Enumeration and Simulation of Random Tree Topologies

Emmanuel Paradis

August 21, 2024

Contents

This document describes how phylogenetic trees are simulated in ape and discusses some issues related to solving multichotomies.

1 Theoretical Background

We use here the term *topology* to mean the shape of a tree without consideration of its branch lengths. The number of possible topologies for n tips (or leaves), denoted as N_n, is notoriously large. This has been studied by Felsenstein [1] and later reviewed in his book [2]. A particularly relevant case is that of rooted, binary, labeled trees where the number of possible topologies is given by the double factorial:

$$N_n = (2n-3)!! = 1 \times 3 \times 5 \times \cdots \times 2n-3. \quad (1)$$

The function `howmanytrees` in ape calculates the number of possible topologies with respect to three criteria: rooted *vs.* unrooted, binary *vs.* multichotomous, and labeled *vs.* unlabeled. These three options should make eight possible cases, but the number of unlabeled topologies can be computed only for rooted, binary trees, so only five cases are effectively considered. The returned number is obviously an integer which grows dramatically with n:[1]

```
> library(ape)
> n <- 1:10
> data.frame(N.topo = sapply(n, howmanytrees))
   N.topo
1       1
2       1
3       3
```

[1] This function is not vectorized, so we use here `sapply`. We could also use `Vectorize(howmanytrees)(n)`.

```
4        15
5       105
6       945
7     10395
8    135135
9   2027025
10 34459425
```

With ten labeled tips, a rooted, binary phylogenetic tree has more than 34 million possible topologies. Suppose we analyze a sample of $n = 25$ sequences (a very modest sample size in today's standards), this number would be:

```
> howmanytrees(25)
[1] 1.192568e+30
```

Imagine we want to print each of these topologies on an A4 paper sheet and arrange them side by side to make a (huge) poster. This would require (in meters):

```
> howmanytrees(25) * 0.21
[1] 2.504393e+29
```

The diameter of the observable Universe[2] is estimated to be $\approx 8.8 \times 10^{26}$ m, so it would not be wide enough to display our trees.

The number of topologies grows much more gently if they are unlabeled, but still reaching huge values if n is moderately large:

```
> data.frame(N.topo = sapply(n, howmanytrees, labeled = FALSE))
   N.topo
1       1
2       1
3       1
4       2
5       3
6       6
7      11
8      23
9      46
10     98
```

```
> n2 <- c(50, 100, 200)
> data.frame(N.topo = sapply(n2, howmanytrees, labeled = FALSE),
+            row.names = n2)
          N.topo
50  5.150149e+16
100 1.019560e+36
200 1.141703e+75
```

The formula is more complicated than for the labeled cases and is computed by recursion setting $N_1 = 1$:

[2] https://en.wikipedia.org/wiki/Observable_universe

$$N_n = \begin{cases} N_1 N_{n-1} + N_2 N_{n-2} + \cdots + \dfrac{N_{n/2}^2 + 1}{2} & n \text{ even} \\ N_1 N_{n-1} + N_2 N_{n-2} + \cdots + N_{(n-1)/2} N_{(n+1)/2} & n \text{ odd} \end{cases}$$

These numbers are so large that the probability to randomly draw twice the same topology when drawing randomly (and independently) two trees is very close to zero:

```
> 1 / howmanytrees(50)
[1] 3.632505e-77
```

However, for small values of n, this is clearly not negligible:

```
> 1 / sapply(n, howmanytrees)
 [1] 1.000000e+00 1.000000e+00 3.333333e-01 6.666667e-02 9.523810e-03
 [6] 1.058201e-03 9.620010e-05 7.400007e-06 4.933338e-07 2.901964e-08
```

These probabilities are of course larger if the topologies are unlabeled:

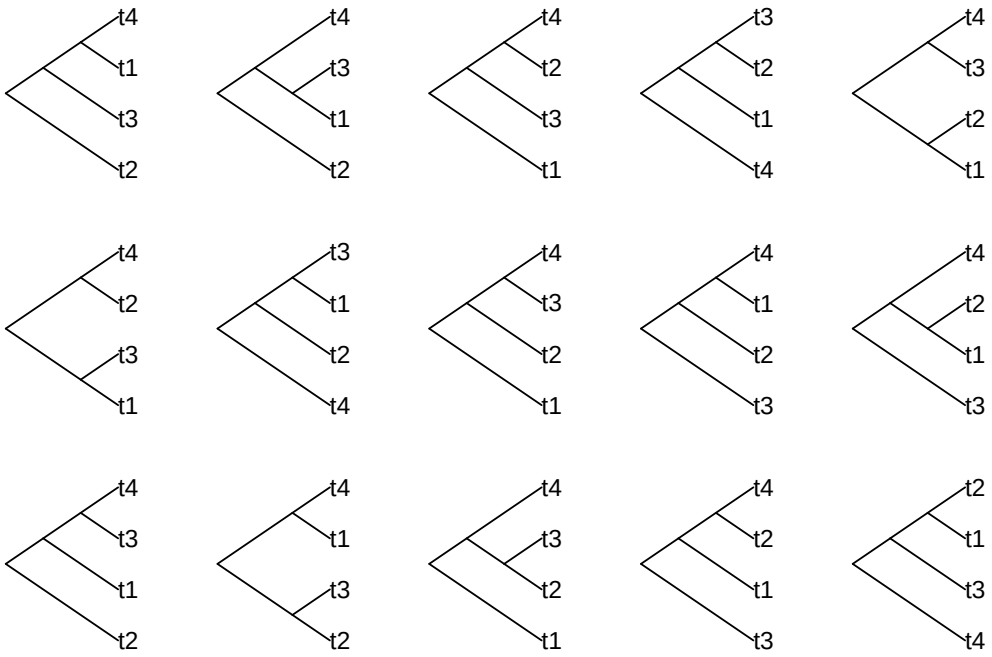
```
> 1 / sapply(n, howmanytrees, labeled = FALSE)
 [1] 1.00000000 1.00000000 1.00000000 0.50000000 0.33333333 0.16666667
 [7] 0.09090909 0.04347826 0.02173913 0.01020408
```

The potential problem is when sampling a large number of trees, their topologies might not be represented adequately if the algorithm does not draw them with equal probability.

phangorn has the function `allTrees`, but, very wisely, it accepts only values of $n \leq 9$ (or 10 if the trees are unrooted, which is the default for this function). We consider here $n = 4$, so there are 15 possible labeled topologies and 2 possible unlabeled topologies:

```
> library(phangorn)
> TR <- allTrees(4, rooted = TRUE)
> TR
15 phylogenetic trees
```

```
> layout(matrix(1:15, 3, 5))
> par(mar = rep(1.5, 4), xpd = TRUE)
> for (i in 1:15) plot(TR[[i]], "c", cex = 1.2, font = 1)
```



This shows that the two unlabeled topologies do not have the same number of labeled cases: there are 12 for the unbalanced topologies, and 3 for the balanced one. Thus, if we sample the unlabeled topologies with equal probabilities, the labeled ones will not be represented in equal frequencies. However, we have seen above that the probability to draw twice the same topology is very small if n is sufficiently large, but this will also depend on the number of trees sampled, denoted here as K.

What is the probability P to have at least twice the same unlabeled topology when drawing K trees assuming equal probability of sampling? We have seen above that the probability to draw a specific topology is the inverse of the possible number of topologies, so the number of times any topology is drawn out of K draws follows a binomial distribution with parameters K and $1/N_n$. The probability to have at least twice this topology is one minus the probability to have it zero time and minus the probability to have it once. Let's calculate this for $n = 50$ and $K = 100$:

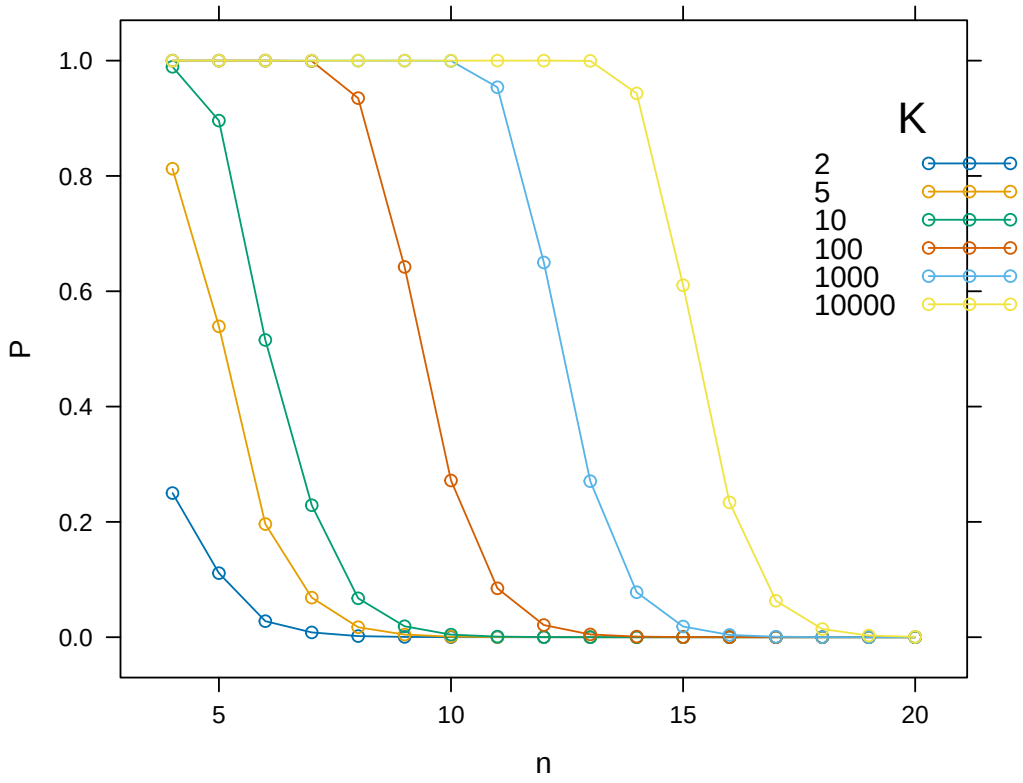
```
> K <- 100
> n <- 50
> N <- howmanytrees(n, labeled = FALSE)
> p <- 1/N
> 1 - (dbinom(0, K, p) + dbinom(1, K, p))
[1] 0
```

These calculations are very close to the numerical precision of most computers, which may result in (very small) negative probabilities, for instance, if we omit the parentheses in the last expression:

```
> 1 - dbinom(0, K, p) - dbinom(1, K, p)
[1] -5.431212e-17
```

We now assess these probabilities for a range of values of n and K:

```
> f <- function(n, K) {
+     N <- howmanytrees(n, labeled = FALSE)
+     p <- 1/N
+     1 - (dbinom(0, K, p) + dbinom(1, K, p))
+ }
> DF <- expand.grid(n = 4:20, K = c(2, 5, 10, 100, 1e3, 1e4))
> DF$P <- mapply(f, n = DF$n, K = DF$K)
> library(lattice)
> xyplot(P ~ n, DF, groups = K, panel = panel.superpose,
+     type = "b", auto.key = list(x = .8, y = .9, title = "K"))
```



We can print the probabilities for a specific value n, or calculate them for other values of n and K:

```
> subset(DF, n == 20)
     n     K             P
17  20     2 1.160494e-11
34  20     5 1.160491e-10
51  20    10 5.222149e-10
68  20   100 5.743191e-08
85  20  1000 5.783570e-06
102 20 10000 5.671842e-04
```

```
> f(20, 1e5)
[1] 0.04638755
```

```
> f(30, 1e6)
[1] 2.525152e-07
```

If the number of tips is small ($n < 20$), we cannot ignore the probability to draw more than once the same topology unless a small number of trees are sampled (e.g., $K < 10$ for $n = 10$). On the other hand, if $n \geq 20$ it is unlikely to draw twice the same topology even when sampling 10,000 trees. We have to keep in mind that these results were obtained assuming that each unlabeled topology has the same probability to be sampled.

2 Simulating Trees with ape

There are five functions to simulate trees in `ape` 5.4:

`rtree`	random topology with fixed n and random branch lengths
`rcoal`	random coalescent tree with fixed n
`rphylo`	random birth–death tree with fixed n
`rbdtree`	random birth–death tree with fixed time and only the surviving lineages
`rlineage`	random birth–death tree with fixed time and all lineages

`rcoal`, `rphylo`, and `rbdtree` always generate ultrametric trees (also `rlineage` if the death rate is equal to zero). All functions generate fully binary trees.[3]

2.1 The function `rtree`

We focus on the function `rtree` since it is the most general one (the other functions have an underlying evolutionary model). Its algorithm is [3]:

1. Draw randomly an integer a on the interval $[1, n-1]$. Set $b = n - a$.
2. If $a > 1$, apply (recursively) step 1 after substituting n by a.
3. Repeat step 2 with b in place of a.
4. Assign randomly the n tip labels to the tips.

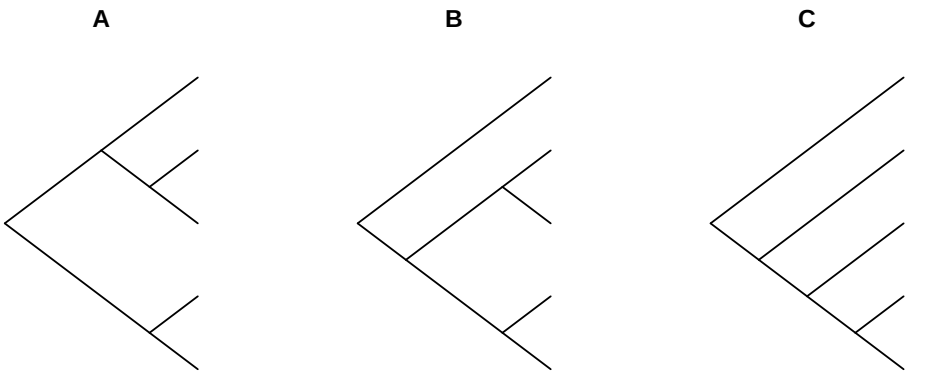
These recursions are used to build the successive splits generating sister-clades. This algorithm is thus well-suited to simulate rooted trees. One of its side-effects is to produce topologies in unequal frequencies. The reason for this is how the integer a is drawn. Suppose that $n = 4$ (we detailed both the labeled and the unlabeled cases above), then during the first iteration a takes the value 1, 2, or 3 with equal probabilities ($\frac{1}{3}$). Obviously, if $a = 2$ a balanced topology will be generated, otherwise the topology will be unbalanced. Consequently, the expected frequencies of these two are $\frac{1}{3}$ and $\frac{2}{3}$ instead of $\frac{1}{2}$ and $\frac{1}{2}$. This also does not agree with the proportions of labeled topologies which should be $\frac{1}{5}$ and $\frac{4}{5}$ of the unlabeled ones.

For larger values of n, these expected frequencies can be calculated in a straightforward way thanks to the recursive nature of the algorithm. If $n = 5$, there are three possible unlabeled topologies:

```
> txt <- c("((,),((,),));", "(((,),(,)),);", "((((,),),),);")
> TR5 <- read.tree(text = txt)
```

[3] The function `stree` generates trees with specific shapes, including multichotomies.

```
> layout(matrix(1:3, 1))
> for (i in 1:3) plot(TR5[[i]], "c", cex = 1.2, main = LETTERS[i])
```



During the first iteration, a takes the value 1, 2, 3, or 4 with equal probabilities ($\frac{1}{4}$): $a = 2$ or 3 leads to topology A, so that it will occur in proportion $\frac{1}{2}$. On the other hand, if $a = 1$ or 4, the above reasoning for $n = 4$ can be applied so that topology B will appear in expected proportion $\frac{1}{2} \times \frac{2}{3} = \frac{1}{3}$, and topology C in proportion $\frac{1}{2} \times \frac{1}{3} = \frac{1}{6}$. Since there are 105 labeled topologies with $n = 5$, we skip their detailed analysis.

In `ape` 5.4-1, the option `equiprob` was introduced to `rtree` to generate topologies in equal proportions.[4] The modification was simple: in step 1 a is drawn uniformly on the interval $[1, \lfloor n/2 \rfloor]$. If $n = 4$, a now takes the value 1 or 2 with equal probabilities ($\frac{1}{2}$), so the two unlabeled topologies are generated in equal proportions. However, if $n = 5$, a takes the same values in the same probabilities leading to incorrect proportions: the draw $a = 1$ should happen twice more frequently than $a = 2$ because the former leads to two different topologies in the following iteration (because $b = 4$) of the algorithm whereas the latter always result in topology A. The solution is to weigh the probabilities by the number of possible topologies for associated to each (a, b) so that those resulting in higher number of topologies will be more likely to be drawn. For $n = 5$, the two possible pairs are $(1, 4)$ and $(2, 3)$; the product of the number of unlabeled topologies gives the correct weights:

```
> howmanytrees(1, labeled = FALSE) * howmanytrees(4, labeled = FALSE)
[1] 2
```

```
> howmanytrees(2, labeled = FALSE) * howmanytrees(3, labeled = FALSE)
[1] 1
```

For instance, if $n = 803$ it will be necessary to calculate the following quantity during the first iteration:

```
> howmanytrees(400, labeled = FALSE) * howmanytrees(403, labeled = FALSE)
[1] Inf
```

[4] The default of this option is `FALSE` because several packages use `rtree()` with `set.seed()` in their examples.

Section 3 explains how these large numbers can be handled. This scheme has been incorporated into `ape` 5.5. These probabilities are passed to the function `sample` and its option `prob` to generate the values of a.

2.2 The new function `rtopology`

`ape` 5.5 introduces `rtopology` in order to provide an alternative to the above functions. The implemented algorithm is as follows:

1. Initialize an unrooted tree with $k = 3$ tips. Set the number of branches to 3.
2. Select randomly one branch of the tree and graft onto it one terminal branch. Increment k by 1 tip and the number of branches by 2.
3. Repeat step 2 until $k = n$.

This algorithm is particularly efficient to generate unrooted trees (this is the default of the function). A rooted tree is generated with one additional step:

4. Select randomly one branch of the tree and add an internal node on it.

Some calculations are simplified compared to `rtree`: it is possible to generate all needed random numbers at once since it is known in advance that we will need to select among 3, 5, 7, ... branches in step 2. Additionally, the tip labels are permuted at the start of the simulation so they are added in random order during step 2. However, `rtree` is faster than `rtopology` because the latter requires to reorder the tree edges after the simulation algorithm is performed.

What are the expected frequencies of the different topologies under this algorithm? Let's consider $n = 4$. Step 1 generates a tree with three terminal branches, then step 2 adds the last terminal branch on any of these three, so that the three possible unrooted, labeled topologies are produced with equal probabilities. The final unrooted tree has five branches: one internal and four terminal. If it is rooted (step 4), then the root node is added randomly on any of these branches, therefore resulting in the balanced topology in 20% of the cases, and one of the unbalanced topologies in the remaining 80%. This conforms to the numbers of labelled topologies found above.

2.3 Tests

The code below is a simple example of testing the frequencies of topologies generated by the above question considering rooted trees with $n = 4$. We first simulate 1000 trees without branch lengths using the three algorithms detailed above:

```
> TR <- list(rmtopology(1000, 4, TRUE, br = NULL),
+            rmtree(1000, 4, TRUE, br = NULL),
+            rmtree(1000, 4, TRUE, br = NULL, equiprob = TRUE))
```

We then build a function to extract the unique topologies and compute their frequencies. The returned value is the χ^2-test for equal frequencies. An option controls whether to consider the toplogies as labeled (the default) or not:

```
> foo.test <- function(x, labeled = TRUE) {
+     uTR <- unique(x, use.tip.label = labeled)
+     f <- integer(length(uTR))
+     for (j in seq_along(uTR)) {
+         for (i in seq_along(x)) {
+             if (all.equal(x[[i]], uTR[[j]], use.tip.label = labeled))
```

```
+                 f[j] <- f[j] + 1L
+         }
+     }
+     print(f)
+     chisq.test(f)
+ }
```

We now apply the function to the list of simulated trees.

```
> lapply(TR, foo.test)
 [1] 76 81 64 65 68 72 67 73 70 68 51 60 66 50 69
 [1]  59  59  53  62 136  48  59  62  53  48  97  59 111  49  45
 [1]  41 166 161 165  42  41  43  53  43  55  52  35  31  33  39
[[1]]

        Chi-squared test for given probabilities

data:  f
X-squared = 14.39, df = 14, p-value = 0.4211


[[2]]

        Chi-squared test for given probabilities

data:  f
X-squared = 147.35, df = 14, p-value < 2.2e-16


[[3]]

        Chi-squared test for given probabilities

data:  f
X-squared = 542.9, df = 14, p-value < 2.2e-16
```

```
> lapply(TR, foo.test, labeled = FALSE)
[1] 793 207
[1] 656 344
[1] 508 492
[[1]]

        Chi-squared test for given probabilities

data:  f
X-squared = 343.4, df = 1, p-value < 2.2e-16


[[2]]

        Chi-squared test for given probabilities
```

```
data:  f
X-squared = 97.344, df = 1, p-value < 2.2e-16


[[3]]

        Chi-squared test for given probabilities

data:  f
X-squared = 0.256, df = 1, p-value = 0.6129
```

The results agree with the above analysis. The same simulation study with $n = 5$ would require larger sample sizes because of the much larger numbers of possible topologies.

2.4 Summary

1. `rtree` simulates topologies in unequal frequencies. The differences in expected frequencies are important for small values of n (see above for $n = 4$ and $n = 5$). For larger values of n, it is practically impossible to determine these frequencies because of the large number of possible topologies. However, it seems these frequencies are sufficiently low that the above analysis assuming equal probabilities might be a good approximation. For instance, `rmtree(1000, 20)` returns 1000 unique unlabeled topologies (not shown here to avoid long computations in this vignette).

2. `rtree(, equiprob = TRUE)` simulates unlabeled topologies in equal frequencies (from `ape` 5.5).

3. `rtopology` simulates labeled topologies with equal probabilities. This is the function now called by `multi2di(, equiprob = TRUE)`.

3 Enumerating Topologies Beyond Large Values

The function `howmanytrees`, like most functions in R, works with double precision floating point numbers, so the largest value which can be returned is:

```
> .Machine$double.xmax
[1] 1.797693e+308
```

This value is reached with $n = 151$ or 792 for labeled or unlabeled topologies, respectively:

```
> howmanytrees(151)
[1] 3.753274e+306
```

```
> howmanytrees(792, labeled = FALSE)
[1] 1.029038e+308
```

This section shows how larger values are handled in `ape` 5.5.

3.1 Labeled Topologies

It is possible to write equation 1 using standard factorial (this uses the fact that $2n-3$ is always an odd integer):

$$(2n-3)!! = \frac{(2n-3)!}{2^{n-2}(n-2)!} \qquad n \geq 2.$$

A log-transformation of both sides leads to:

$$\ln[(2n-3)!!] = \ln[(2n-3)!] - (n-2)\ln 2 - \ln[(n-2)!]. \tag{2}$$

R has the function `lfactorial` making possible to calculate this expression with large values of n.[5] Obviously, if we try to compute the exponential of the result, we will again obtain `Inf`. Instead, we can use the fact that an expression such as a^b can be expressed with a power of ten:

$$a^b = x \times 10^n,$$

with x a real number and n an integer. We do a $\log_{10}$-transformation of both sides:

$$b \times \log_{10} a = \log_{10} x + n.$$

This is solved with $n = \lfloor b \times \log_{10} a \rfloor$ and $x = 10^c$, $c = b \times \log_{10} a - n$. This can be used with $a = \mathrm{e}$ (≈ 2.718) and $b = \ln[(2n-3)!!]$ calculated with equation 2.

Let's find how many possible trees there are for the number of living species on Earth using an estimate of two million species:[6]

```
> (N <- howmanytrees(2e6))

approximately 3.692364 * 10^12335524
```

How many digits are needed to print this number?

```
> N[2] + 1

       n
12335525
```

How many pages to print it (assuming 3000 characters per page)?

```
> ceiling((N[2] + 1) / 3000)

   n
4112
```

3.2 Unlabeled Topologies

In the unlabeled case, the formulas to calculate the number of topologies are more complicated and do not allow the previous approach. However, there is an almost linear relationship between n and N_n after log-transformation of the latter:

```
> n <- 1:792
> N <- howmanytrees(792, labeled = FALSE, detail = TRUE)
> lN <- log10(N)
> summary(lm(lN ~ n))
```

[5] `phangorn` has the function `ldfactorial` doing the same calculation based on `lgamma`.
[6] This returned `Inf` prior to `ape` 5.5.

```
Call:
lm(formula = lN ~ n)

Residuals:
     Min       1Q   Median       3Q      Max
-0.25968 -0.21424 -0.08311  0.12406  2.82605

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -3.219e+00  2.258e-02  -142.5   <2e-16 ***
n            3.926e-01  4.934e-05  7956.3   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.3175 on 790 degrees of freedom
Multiple R-squared:      1,        Adjusted R-squared:      1
F-statistic: 6.33e+07 on 1 and 790 DF,  p-value: < 2.2e-16
```

The linear approximation is good but it is still better for $n > 700$:

```
> s <- n > 700
> summary(lm(lN[s] ~ n[s]))

Call:
lm(formula = lN[s] ~ n[s])

Residuals:
       Min         1Q     Median         3Q        Max
-0.0004133 -0.0003323 -0.0001032  0.0002834  0.0008197

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -4.153e+00  1.096e-03   -3788   <2e-16 ***
n[s]         3.941e-01  1.468e-06  268567   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.0003738 on 90 degrees of freedom
Multiple R-squared:      1,        Adjusted R-squared:      1
F-statistic: 7.213e+10 on 1 and 90 DF,  p-value: < 2.2e-16
```

The residuals are now almost equal to zero. We may thus write:

$$\log_{10} N_n \approx 0.3941 \times n - 4.153 \qquad n > 792.$$

Let's evaluate the quality of this approximation for a few values:

```
> n <- 788:792
> log10(sapply(n, howmanytrees, labeled = FALSE))
[1] 306.4356 306.8298 307.2240 307.6182 308.0124


> 0.3941 * n - 4.153
[1] 306.3978 306.7919 307.1860 307.5801 307.9742
```

The approximation is not so good for smaller values of n and we use it only for $n > 792$. This can be used to handle some of the calculations detailed in Section 2.1. The coefficient (0.3941) is actually the difference between successive values of N_n on the logarithmic scale:

$$\log_{10} N_{n+1} - \log_{10} N_n = 0.3941.$$

This is equivalent to a ratio of these successive values equal to $N_{n+1}/N_n = 10^{0.3941} = 2.477993$. We denote this last number as ξ. This can help to find an approximation of the sum of the N_i's for $i = 1, \ldots, n$, when n is large:

$$\begin{aligned}\sum_{i=1}^{n} N_i &= N_1 + N_2 + N_3 + \cdots + N_n \\ &\approx N_1 + \xi N_1 + \xi^2 N_1 + \cdots + \xi^{n-1} N_1 \\ &= N_1 \sum_{i=1}^{n} \xi^{i-1}.\end{aligned}$$

This is a divergent geometric series ($\xi > 1$). Since $N_1 = 1$, this first term can be ignored. The logarithm of this sum may be approximated using the fact that $\log(x + \delta) \approx \log(x)$ if $x >>> \delta$. More generally, if we have a series of numbers A, B, C, ..., such that:

$$A > B > C > \ldots$$

then the logarithm of their sum can be approximated with successive calculations (the base of the logarithm is unimportant):

$$\begin{aligned}\log(A + B + C + \ldots) &\approx \log(A) \\ &\approx \log(A + B) \\ &\approx \log(A + B + C) \\ &\approx \ldots\end{aligned}$$

In other words, when calculating the logarithm of a sum, the largest terms will contribute the most. It is possible to assess the convergence of the successive approximations.

$$\begin{aligned}\log_{10}\left(\sum_{i=1}^{n} N_n\right) &\approx \log_{10}(\xi^{n-1}) = (n-1)\log_{10}\xi \\ &\approx \log_{10}(\xi^{n-2} + \xi^{n-1}) = (n-2)\log_{10}\xi + \log_{10}(1+\xi) \\ &\approx \log_{10}(\xi^{n-3} + \xi^{n-2} + \xi^{n-1}) = (n-3)\log_{10}\xi + \log_{10}(1+\xi+\xi^2) \\ &\approx \ldots\end{aligned}$$

The kth approximation can be calculated directly with:

$$(n-k)\log_{10}\xi + \log_{10}\left(\sum_{i=0}^{k-1} \xi^i\right) \qquad k > 1$$

This converges after a few iterations:

```
> xi <- 2.477993
> lxi <- log10(xi)
> n <- 1000
> (n - 1) * lxi # k = 1
```

```
[1] 393.706


> for (k in 2:11)
+     print((n - k) * lxi + log10(sum(xi^(0:(k - 1)))))
[1] 393.8532
[1] 393.9009
[1] 393.9187
[1] 393.9257
[1] 393.9285
[1] 393.9296
[1] 393.9301
[1] 393.9303
[1] 393.9304
[1] 393.9304
```

References

[1] J. Felsenstein. Number of evolutionary trees. *Systematic Zoology*, 27(1):27–33, 1978.

[2] J. Felsenstein. *Inferring Phylogenies*. Sinauer Associates, Sunderland, MA, 2004.

[3] E. Paradis. *Analysis of Phylogenetics and Evolution with R (Second Edition)*. Springer, New York, 2012.