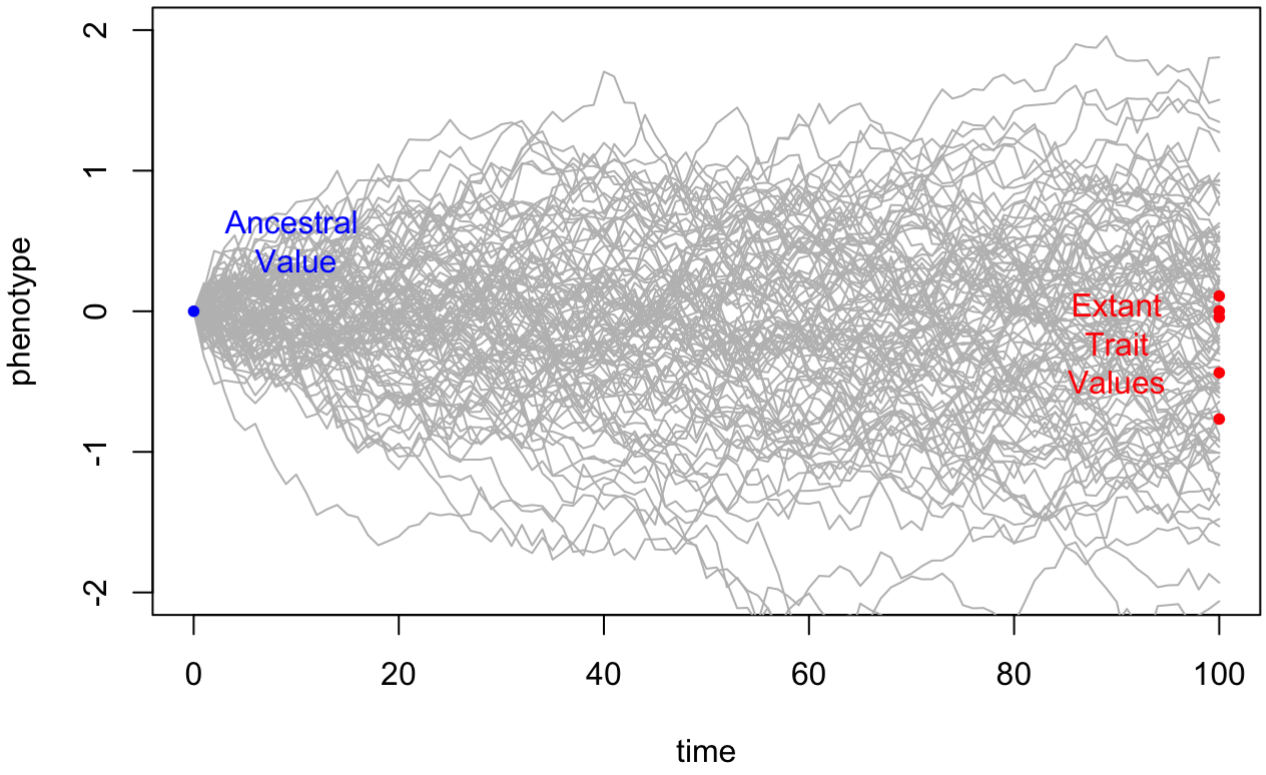

```
# We can see that the SMALLER the sig2 is, the CLOSER the final character state is to the initial character state.
# This is resonable because sig2 determines the relative size of jumps that tend to occur. As mentioned before in this homework document, when sig2 (sigma squared) is small, character state changes tend to be smaller. when its big (maximum of 1), individual state changes tend to be larger.
```

How does all of this connect to Brownian motion in the context of phylogenetics? We are never given the full distribution of thousands of Brownian motion simulations. Instead, we are given a couple of trait values in the present, and are often interested in reconstruction the ancestral value - the start point of the simulation. Here's a visualization of that.

```
plot(t, sim_matrix[1, ], xlab = "time", ylab = "phenotype",
     ylim = c(-2, 2), type = "l", col = "grey")

apply(sim_matrix[2:nsim, ], 1, function(x, t) lines(t, x, col = "grey"), t = t)
```

```
## NULL
```

```
points(0,0, col = "blue", pch = 20)

text(10,.5, "Ancestral \nValue", col = "blue")

points(x = rep(100, times = 5),
       y = sim_matrix[c(1:5),100],
       col = "red", pch = 20)

text(90, mean(sim_matrix[c(1:5),100]), "Extant\nTrait\nValues", col = "red")
```



So, how does Brownian motion help us understand trait evolution? How can we calculate the ancestral value given the extant trait values? Let's go through the following questions to relate this all back to evolution.

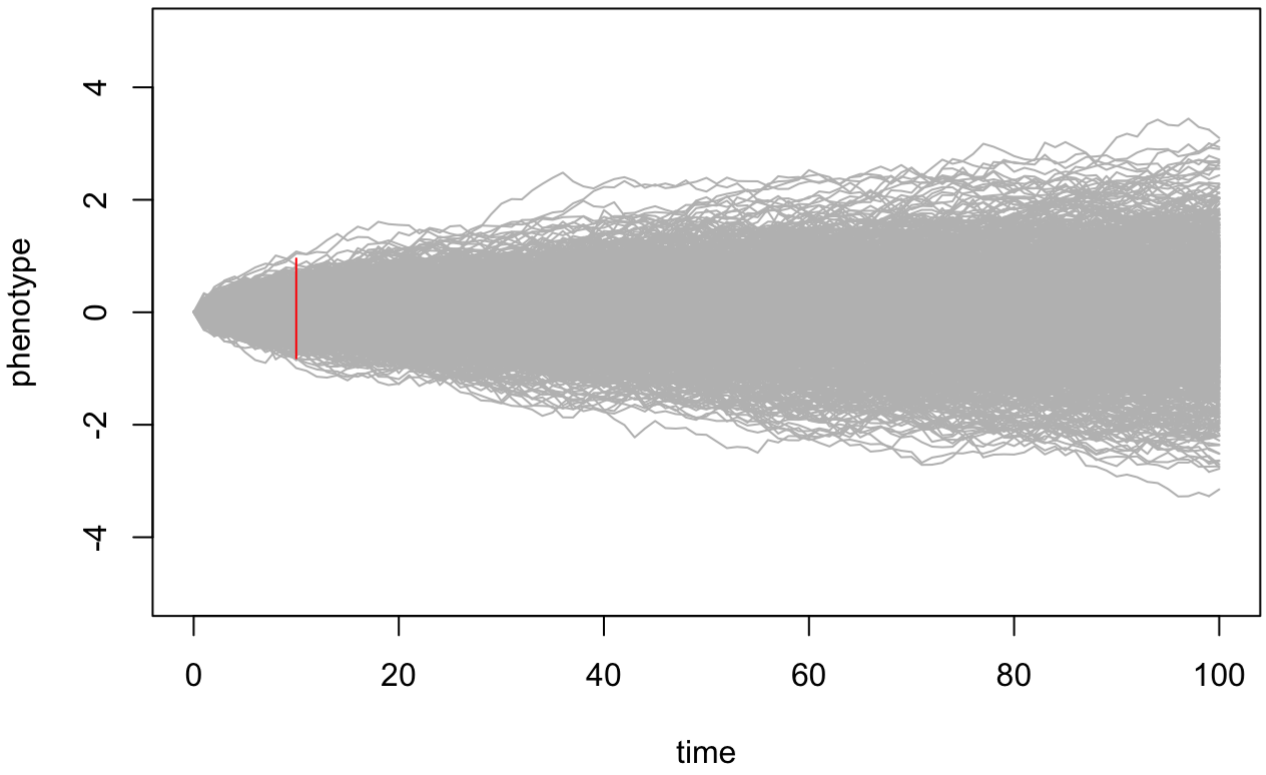
Here's another simulation under Brownian motion. Add 4 additional features to the plot below:

- A horizontal line (in blue) showing the starting trait value. This is also known as the expected value for this Brownian motion simulation.
- 3 verticle lines (in red) illustrating the variance of the distribution at t = 20, t = 60, and t = 100. For example, in the plot below, I've included the variance of the distribution at t = 10. Add on the additional lines to this plot.

```
nsim <- 1000

X <- matrix(rnorm(n = nsim * (length(t) - 1), sd = sqrt(sig2)), nsim, length(t) - 1)

sim_matrix <- cbind(rep(0, nsim), t(apply(X, 1, cumsum)))

plot(t, sim_matrix[1, ], xlab = "time", ylab = "phenotype",
     ylim = c(-5, 5), type = "l", col = "grey")

apply(sim_matrix[2:nsim, ], 1, function(x, t) lines(t, x, col = "grey"), t = t)
```

```
## NULL
```

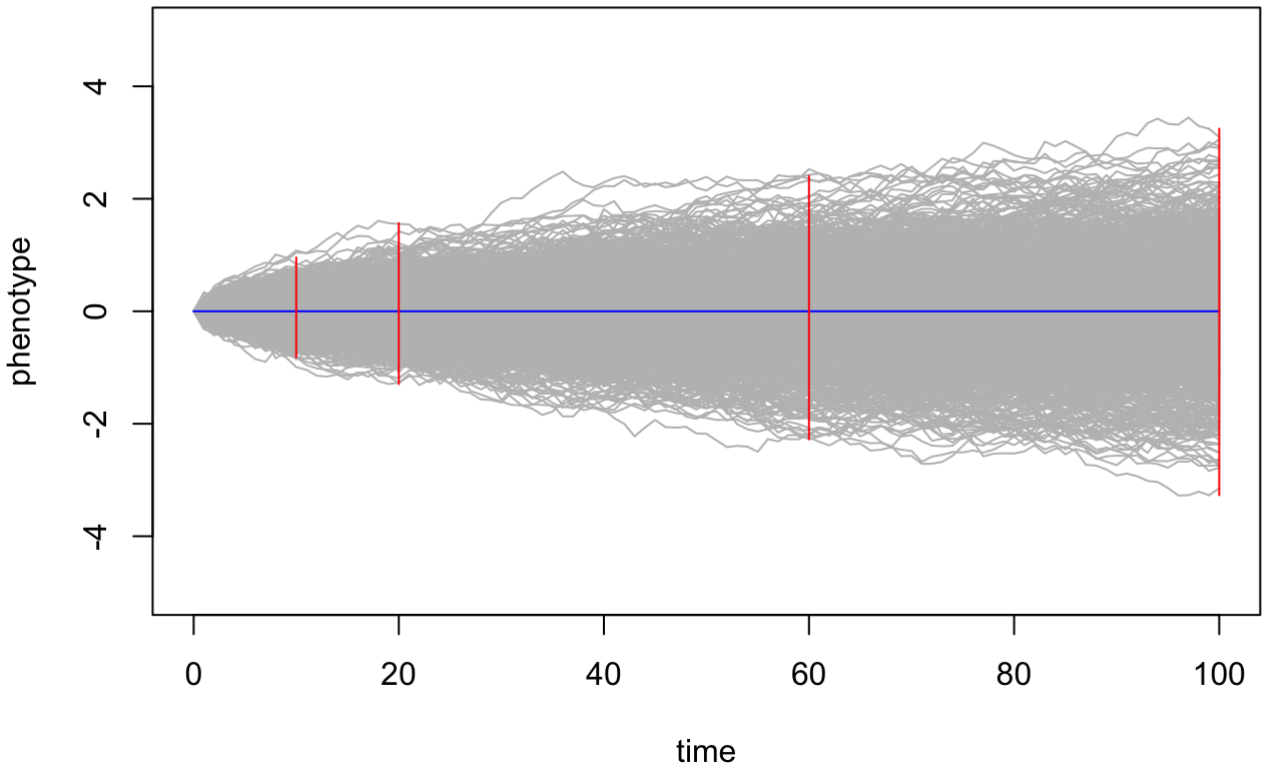
```
segments(x0 = 10, y0 = min(sim_matrix[,10]), y1 = max(sim_matrix[,10]), col = "red" )
```




```
#insert your code here for the completed plot
plot(t, sim_matrix[1, ], xlab = "time", ylab = "phenotype",
     ylim = c(-5, 5), type = "l", col = "grey")
apply(sim_matrix[2:nsim, ], 1, function(x, t) lines(t, x, col = "grey"), t = t)
```

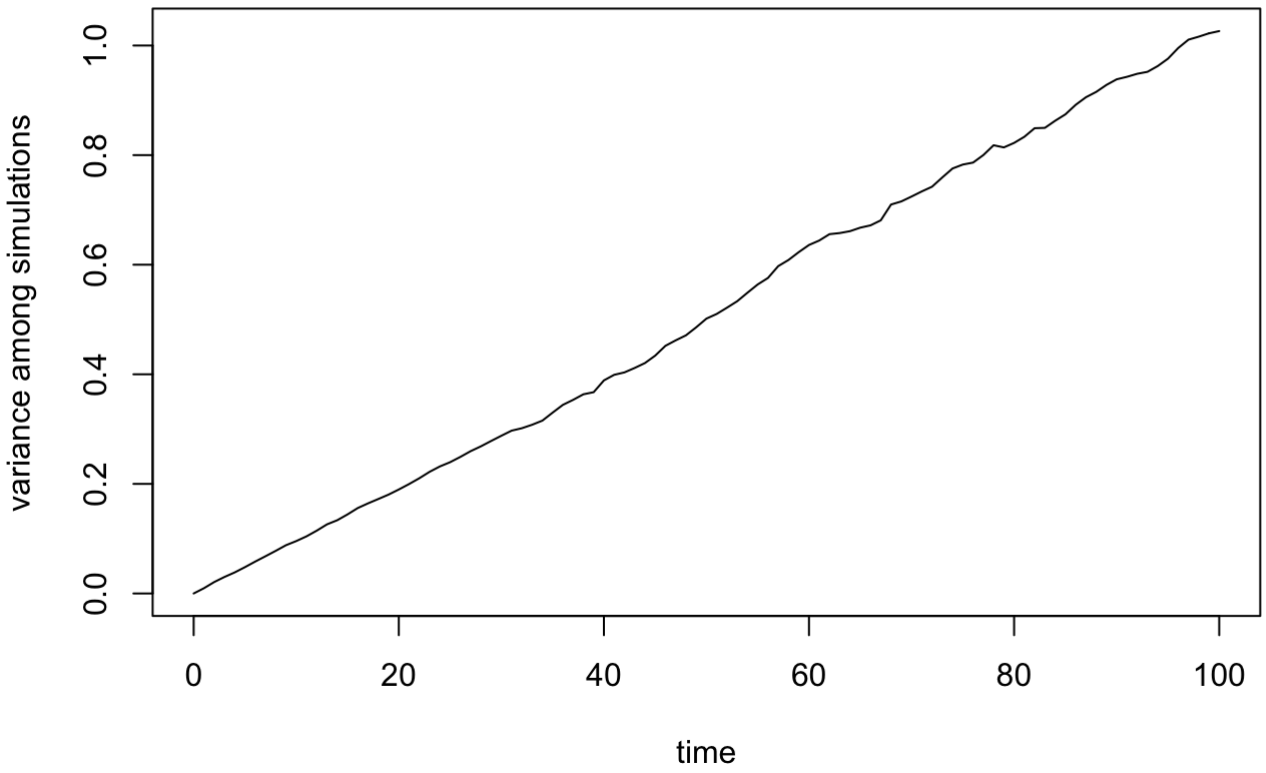
```
## NULL
```

```
segments(x0 = 0, x1=100, y0 = 0, col = "blue" )
segments(x0 = 10, y0 = min(sim_matrix[,10]), y1 = max(sim_matrix[,10]), col = "red" )
segments(x0 = 20, y0 = min(sim_matrix[,20]), y1 = max(sim_matrix[,20]), col = "red" )
segments(x0 = 60, y0 = min(sim_matrix[,60]), y1 = max(sim_matrix[,60]), col = "red" )
segments(x0 = 100, y0 = min(sim_matrix[,100]), y1 = max(sim_matrix[,100]), col = "re
d" )
```



Here's a graphical illustration of the relationship between t and the variance among simulations.

```
v <- apply(sim_matrix, 2, var)

plot(t, v, type = "l", xlab = "time", ylab = "variance among simulations")
```

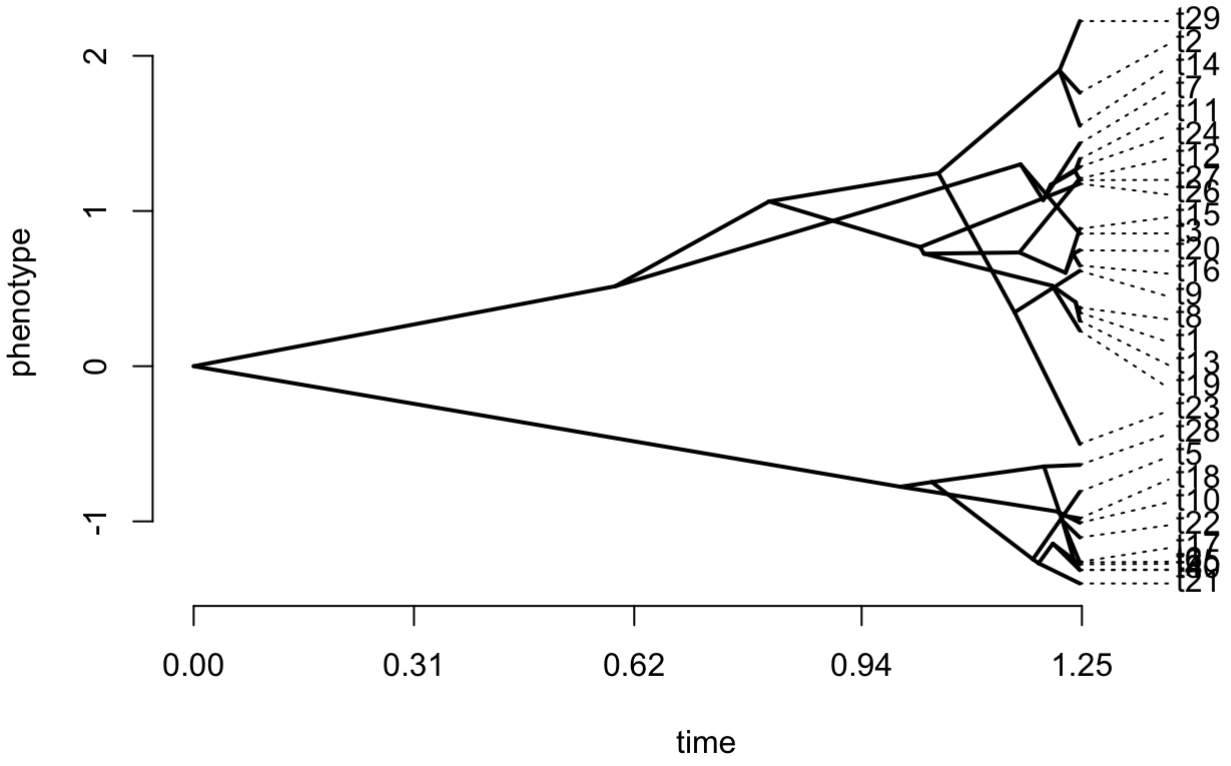


In these simulations, we have discussed the relationships between time, trait values/ phenotype, and variance among simulations. For each of these 3 terms, describe how they would relate to a phylogenetic ancestral state reconstruction of a quantitative trait such as leaf size.

```
# time:
## How long the leaf size has evolved.

# trait values/ phenotype:
## The amount of changes of the leaf size over time.

# variance:
## Given the same amount of time, if we have several lineages evolving parallelly, th
e variance indicates the differnece between these parallel lineages.
```

There's a funtion for simulating Brownian Motion over a phylogeny in Phytools. Given a tree and your Brownian Motion parameters, you can use this function to simulate under the model. Here, we'll simulate a tree, simulate a character evolving over the tree, and plot the tree and associated character values as a 'traitgram' (Ackerly, 2009).

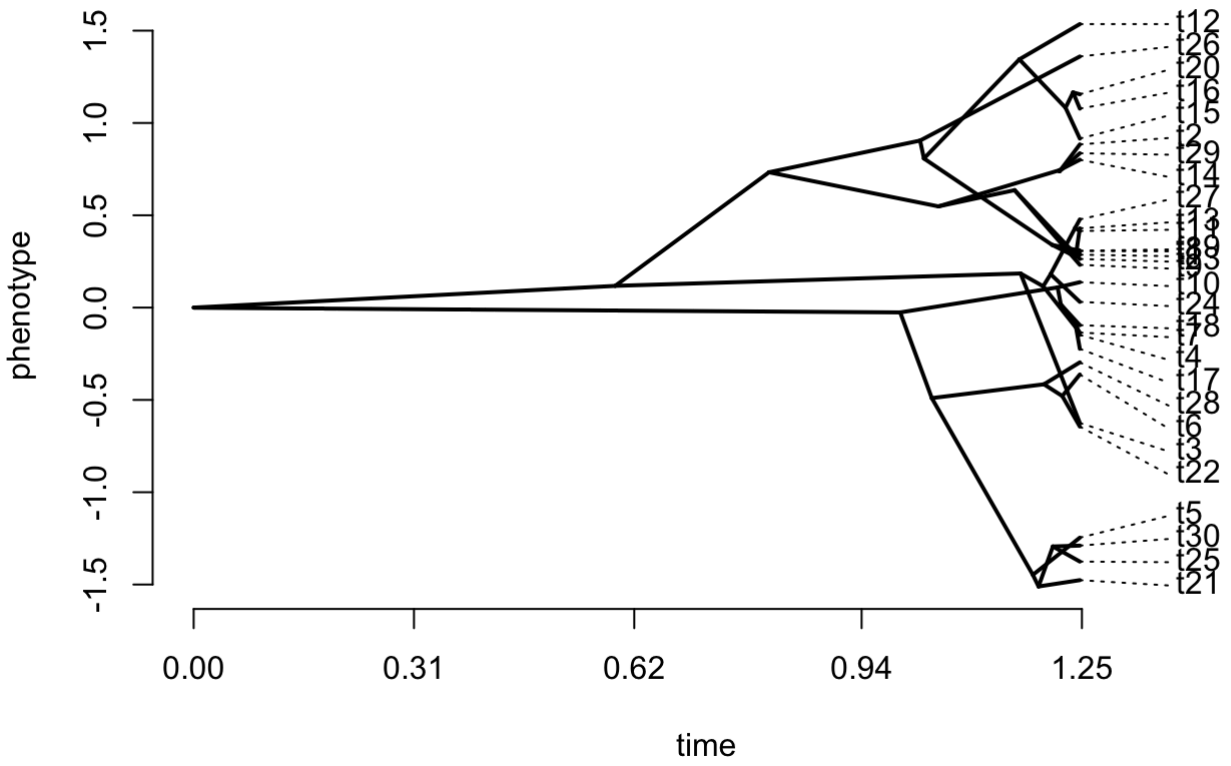
```
tree <- rcoal(n = 30)

x <- fastBM(tree, a=0, sig2=1.0, internal = TRUE)

phenogram(tree, x, spread.labels = TRUE)
```



Let's now simulate a character evolving under the Ornstein-Uhlenbeck process. The OU model is Brownian motion but with two extra parameters: the optimum (theta) and the strength of selection (alpha). We simulate under this model using the same fastBM model as above.

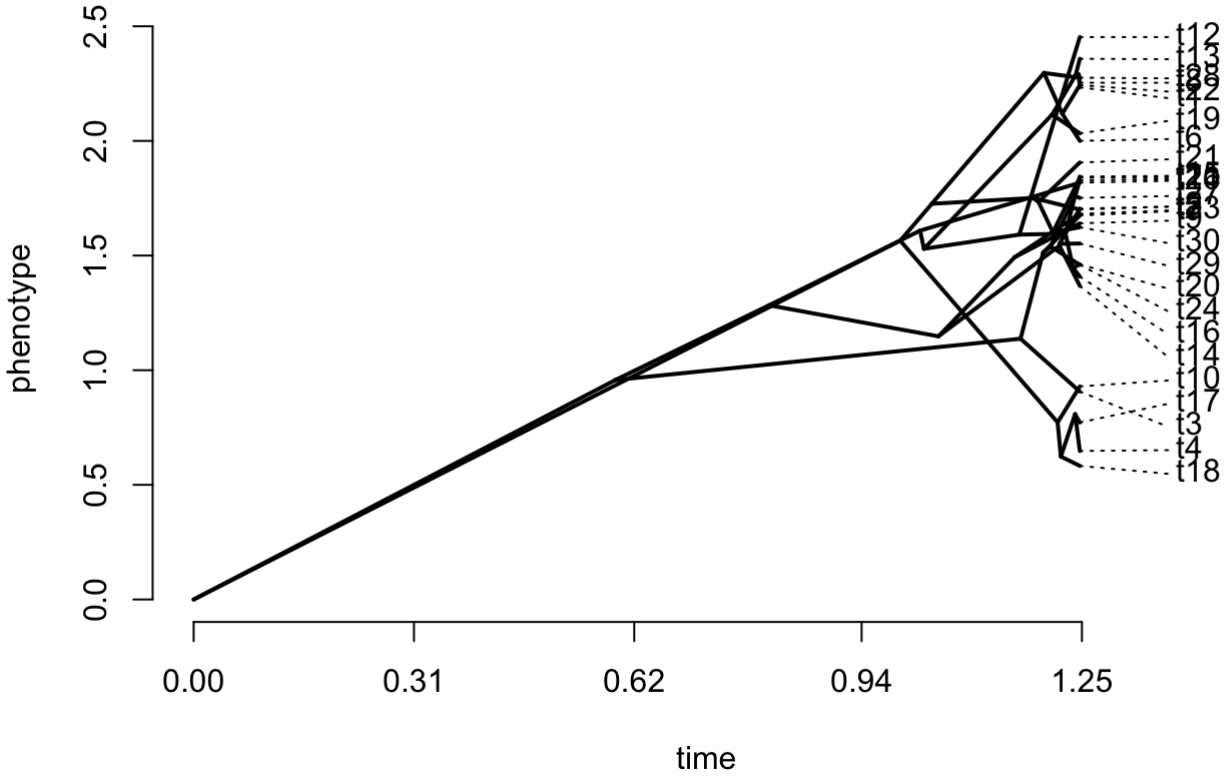
When alpha is close to 0, the OU model collapses down to Brownian motion:

```
x <- fastBM(tree, a=0, sig2=1.0, alpha=0.01, theta=4.0, internal = TRUE)

phenogram(tree, x, spread.labels = TRUE)
```



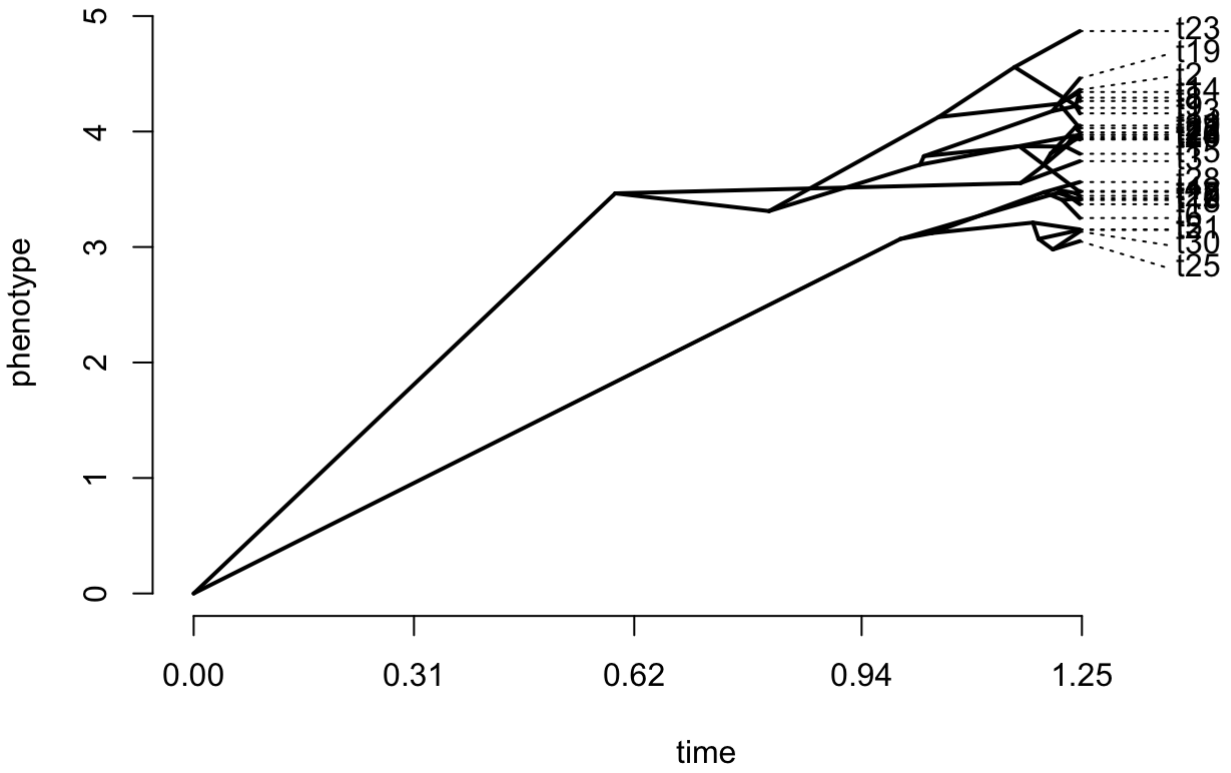
As alpha increase, we see more of a trend towards the optimum:

```
x <- fastBM(tree, a=0, sig2=1.0, alpha=0.5, theta=4.0, internal = TRUE)

phenogram(tree, x, spread.labels = TRUE)
```



Alpha acts as a ‘rubber band’, pulling the trait to the optimum:

```
x <- fastBM(tree, a=0, sig2=1.0, alpha=2.0, theta=4.0, internal = TRUE)

phenogram(tree, x, spread.labels = TRUE)
```



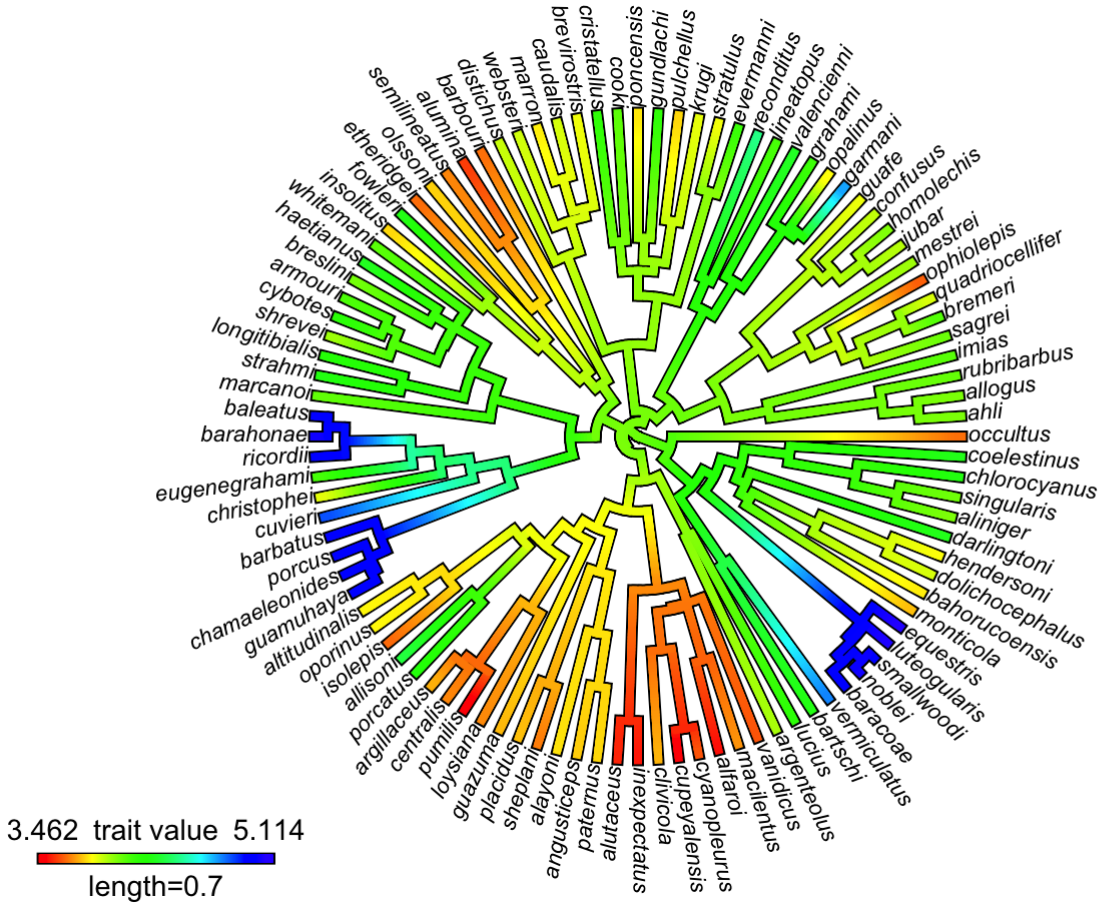
The above examples of Brownian motion and Ornstein-Uhlenbeck assume that the same model applies to the entire tree. It is also possible to have time or branch heterogenous models, in which the parameter values or model changes over the tree. If you are interested in these models, check out the R packages OUwie, ouch, and bayou.

Now that we have a good understanding of Brownian Motion, let's apply the model to reconstruction ancestral states for continous traits. We will use sample data on Anoles again.

```
anole_tree<-read.tree("http://www.phytools.org/eqg2015/data/anole.tre")

svl <- read.csv("http://www.phytools.org/eqg2015/data/svl.csv",
    row.names=1)

svl <- as.matrix(svl)[,1] #convert dataframe to a vector
```

We've read in the tree and character data. Feel free to take a look at those data files to be sure you're comfortable with how they are formatted, etc. Now, we will use the fastAnc funciton in Phytools to reconstruct the ancestral states using maximum likelihood. You can read more about the specifics of the function by looking at the help page. contMap uses the same method as fastAnc but projects those values along the branches of the tree to produce the figure showing changes along branches. Again, read more in the help page for the function if you are interested.

```
fit <- fastAnc(anole_tree,svl,vars=TRUE,CI=TRUE)

fit_obj <- contMap(anole_tree, svl, plot=FALSE)

plot(fit_obj, type="fan", legend=0.7*max(nodeHeights(anole_tree)),
    fsize=c(0.7,0.9))
```



There are several functions that reconstruct ancestral states in R. Another commonly used function is ace from the ape package. Feel free to explore these additional functions if you like.

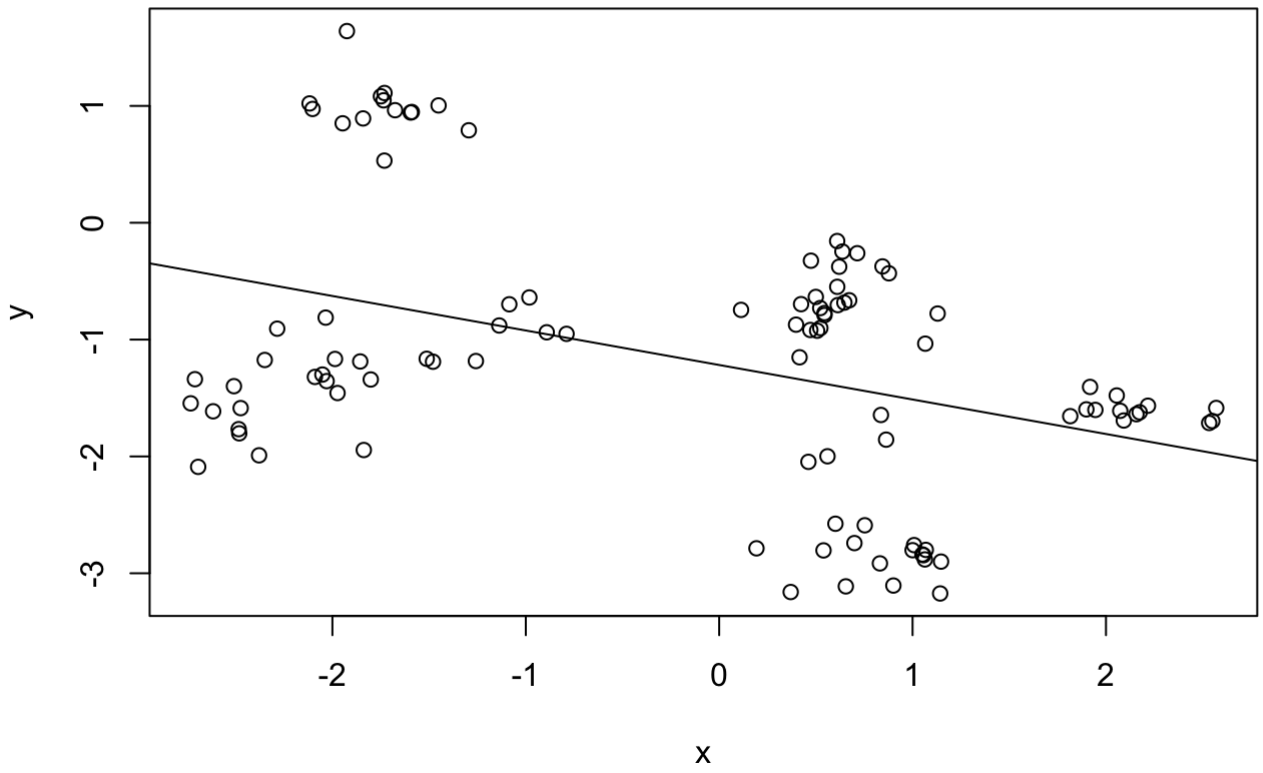
Part 4: Correlated Evolution of Continuous Characters

How do we test if two traits are evolving in a correlated manner? Let's simulate a tree and two traits independently evolving on the tree:

```
tree <- rcoal(n = 100)

x <- fastBM(tree)

y = fastBM(tree)
```

Are the traits correlated if we ignore the phylogeny?

```
plot(x, y)

abline(lm(y ~ x))
```




```
cor.test(x, y)
```

```
##
##  Pearson's product-moment correlation
##
## data:  x and y
## t = -4.3002, df = 98, p-value = 4.035e-05
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.5516686 -0.2191550
## sample estimates:
##       cor
## -0.3984226
```

I just got p-value = 0.02679, but we know these two traits evolved independently (because we simulated them that way! You can see how easy it is to get a type I error (false positive). As an alternative, let's use phylogenetically independent contrasts (PIC; Felsenstein 1985) to estimate the evolutionary correlation between characters:

```
x_c <- pic(x, tree)

y_c <- pic(y, tree)

cor.test(x_c, y_c)
```

```
##
##  Pearson's product-moment correlation
##
## data:  x_c and y_c
## t = 0.51695, df = 97, p-value = 0.6064
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.1465114  0.2472697
## sample estimates:
##       cor
## 0.05241648
```

Now I got a p-value = 0.4457, so we correctly reject the hypothesis that the two characters were correlated.

Content in this lab is drawn from the IB200 2016 lab by Will Freyman, the Phytools blog, and an exercise co-written by Jenna Baughman and Carrie Tribble.