New balance indices and metrics for phylogenetic trees - Scripts

Lucia Rotger

Universitat de les Illes Balears

Contents

A.1 Packages required **1**

A.2 List of all binary trees **2**

A.3 General functions **2**

A.4 Scripts from Chapter 2 **3**

A.4.1 Computation of $E_Y(\Phi_n)$ 3
A.4.2 Computation of $E_U(\Phi_n)$ 4
A.4.3 Computation of $\sigma^2_U(\Phi_n)$ 4
A.4.4 Computation of $\sigma^2_U(S_n)$ 8
A.4.5 Computation of $Cov_U(S_n, \Phi_n)$ 11
A.4.6 Computation of the estimated probability of a tie 14
A.4.7 Testing Φ_n on TreeBASE 17

A.5 Scripts from Chapter 3 **19**

A.5.1 Computation of $E(D^2_n)$ 19
A.5.2 Computation of $\sigma^2(D^2_n)$ 21

A.6 Scripts from Chapter 4 **24**

A.6.1 The R package *CollessLike* 24
A.6.2 A real example 40
A.6.3 Computation of the mean and variance 43
A.6.4 Computation of the number of ties 52
A.6.4 Computation of Spearman's rank correlation 53
A test on the distribution of TreeBASE 56

A.1 Packages required

The functions used in this appendix need the following R packages to be installed and loaded:

```r
library(Zseq)
library(gmp)
library(ape)
library(igraph)
library(CollessLike)
```

A.2 List of all binary trees

We have obtained al phylogenetic trees in $\mathcal{BT}_n$ for $n = 3, \dots, 8$ using the *Python* package *phylonetwork*:

```
import phylonetwork.generators as gen
from phylonetwork.distances import cophenetic_distance as cophdist
from math import factorial

for n in range(3,9):
    taxa = [str(i+1) for i in range(n)]
    tg = gen.all_trees(taxa = taxa, binary = True, nested_taxa = False)
    trees = list(tg)
    newicks = []
    file = open("bintrees-n"+str(n)+".txt", "w+")
    for i in range(len(trees)):
        newicks.append(trees[i].eNewick())
        print >>file, newicks[i]
    file.close()
```

The resulting lists of trees are available in the *List of Trees* folder of the GitHub repository.

A.3 General functions

The following functions are needed in some computations performed in the next two sections.

```
big.factorial = function(n){
  if(n<2) return(1)
  return(Factorial(n+1)[n+1])
}

big.double.factorial = function(n){
  if(n<2) return(1)
  m = (n+2+n%%2)/2
  return(Factorial.Double(m,odd=(n%%2==1))[m])
}

big.binomial = function(n,k){
  return(big.factorial(n)/(big.factorial(k)*big.factorial(n-k)))
}

Cknk = function(k,n){
  return(big.binomial(n,k)*((big.double.factorial(2*k-3)*
          big.double.factorial(2*(n-k)-3))/
          (2*big.double.factorial(2*n-3))))
}
```

A.4 Scripts from Chapter 2

A.4.1 Computation of $E_Y(\Phi_n)$

The formula in Theorem 2.20 can be computed with the following function:

```
harmonic=function(n){return(sum(1/(1:n)))}
EYPhi=function(n){
  return(n*(n+1-2*harmonic(n)))
}
```

For $n = 3, ..., 20$ the results are:

```
sapply(3:20,EYPhi)
```

```
##  [1]   1.000000   3.333333   7.166667  12.600000  19.700000  28.514286
##  [7]  39.078571  51.420635  65.562698  81.522944  99.316522 118.956255
## [13] 140.453130 163.816672 189.055214 216.176109 245.185893 276.090414
```

To double-check the formula, we have computed the values of $E_Y(\Phi)$, for $n = 3, \ldots, 8$, from the cophenetic indices of all trees in the corresponding $\mathcal{BT}_n$.

To do that, we have used the full content of $\mathcal{BT}_n$ for $n = 3, \ldots, 8$ obtained in Section A.2. Then, on the one hand, we have computed the probability of each tree under the Yule model with the following function:

```
yule.prob = function(tree){
  if (class(tree)=="phylo")
    tree=graph.edgelist(tree$edge, directed=TRUE)
  sp = shortest.paths(tree,mode = "out")
  deg = degree(tree,mode="out")
  leaves = which(deg==0)
  n = length(leaves)
  k.node = function(node){
    subtree=which(sp[node,]<Inf)
    return(length(intersect(leaves,subtree)))
  }
  kappas = sapply(which(deg>0), k.node)
  value = (2^(n-1)/as.numeric(big.factorial(n)))*
              prod(1/(kappas-1))
  return(value)
}
```

And, on the other hand, we have computed the total cophenetic index of each tree with the function `cophen.index` contained in our R package *CollessLike* (see Section A.6.1). Finally, we have computed the desired expected value for each n as the sum over all phylogenetic trees in BT n of the product of their total cophenetic index and their probability:

```
exp.yule = c()
for(n in 3:8){
  trees=read.tree(file=paste("./bintrees-n",n,".txt",sep=""))
  indices = sapply(trees, cophen.index)
  probs=sapply(trees, yule.prob)
  exp.yule[n]=sum(indices*probs)
}
exp.yule
```

```
## [1]  1.000000  3.333333  7.166667 12.600000 19.700000 28.514286
```

So, the results agree with the figures given by our formula.

A.4.2 Computation of $E_U(\Phi_n)$

The formula in Theorem 2.28 can be computed with the following function (it uses the function `big.double.factorial` explained in Section A.3):

```
EUPhi = function(n){
  return(as.numeric((n*(n-1)/4)*
          (big.double.factorial(2*n-2)/
            big.double.factorial(2*n-3)-2)))
}
```

For $n = 3, ..., 20$ the results are:

```
sapply(3:20,EUPhi)
```

```
##   [1]   1.000000   3.600000   8.285714  15.476190  25.545455  38.834499
##   [7]  55.658741  76.313040 101.075256 130.208893 163.965117 202.584342
##  [13] 246.297504 295.327098 349.888046 410.188417 476.430046 548.809061
```

To double-check the formula, we have computed the values of $E_U(\Phi)$, for $n = 3, \ldots, 8$, as the arithmetic mean of the total cophenetic indices of all phylogenetic trees in $\mathcal{BT}_n$ computed with our function `cophen.index`.

```
exp.uni = c()
for(n in 3:8){
  trees=read.tree(file=paste("./bintrees-n",n,".txt",sep=""))
  indices = sapply(trees, cophen.index)
  exp.uni[n]=mean(indices)
}
exp.uni
```

```
## [1]  1.000000  3.600000  8.285714 15.476190 38.834499
```

Again, the results agree with the figures given by our formula.

A.4.3 Computation of $\sigma^2_U(\Phi_n)$

Computing the variance of Φ_n using our formula

We can compute $\sigma^2_U(\Phi_n)$ using the recurrence for $E_U(\Phi_n^2)$, the exact formula of $E_U(\Phi_2)^2$, and the identity:

$$\sigma^2_U(\Phi_n) = E_U(\Phi_n^2) - E_U(\Phi_2)^2$$

The following functions are needed to compute this variance, in addition to those in Section A.3.

```r
EUPhi = function(n){
    return(as.numeric((n*(n-1)/4)*
            (big.double.factorial(2*n-2)/
              big.double.factorial(2*n-3)-2)))
}

term.Phi = function(n){
  return(mul.bigq(as.bigq(n*(n-1)/2),(mul.bigq(as.bigq(
          (49*n^3-57*n^2-22*n+24)/48),
          big.double.factorial(2*n-4)/big.double.factorial(
            2*n-3))-as.bigq((63*n^2-95*n+28)/30))))
}

compute.EUPhi2 = function(n.max=500){
  terms = lapply(2:n.max,term.Phi)
  terms = c(0,terms)
  exp.values = list(0)
  for(n in 2:n.max){
    sums = 0
    if(n>2){
      for(k in 2:(n-1)){
        sums = sums + Cknk(k,n)*exp.values[[k]]
      }
      sums = 2*sums
    }
    sums = sums + terms[[n]]
    exp.values[[n]] = sums
    print(n)
  }
  exp.values = sapply(exp.values, as.numeric)
  write.table(exp.values,file = paste("C2-EU(Phi2)",n.max,".txt",
                            sep = ""),row.names = F,col.names = F)
  return(exp.values)
}

compute.varUPhi = function(exp.values,n.max=500){
  var.form = function(i)return(exp.values[i]-EUPhi(i)^2)
  var.values = sapply(1:n.max, var.form)
  write.table(var.values,file = paste("C2-varU(Phi)",n.max,".txt",
                            sep = ""),row.names = F,col.names = F)
  return(var.values)
}
```

We have computed these variances up to $n = 1000$ with the following commands:

```r
exp.values.Phi = compute.EUPhi2(1000)
var.values.Phi = compute.varUPhi(exp.values.Phi,1000)
```

For $n = 3, ..., 20$ the results have been:

```
exp.values.Phi[3:20]
```

```
##  [1]      1.00000     13.60000     73.42857    259.09524    711.54545
##  [6]   1654.34965   3414.67413   6444.77869  11343.93737  18880.70867
## [11]  30015.50122  45923.39304  68017.17207  97970.57161 137741.67942
## [16] 189596.50276 256132.67433 340303.28664
```

```
var.values.Phi[3:20]
```

```
##  [1]     0.00000     0.64000     4.77551    19.58277    58.97521
##  [6]   146.23135   316.77865   621.09863  1127.72999  1926.35278
## [11]  3130.94150  4882.97724  7354.71170 10752.47676 15320.03476
## [16] 21341.96545 29147.08600 39111.90118
```

The rest of the values are available in the files "C2-EU(Phi2)1000.txt" and "C2-varU(Phi)1000.txt".

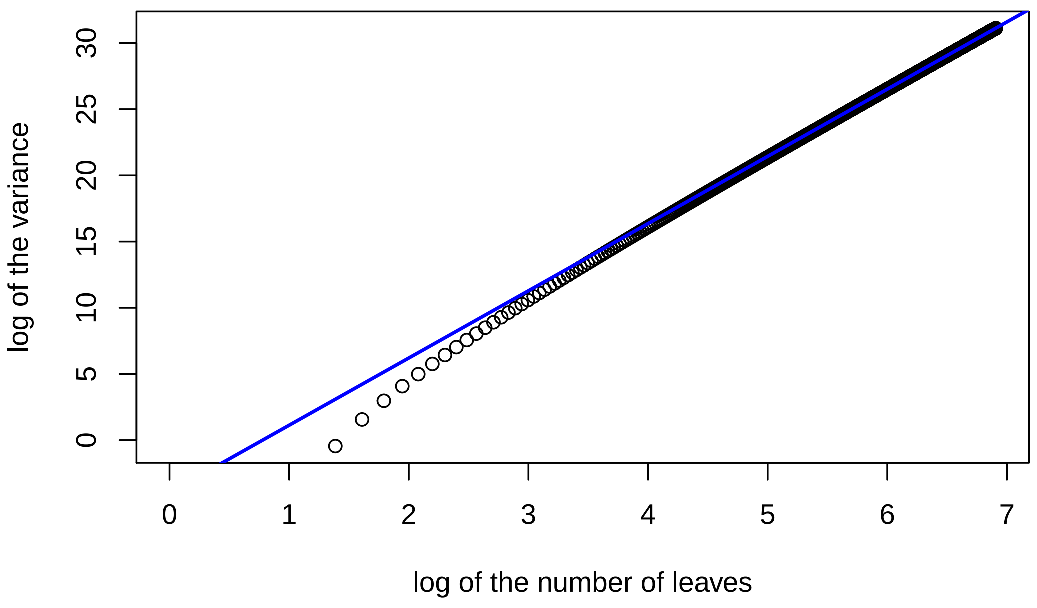
We have estimated the main order in the expansion of $\sigma_U^2(\Phi_n)$ as a function of n, by performing the minimum squares linear regression of $\ln(\sigma_U^2(\Phi_n))$ as a function of $\ln(n)$ for $n = 900, \ldots, 1000$,

```
summary(lm(log(var.values.Phi[900:1000])~log(900:1000)))
```

```
##
## Call:
## lm(formula = log(var.values.Phi[900:1000]) ~ log(900:1000))
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -3.330e-05 -1.131e-05  4.161e-06  1.340e-05  1.671e-05
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -3.8743868  0.0003353  -11555   <2e-16 ***
## log(900:1000)  5.0657352  0.0000489  103586   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.509e-05 on 99 degrees of freedom
## Multiple R-squared:      1,  Adjusted R-squared:      1
## F-statistic: 1.073e+10 on 1 and 99 DF,  p-value: < 2.2e-16
```

The commands below produce Fig. 2.11 that displays $\ln(\sigma_U^2(\Phi_n))$ as a function of $\ln(n)$, together with the corresponding regression line.

```
plot(log(1:1000),log(var.values.Phi),
              xlab="log of the number of leaves",
              ylab="log of the variance")
reg.phi=lm(log(var.values.Phi[500:1000])~log(500:1000))
abline(reg.phi,col="blue",lwd=2)
```



Computing the variance of Φ_n from the cophenetic indices

To double-check the recurrence, we have computed the values of $\sigma_U^2(\Phi_n)$, for $n = 3, \ldots, 8$, from the cophenetic indices of all trees in the corresponding $\mathcal{BT}_n$.

To do this, we have carried out a similar process as in Section A.4.2, replacing the arithmetic mean by the (true) variance:

```
var.n = function (vec)
        return(var(vec)*(length(vec)-1)/length(vec))
trees = list()
all.cophen.index = list()
real.var.Phi = c()
for(n in 3:8){
  trees[[n]] = read.tree(file = paste("bintrees-n",n,".txt",
                                      sep = ""))
  all.cophen.index[[n]] = sapply(trees[[n]],cophen.index)
  real.var.Phi[n] = var.n(all.cophen.index[[n]])
  print(paste("var(Phi",n,") = ",real.var.Phi[n],sep = ""))
}
real.var.Phi
```

```
## [1]    0.00000    0.64000    4.77551   19.58277   58.97521   58.97521 146.23135
```

The results agree with the figures given by our recurrence.

A.4.4 Computation of $\sigma_U^2(S_n)$

Computing the variance of S_n using our formula

We can compute $\sigma_U^2(S_n)$ using the recurrence for $E_U(S_n^2)$, the exact formula of $E_U(S_n)^2$, and the identity

$$\sigma_U^2(S_n) = E_U(S_n^2) - E_U(S_n)^2$$

The following functions are needed to compute this variance, in addition to those in Section A.3.

```
EUS = function(n){
    return(as.numeric(n*(big.double.factorial(2*n-2)/
                           big.double.factorial(2*n-3)-1)))
}

term.S = function(n){
  return((5*n*2^(n-2)*big.factorial(n))/(
          big.double.factorial(2*n-3))-n*(5*n-2))
}

compute.EUS2 = function(n.max=500){
  terms = lapply(2:n.max,term.S)
  terms = c(0,terms)
  exp.values = list(0)
  for(n in 2:n.max){
    sums = 0
    if(n>2){
      for(k in 2:(n-1)){
        sums = sums + Cknk(k,n)*exp.values[[k]]
      }
      sums = 2*sums
    }
    sums = sums + terms[[n]]
    exp.values[[n]] = sums
    print(n)
  }
  exp.values = sapply(exp.values, as.numeric)
  write.table(exp.values,file = paste("C2-EU(S2)",n.max,".txt",
                      sep = ""),row.names = F,col.names = F)
  return(exp.values)
}

compute.varUS = function(exp.values, n.max){
  var.form = function(i)return(exp.values[i]-EUS(i)^2)
  var.values = sapply(1:n.max,var.form)
  write.table(var.values,file = paste("C2-varU(S)",n.max,".txt",
                      sep = ""),row.names = F,col.names = F)
  return(var.values)
}
```

We have computed these variances up to $n = 1000$ with the following commands:

```
exp.values.S = compute.EUS2(1000)
var.values.S = compute.varUS(exp.values.S,1000)
```

For $n = 3, ..., 20$ the results have been:

```
exp.values.S[3:20]
```

```
##  [1]    25.0000    77.6000   177.2857   340.0952   582.4545   921.0816
##  [7]  1372.9245  1955.1189  2684.9571  3579.8650  4657.3837  5935.1556
## [13]  7430.9128  9162.4673 11147.7035 13404.5711 15951.0795 18805.2931
```

```
var.values.S[3:20]
```

```
##  [1]   0.0000000   0.1600000   0.7755102   2.2358277   4.9990817
##  [6]   9.5765183  16.5219346  26.4241938  39.9016992  57.5981796
## [11]  80.1793886 108.3304640 142.7537743 184.1671371 233.3023247
## [16] 290.9037954 357.7276063 434.5404734
```

The rest of the values are available in the files “C2-EU(S2)1000.txt” and “C2-varU(S)1000.txt”.

We have estimated the main order in the expansion of $\sigma_U^2(S_n)$ as a function of n, by performing the minimum squares linear regression of $\ln(\sigma_U^2(S_n))$ as a function of $\ln(n)$ for $n = 900, \ldots, 1000$,

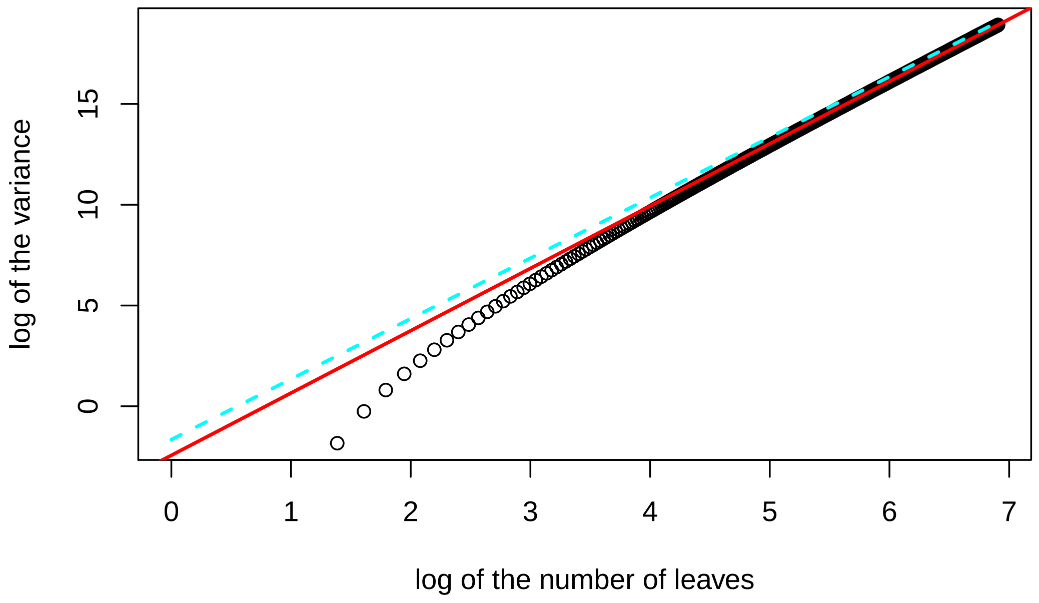
```
summary(lm(log(var.values.S[900:1000])~log(900:1000)))
```

```
##
## Call:
## lm(formula = log(var.values.S[900:1000]) ~ log(900:1000))
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -3.907e-05 -1.327e-05  4.879e-06  1.573e-05  1.961e-05
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -2.347e+00  3.935e-04   -5965   <2e-16 ***
## log(900:1000)  3.079e+00  5.739e-05   53646   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.771e-05 on 99 degrees of freedom
## Multiple R-squared:      1,  Adjusted R-squared:      1
## F-statistic: 2.878e+09 on 1 and 99 DF,  p-value: < 2.2e-16
```

The following code produces Fig. 2.12.

```
plot(log(1:1000),log(var.values.S),
              xlab="log of the number of leaves",
              ylab="log of the variance")
reg.S=lm(log(var.values.S[500:1000])~log(500:1000))
abline(reg.S,col="red",lwd=2)
sackin.approx = ((10-3*pi)/3)*(1:1000)^3
```

```
lines(log(1:1000),log(sackin.approx),col="cyan",lty=2,lwd=2)
```



Computing the variance of S from the Sackin indices

To double-check the recurrence, we have computed the values of $\sigma^2_U(S_n)$, for $n = 3, \ldots, 8$, from the Sackin indices of all trees in the corresponding $\mathcal{BT}_n$. We have proceeded as in Section A.4.3, using the function `sackin.index` in the package *CollessLike* (see Section A.6.1) to compute the Sackin indices:

```
var.n = function (vec)
        return(var(vec)*(length(vec)-1)/length(vec))
trees = list()
all.sackin.index = list()
real.var.sackin = c()
for(n in 3:8){
  trees[[n]] = read.tree(file = paste("bintrees-n",n,".txt",
                                      sep = ""))
  all.sackin.index[[n]] = sapply(trees[[n]],sackin.index)
  real.var.sackin[n] = var.n(all.sackin.index[[n]])
  print(paste("var(S_",n,") = ",real.var.sackin[n],sep = ""))
}
real.var.sackin
```

```
## [1] 0.0000000 0.1600000 0.7755102 2.2358277 4.9990817 9.5765183
```

So, the results agree again with the figures given by our recurrence.

A.4.5 Computation of $Cov_U(S_n, \Phi_n)$

Computing the covariance of S_n and Φ_n using our formula

We can compute $Cov_U(S_n, \Phi_n)$ using the recurrence for $E_U(S_n\Phi_n)$, the exact formula of $E_U(S_n)$ and $E_U(\Phi)$, and the identity

$$Cov_U(S_n, \Phi_n) = E_U(S_n\Phi_n) - E_U(S_n)E_U(\Phi)$$

The following functions are needed to compute this covariance, in addition to `EUPhi` from Section A.4.2, `EUS` from Section Section A.4.4, and the functions in Section Section A.3.

```
term.cov = function(n){
  return(((13*n^2-9*n-2)*2^(n-5)*big.factorial(n))/(
            big.double.factorial(2*n-3))-(n*(n-1)/2)*(5*n-2))
}

compute.EUcov = function(n.max=500){
  terms = lapply(2:n.max,term.cov)
  terms = c(0,terms)
  exp.values = list(0)
  for(n in 2:n.max){
    sums = 0
    if(n>2){
      for(k in 2:(n-1)){
        sums = sums + Cknk(k,n)*exp.values[[k]]
      }
      sums = 2*sums
    }
    sums = sums + terms[[n]]
    exp.values[[n]] = sums
    print(n)
  }
  exp.values = sapply(exp.values, as.numeric)
  write.table(exp.values,file=paste("C2-EU(SxPhi)",n.max,".txt",
                      sep = ""),row.names = F,col.names = F)
  return(exp.values)
}

compute.cov = function(exp.values,n.max = 500){
  cov.form = function(i)return(exp.values[i]-EUS(i)*EUPhi(i))
  cov.values = sapply(1:n.max, cov.form)
  write.table(cov.values,file = paste("C2-covU(SPhi)",n.max,".txt",
                      sep = ""),row.names = F,col.names = F)
  return(cov.values)
}
```

We have computed these covariances and correlations up to $n = 1000$ with the following commands:

```
exp.values.cov = compute.EUcov(1000)
cov.values = compute.cov(exp.values.cov,1000)
```

For $n = 3, ..., 20$ the results have been:

```
exp.values.cov[3:20]
```

```
##  [1]     5.0000    32.0000   112.0000   291.0476   630.9091  1209.5478
##  [7]  2121.4881  3478.1045  5407.8581  8056.4954 11587.2180 16180.8299
## [13] 22035.8667 29368.7106 38413.6931 49423.1877 62667.6942 78435.9159
```

```
cov.values[3:20]
```

```
##  [1]    0.000000    0.320000    1.918367    6.580499   17.044077
##  [6]   37.089909   71.611701  126.671836  209.547294  328.768314
## [11]  494.151534  716.828809 1009.272571 1385.318372 1860.185104
## [16] 2450.493255 3174.281512 4051.021944
```

The rest of the values are available in the files "C2-EU(SxPhi)1000.txt" and "C2-covU(SPhi)1000.txt".

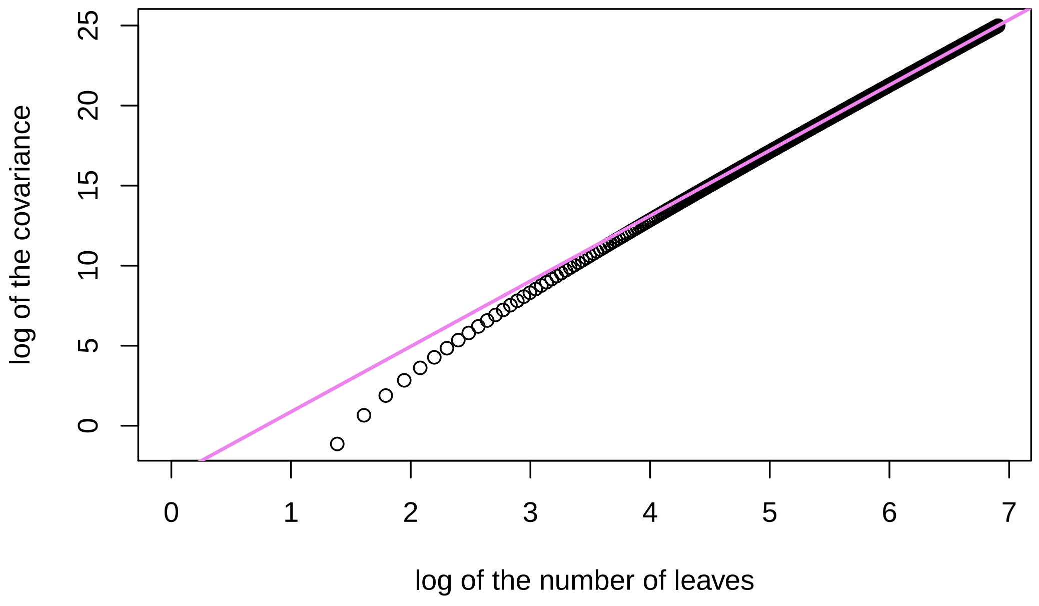
We have estimated the main order in the expansion of $Cov_U(S_n, \Phi_n)$ as a function of n, by performing the minimum squares linear regression of $\ln(Cov_U(S_n, \Phi_n))$ as a function of $\ln(n)$ for $n = 900, \ldots, 1000$,

```
summary(lm(log(cov.values[900:1000])~log(900:1000)))
```

```
##
## Call:
## lm(formula = log(cov.values[900:1000]) ~ log(900:1000))
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -3.553e-05 -1.207e-05  4.439e-06  1.430e-05  1.783e-05
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -3.1333995  0.0003579   -8756   <2e-16 ***
## log(900:1000)  4.0709153  0.0000522   77993   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.611e-05 on 99 degrees of freedom
## Multiple R-squared:      1,  Adjusted R-squared:      1
## F-statistic: 6.083e+09 on 1 and 99 DF,  p-value: < 2.2e-16
```

The code below produces Fig. 2.13, which displays $\ln(Cov_U(S_n, \Phi_n))$ as a function of $\ln(n)$, together with the corresponding regression line.

```
plot(log(1:1000),log(cov.values),
              xlab="log of the number of leaves",
               ylab="log of the covariance")
reg.cov=lm(log(cov.values[500:1000])~log(500:1000))
abline(reg.cov,col="violet",lwd=2)
```



Computing the covariance of S and Φ from the values of the indices

To double-check the recurrence, we have computed the values of $Cov_U(S_n\Phi_n)$, for $n = 3, \dots, 8$, from the Sackin and cophenetic indices of all trees in the corresponding $\mathcal{BT}_n$ (see Section A.4.3 and Section A.4.4):

```
covariancesU = function(n){
  len = length(all.sackin.index[[n]])
  value = cov(all.sackin.index[[n]],all.cophen.index[[n]]*(len-1)/len)
  return(value)
}
real.cov.values = sapply(3:7,covariancesU)
real.cov.values
```

```
## [1]  0.000000  0.320000  1.918367  6.580499 17.044077 37.089909
```

The results agree again with the figures given by our recurrence.

Computing the correlation of S and Φ from the values of the indices

Since we know how to compute recurrently $Cov_U(S_n, \Phi_n)$, $\sigma_U^2(S_n)$ and $\sigma_U^2(\Phi_n)$, we can compute Pearson's correlation ρ of S_n and Φ_n under the uniform model for any desired $n \geqslant 4$, by means of the identity

$$\rho_U(S_n, \Phi_n) = \frac{Cov_U(S_n, \Phi_n)}{\sigma_U(S_n) \cdot \sigma_U(\Phi_n)}.$$

Therefore

```r
pearson.cor = function(n.max){
  return(cov.values[4:n.max]/sqrt(var.values.S[4:n.max]*
                                     var.values.Phi[4:n.max]))
}
```

For $n = 4, ..., 20$ the results are:

```r
pearson.cor(20)
```

```
##  [1] 1.0000000 0.9968461 0.9944951 0.9926443 0.9911325 0.9898641 0.9887783
##  [8] 0.9878336 0.9870012 0.9862597 0.9855934 0.9849901 0.9844400 0.9839358
## [15] 0.9834711 0.9830410 0.9826413
```

A.4.6 Computation of the estimated probability of a tie

We have estimated the probability that a pair of trees $T_1, T_2 \in \mathcal{BT}_n$ have $I(T_1) = I(T_2)$, for $I = C, S, \Phi$. To do that, for every $n = 3, \ldots, 50$ we have chosen uniformly a set of N random pairs of trees in $\mathcal{BT}_n$ (for $n = 3, \ldots, 7$, we took $N = |\mathcal{BT}_n|$ and, for $n \geqslant 8$, we took $N = 3000$), and computed, for $I = C, S, \Phi$,

$$\widehat{p}_n(I) = \frac{\text{number of pairs } (T_1, T_2) \text{ with } n \text{ leaves such that } I(T_1) = I(T_2)}{N}.$$

The balance indices have been computed with the function `balance.indices` from the package *CollessLike* (see Section A.6.1), with the parameter `binary.Colless` set to `TRUE`. The following functions compute these probabilities $\widehat{p}_n$:

```r
are.tie = function(xx,yy) return(xx==yy)

exact.ties = function(){
  trees = list()
  all.indices = list()
  num.ties = c(0,0,0)
  prob.ties = list()
  for(n in 3:7){
    trees[[n]] = read.tree(file = paste("bintrees-n",n,".txt",sep=""))
    total.trees = length(trees[[n]])
    total.pairs = total.trees*(total.trees-1)/2
    all.indices[[n]] = matrix(sapply(trees[[n]],
                          balance.indices2),ncol=3,byrow=T)
    num.ties[1]=sum(outer(all.indices[[n]][,1],
                         all.indices[[n]][,1],are.tie))
    num.ties[2]=sum(outer(all.indices[[n]][,2],
                            all.indices[[n]][,2],are.tie))
    num.ties[3]=sum(outer(all.indices[[n]][,3],
                            all.indices[[n]][,3],are.tie))
    num.ties = (num.ties-total.trees)/2
    prob.ties[[n]] = num.ties/total.pairs
    print(paste("Ties for n =",n," : ",
                paste(c("p_C=","p_S=","p_Phi"),
                round(prob.ties[[n]],4),collapse=", "),sep=""))
  }
  return(prob.ties)
}

sim.ties.n = function(n,num.pairs.sim=3000){
  num.ties = c(0,0,0)
  for(i in 1:num.pairs.sim){
    t1 = rtree(n,rooted=TRUE)
    continue = TRUE
    while(continue){
      t2 = rtree(n,rooted=TRUE)
      continue = all.equal(t1,t2,use.length=FALSE,use.tip.label=FALSE)
    }
    t1.indices = balance.indices2(t1)
    t2.indices = balance.indices2(t2)
    num.ties = num.ties + (t1.indices==t2.indices)
  }
  print(paste("n =",n))
  print(paste("Ties :",num.ties))
  prob.ties = num.ties/num.pairs.sim
  print(paste("Prob :",round(prob.ties,4)))
  return(prob.ties)
}
```

Now, with the following commands we compute the probabilities for each $n = 3, \ldots, 50$

```
ties.1 = exact.ties()
ties.2 = lapply(8:50, sim.ties.n,num.pairs.sim=3000)
ties = matrix(c(unlist(ties.1),unlist(ties.2)),ncol=3,byrow=TRUE)
colnames(ties)=c("Colless","Sackin","Cophenetic")
rownames(ties)=3:50
```

For $n = 4, ..., 20$ the results are:

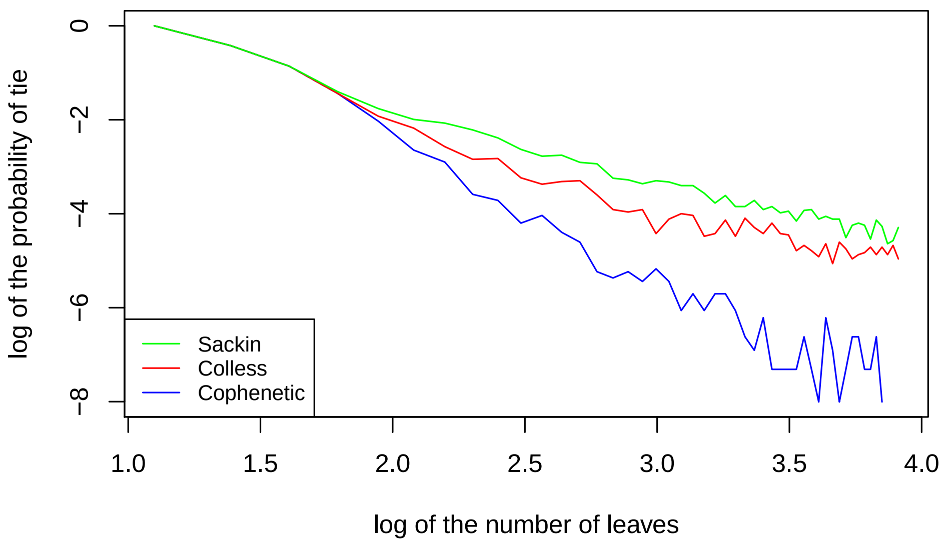
```
ties[1:18,]
```

	Colles	Sackin	Cophenetic
3	1.0000000	1.0000000	1.0000000
4	0.6571429	0.6571429	0.6571429
5	0.4230769	0.4230769	0.4230769
6	0.2372881	0.2463680	0.2372881

	Colles	Sackin	Cophenetic
7	0.1459233	0.1716375	0.1312296
8	0.1133333	0.1363333	0.0710000
9	0.0763333	0.1260000	0.0550000
10	0.0583333	0.1090000	0.0276667
11	0.0593333	0.0920000	0.0243333
12	0.0393333	0.0720000	0.0150000
13	0.0343333	0.0623333	0.0176667
14	0.0363333	0.0636667	0.0123333
15	0.0370000	0.0546667	0.0100000
16	0.0273333	0.0530000	0.0053333
17	0.0200000	0.0390000	0.0046667
18	0.0190000	0.0376667	0.0053333
19	0.0200000	0.0346667	0.0043333
20	0.0120000	0.0370000	0.0056667

The whole table is available at "C2-table-ties.txt". Fig. 2.14, which summarizes the results, has been produced with the following commands:

```
plot(log(3:50),log(ties[,3]),type="l",
     xlab="log of the number of leaves",
     ylab="log of the probability of tie", col="blue")
lines(log(3:50),log(ties[,1]),col="red")
lines(log(3:50),log(ties[,2]),col="green")
legend("bottomleft", legend=c("Sackin","Colless","Cophenetic"),
       col=c("green","red", "blue"),lty=1, cex=0.8)
```



A.4.7 Testing Φ_n on TreeBASE

In this subsection we explain how we have performed the test reported in Section 2.8.2. We have loaded the data table containing the Newick representations of all trees in TreeBASE, which we had previously downloaded using the function `search_treebase()` of the R package *treebase* and saved in the *List of Trees* folder of the PhD Thesis GitHub repository as a text file and as an R object. So, we have two ways to import these data:

```r
# Option 1
tb.ape = read.tree(file = "./tb-newicks.txt")
# Option 2
load("./treeBASE-database.RData")
```

We have considered only those numbers n of leaves for which the TreeBASE contains at least 20 binary phylogenetic trees with n leaves, and for each such n we have computed the mean of the total cophenetic indices of the corresponding binary trees:

```r
bin.tb.ape=tb.ape[sapply(tb.ape,is.rooted)]
bin.tb.ape=bin.tb.ape[sapply(bin.tb.ape,is.binary)]
bin.tb.n = sapply(bin.tb.ape,Ntip)
leaves=as.numeric(names(which(table(bin.tb.n)>20)))
bin.tb.mean = c()
indices.tb = list()
for(k in leaves){
  trees = bin.tb.ape[bin.tb.n==k]
  indices.tb[[k]] = sapply(trees, cophen.index)
  value = mean( indices.tb[[k]] )
  bin.tb.mean = rbind(bin.tb.mean,c(k,value))
}
```

The results of these computations are available in “C2-table-tb-means.txt”.

The following code computes $E_Y(\Phi_n)$ and $E_U(\Phi_n)$ for $n = 3, \ldots, 140$, using functions `EYPhi` and `EUPhi` from Section A.4.1 and Section A.4.2, respectively:

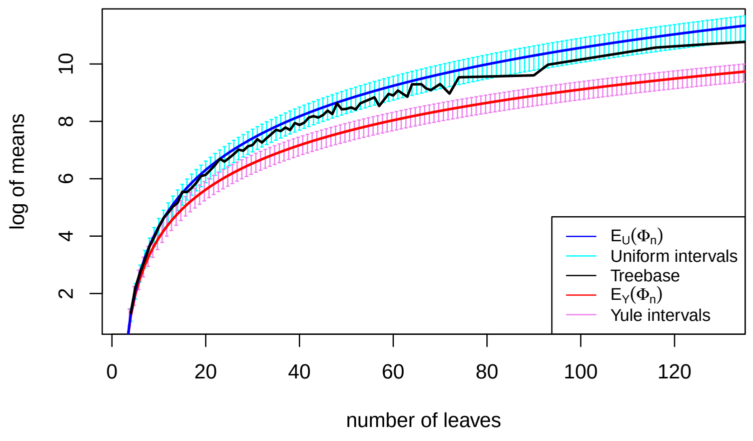
```r
range.plot = 3:140
eyphi.values = sapply(range.plot, EYPhi)
euphi.values = sapply(range.plot, EUPhi)
```

Using the computations of the variance of Φ_n under the uniform model (from Section A.4.3) and the exact formula for $\sigma_Y^2(\Phi_n)$ (Formula 2.2) we can obtain the reference intervals for Φ_n that will be drawn in the figure..

```r
harmonic2 = function(n){return(sum(1/((1:n)^2)))}
varYPhi = function(n){
  return((n^4-10*n^3+131*n^2-2*n)/12-4*n^2*harmonic2(n)-
            6*n*harmonic(n))
}
varYPhi.values = sapply(range.plot,varYPhi)
intY = cbind(range.plot,log(eyphi.values-sqrt(varYPhi.values)),
               log(eyphi.values+1*sqrt(varYPhi.values)))
intU = cbind(range.plot,log(euphi.values-
            sqrt(var.values.Phi[range.plot])),
            log(euphi.values+1*sqrt(var.values.Phi[range.plot])))

draw.intervals =
  function(range.plot,int.yule,int.uniform,delta=0){
  epsilon = 0.3
  for(i in range.plot){
    lines(c(i ,i ),int.uniform[i-2,2:3],col="cyan")
    lines(c(i-epsilon,i+epsilon),rep(int.uniform[i-2,2],2),
            col="cyan")
    lines(c(i-epsilon,i+epsilon),rep(int.uniform[i-2,3],2),
            col="cyan")
    lines(c(i ,i )-delta,int.yule[i-2,2:3],col="violet")
    lines(c(i-epsilon,i+epsilon)-delta,rep(int.yule[i-2,2],2),
            col="violet")
    lines(c(i-epsilon,i+epsilon)-delta,rep(int.yule[i-2,3],2),
            col="violet")
  }
}
```

Finally, the following code produces Fig. 2.15:

```
plot(NULL,NULL,col="blue",xlab="number of leaves",
      ylab="log of means",xlim=c(3,130),ylim=c(1,11.5),
     type="l",lwd=2)
draw.intervals(range.plot,intY,intU,delta=0.3)
lines(range.plot,log(eyphi.values),col="red",lwd=2)
lines(range.plot,log(euphi.values),col="blue",lwd=2)
lines(bin.tb.mean[,1],log(bin.tb.mean[,2]),type="l",lwd=2)
legend("bottomright", legend=c(expression(E[U]*(Phi[n])),
        "Uniform intervals","Treebase",expression(E[Y]*(Phi[n])),
        "Yule intervals"),col=c( "blue","cyan","black","red",
        "violet"),lty=1,cex=0.8)
```



A.5 Scripts from Chapter 3

A.5.1 Computation of $E(D_n^2)$

The formulas in Theorem 3.31 and 3.38, ccorresponding to the expected value of D_n^2 under the Yule and uniform models, respectively, can be computed with the following functions:

```
harmonic=function(n){return(sum(1/(1:n)))}
EYD2n = function(n){
  return((2*n/(n-1))*(3*n^2-10*n-1+8*(n+1)*harmonic(n)-
                            4*(n+1)*harmonic(n)^2))
}

EUD2n = function(n){
  return((4*n^3+18*n^2-10*n)/3+as.numeric(-as.bigq((n*(n+3))/2)*
           (big.double.factorial(2*n-2)/
               big.double.factorial(2*n-3))
           -as.bigq((n*(n+7))/4)*((big.double.factorial(2*n-2)/
               big.double.factorial(2*n-3))^2)))
}
```

For $n = 3, ..., 20$ the results are:

```
# Yule model
sapply(3:20, EYD2n)
```

```
##  [1]    2.666667    9.407407   21.183333   38.712000   62.556190
##  [6]   93.172128  130.938761  176.176855  229.162086  290.134368
## [11]  359.304706  436.860362  522.968823  617.780914  721.433274
## [16]  834.050354  955.746046 1086.625029
```

```
# uniform model
sapply(3:20, EUD2n)
```

```
##  [1]    2.666667   10.560000   26.236735   52.302343   91.408632
##  [6]  146.247151  219.543237  314.051159  432.550230  577.841679
## [11]  752.746096  960.101325 1202.760711 1483.591615 1805.474154
## [16] 2171.300112 2583.971999 3046.402233
```

To double-check the formulas, we have computed the values of $d_{\varphi,2}(T,T')^2$, for $n = 3, \ldots, 7$, from the cophenetic distance between all pairs of trees in the correpondinq $\mathcal{BT}_n$.

The cophenetic vectors of the phylogenetic trees have been computed with the function `cophen.vect` in the R package *CollessLike*. In the Yule case, we have used the function yule.prob explained in Section A.4.1 to compute the probabilities. Finally, the expected values and the variances of the square of the cophenetic distance for each n have been computed in the usual way:

```
real.exp.var = function(n.max=7){
  means = matrix(0,ncol=2,nrow=8)
  colnames(means) = c("uniform","Yule")
  vars = matrix(0,ncol=2,nrow=8)
  colnames(vars) = c("uniform","Yule")
  for(n in 3:n.max){
    trees = read.tree(file = paste("bintrees-n",n,".txt",sep=""))
    total.trees = length(trees)
    probs=sapply(trees, yule.prob)
    pairs.probs = c()
    all.vectors = lapply(trees, cophen.vect)
    values = c()
    for(i in 1:(total.trees)){
      for(j in (1):total.trees){
        values = c(values,sum((all.vectors[[i]]-
                                   all.vectors[[j]])^2))
        pairs.probs = c(pairs.probs,probs[i]*probs[j])
      }
    }
    means[n,1]=mean(values)
    means[n,2]=sum(pairs.probs*values)
    vars[n,1]=mean(values^2)-means[n,1]^2
    vars[n,2]=sum(pairs.probs*values^2)-means[n,2]^2
    print(paste("n =",n))
    print(means[n,])
    print(vars[n,])
  }
  results = cbind(3:7,means[3:7,2],vars[3:7,2],means[3:7,1],
                  vars[3:7,1])
  colnames(results) = c("n","EY(D2n)","varY(D2n)","EU(D2n)",
                        "varU(D2n)")
  return(results)
}
results=real.exp.var()
```

We have obtained the following results. They agree with the

gures given by our formulas.

n	3	4	5	6	7
$E_Y(D_n^2)$	2.66667	9.40741	21.18333	38.71200	62.55619
$E_U(D_n^2)$	2.66667	10.56000	26.23673	52.30234	91.40863

In the previous chunk of code, we have also computed the exact values for the variance:

n	3	4	5	6	7
$\sigma_Y^2(D_n^2)$	3.55556	29.13032	117.63306	339.28881	797.15834
$\sigma_U^2(D_n^2)$	3.55556	34.08640	159.50314	539.50829	1502.72330

A.5.2 Computation of $\sigma^2(D_n^2)$

In order to estimate the asymptotic order of $E(D_n^4)$ and $\sigma^2(D_n^2)$, both for the Yule and the uniform models, and for every $n = 3, \ldots, 100$, we have randomly generated $N = 10000$ pairs of binary trees $(T, T') \in \mathcal{T}_n \times \mathcal{T}_n$ using the R package *apTreeshape*, and converted them into *phylo* objects for the R package *ape*:

```
require(apTreeshape)
generate.trees = function(n,model,repetitions=10000){
  if(model=="yule")trees=rtreeshape(repetitions*2,n,model="yule")
  if(model=="uniform")trees = rtreeshape(repe*2,n,model="pda")
  trees = lapply(trees,as.phylo)
  return(trees)
}
```

We have computed the value of $d_{\varphi,2}(T,T')^2$ and $d_{\varphi,2}(T,T')^4$ for each such pair (T, T') with the following function,

```
computate.values.pairs = function(n,model,repetitions=10000){
  euc.dist2 = function(pair){
    m = length(pair)/2
    value = sum((pair[1:m] - pair[(m+1):(2*m)])^2)
    return(value)
  }
  trees=generate.trees(n,model,repetitions)
  vectors = lapply(trees, cophen.vector)
  vectors = matrix(unlist(vectors),byrow=T,nrow=repetitions)
  result = apply(vectors,1,euc.dist2)
  result = c(mean(result),mean(result^2))
  return(c(result,result[2]-result[1]^2))
}
```

We have computed the arithmetic means $\overline{D_n^2}$ and $\overline{D_n^4}$ of these N values, and, finally, the variance of the values $d_{\varphi,2}(T,T')^2$ using the identity

$$\widehat{\sigma^2}(D_n^2) = \overline{D_n^4} - \overline{D_n^2}^2.$$

This value is an estimation of $\sigma^2(D_n^2)$ under the corresponding model. Next commands show how we can compute these variances:

```
varD2n = c()
for(k in 3:100){
  values.yule = computate.values.pairs(k,"yule",10000)
  values.uniform = computate.values.pairs(k,"uniform",10000)
  varD2n = rbind(varD2n,c(k,values.yule[2:3],values.uniform[2:3]))
}
colnames(varD2n) = c("n","Yule_EDn4","Yule_varDn2",
                       "uniform_EDn4","uniform_varDn2")
```

Since these computations take a long time to finish, we have parallelized them with the R package *parallel*. For $n = 3, ..., 20$ the results have been:

```
varD2n[1:13,]
```

n	Yule_EDn4	Yule_varDn2	uniform_EDn4	uniform_varDn2
3	10.6032	3.5765	10.6816	3.5506
4	116.7946	29.1438	143.3660	33.8602
5	559.7510	115.1738	852.0843	152.8521
6	1837.4050	334.2533	3299.1028	531.5538
7	4648.0487	761.6796	9872.4776	1498.6903
8	10330.2279	1621.1155	25175.2292	3679.6130
9	19785.9321	2975.1427	57131.8612	8153.9221
10	35948.3028	4931.6335	115022.2093	16236.0131
11	60700.8108	8278.0529	216900.0930	31384.2511
12	95272.0040	12040.3310	391444.9727	56683.5990
13	148901.4119	18688.7925	669652.7456	98755.8404
14	218716.4557	27015.5924	1088410.0273	161050.1757
15	312615.8741	37466.0192	1698106.1014	254385.1408

The rest of the values are available in the file "C4-table-expDn4-varDn2.txt".

We have computed the slope α of the regression line of $\log(\overline{\sigma^2}(D_n^2))$ as a function of $\log(n)$ using the values for $n = 50, \ldots, 100$ with the following commands.

```
#var_Y(D2n)
reg.yule = lm(log(varD2n[48:98,3])~log(50:100))
summary(reg.yule)
```

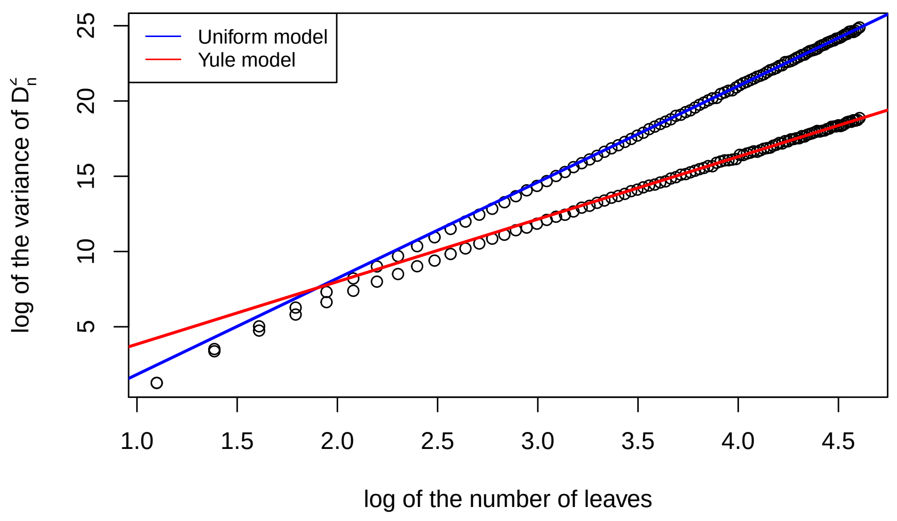
```
##
## Call:
## lm(formula = log(varD2n[48:98, 3]) ~ log(50:100))
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.113573 -0.034866  0.004031  0.033017  0.092599
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.30850    0.13540  -2.278   0.0271 *
## log(50:100)  4.15220    0.03147 131.931   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.04537 on 49 degrees of freedom
## Multiple R-squared:  0.9972, Adjusted R-squared:  0.9971
## F-statistic: 1.741e+04 on 1 and 49 DF,  p-value: < 2.2e-16
```

```
#var_U(D2n)
reg.uniform = lm(log(varD2n[48:98,5])~log(50:100))
summary(reg.uniform)
```

```
##
## Call:
## lm(formula = log(varD2n[48:98, 5]) ~ log(50:100))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.09680 -0.02160  0.00325  0.02406  0.09204
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -4.55620    0.10217  -44.59   <2e-16 ***
## log(50:100)  6.38830    0.02375  269.01   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.03423 on 49 degrees of freedom
## Multiple R-squared:  0.9993, Adjusted R-squared:  0.9993
## F-statistic: 7.237e+04 on 1 and 49 DF,  p-value: < 2.2e-16
```

Finally, the following commands produce Fig. 3.26:

```
plot(log(3:100),log(varD2n[,5]),
          xlab="log of the number of leaves",
          ylab=expression("log of the variance of "*D[n]^2))
abline(reg.uniform,col="blue",lwd=2)
points(log(3:100),log(varD2n[,3]))
abline(reg.yule,col="red",lwd=2)
legend("topleft",legend=c("Uniform model","Yule model"),
          col=c("blue","red"),lty=1,cex=0.8)
```



A.6 Scripts from Chapter 4

A.6.1 The R package *CollessLike*

We have written the R package *CollessLike*, available on the CRAN and on the GitHub, that computes the Colless-like indices and their normalized version, as well as several other balance indices, and simulates the distribution of these indices under the α-γ-model. We describe its contents in this subsection. The following function computes the Sackin index of a tree. The value can be normalized with `norm=TRUE`.

```
sackin.index <-
  function(tree,norm=FALSE){
    if(class(tree)=="character")
      tree=read.tree(text = tree)
    if (class(tree)=="phylo")
      tree=graph.edgelist(tree$edge, directed=TRUE)
    if(class(tree)!="igraph")
      stop("Not an igraph object. Please introduce a newick
           string, an ape tree or an igraph tree.")
    root.node = which(degree(tree,mode="in")==0)
    deg.out = degree(tree,mode="out")
    if(deg.out[root.node]==1){ #exists a root-edge
      tree = delete.vertices(tree,root.node)
      deg.out = degree(tree,mode="out")
      root.node = which(degree(tree,mode="in")==0)
    }
    leaves = which(deg.out==0)
    root.list = get.shortest.paths(tree,root.node)$vpath
    depths = unlist(lapply(root.list,function(xx){length(xx)-1}))
    SACKIN=sum(depths[leaves])
    if(norm){
      N = length(leaves)
      max.s = N*(N-1)/2 + N-1
      SACKIN = (SACKIN-N)/(max.s-N)
    }
    return(SACKIN)
  }
```

The following function computes the total cophenetic index of a tree. The value can be normalized with `norm=TRUE`.

```
cophen.index <-
  function(tree,norm=FALSE){
    if(class(tree)=="character")
      tree=read.tree(text = tree)
    if (class(tree)=="phylo")
      tree=graph.edgelist(tree$edge, directed=TRUE)
    if(class(tree)!="igraph")
      stop("Not an igraph object. Please introduce a newick
           string, an ape tree or an igraph tree.")
    root.node = which(degree(tree,mode="in")==0)
    deg.out = degree(tree,mode="out")
    if(deg.out[root.node]==1){ #exists a root-edge
      tree = delete.vertices(tree,root.node)
      deg.out = degree(tree,mode="out")
      root.node = which(degree(tree,mode="in")==0)
    }
    leaves = which(deg.out==0)
    root.list = get.shortest.paths(tree,root.node)$vpath
    # COPHENETIC #
    N = length(leaves)
    COPHEN = 0
    for(i in 1:(N-1))
      for(j in (i+1):N){
        aux  = length(intersect(root.list[[leaves[i]]],
                                root.list[[leaves[j]]]))-1
        COPHEN = COPHEN + aux
      }
    if(norm){
      max.c = N*(N-1)*(N-2)/6
      COPHEN = COPHEN/max.c
    }
    return(COPHEN)
  }
```

The following function computes the Colless-like index of a tree. The value can be normalized with `norm=TRUE`. By default, the f-size is $f(n) = \ln(n+e)$, if `f.size="exp"` then $f(n) = e^n$. It can also be a user-defined function but in this case, the index cannot be normalized. On the other hand, the default value of the dissimilarity is MDM (mean deviation from the median). Other values can be set as `diss="sd"` for the sample standard deviation or `diss="var"` for the sample variance. It can also be a user-defined function but, again, in this case the value cannot be normalized.

```
colless.like.index <-
  function(tree,f.size="ln",diss="MDM",norm=FALSE){
    if(class(tree)=="character")
      tree=read.tree(text = tree)
    if (class(tree)=="phylo")
      tree=graph.edgelist(tree$edge, directed=TRUE)
    if(class(tree)!="igraph")
      stop("Not an igraph object. Please introduce a newick
           string, an ape tree or an igraph tree.")
    root.node = which(degree(tree,mode="in")==0)
    deg.out = degree(tree,mode="out")
    case.norm = 0
    if(class(f.size)=="character"){
      if(f.size=="ln"){
        f.size = function(nn)  return(log(nn+exp(1)))
        case.norm = 1
      }
      else if((f.size=="exp")||(f.size=="e")){
        f.size = function(nn)  return( exp(nn) )
        case.norm = 4
      }
      else stop("The f-size introduced is not correct.")
    }
    if(class(diss)=="character"){
      if((diss=="MDM")||(diss=="mdm")){
        diss = function(xx)
                  return(sum(abs(xx-median(xx)))/length(xx))
        case.norm = case.norm*1
      }
      else if(diss=="var"){
        diss = function(xx) return(sum((xx-mean(xx))^2)/
                                     (length(xx)-1))
        case.norm = case.norm*2
      }
      else if(diss=="sd"){
        diss = function(xx) return(sqrt(sum((xx-mean(xx))^2)/
                                           (length(xx)-1)))
        case.norm = case.norm*3
      }
      else stop("The dissimilarity introduced is not correct.")
    }
    int.nodes = (1:length(V(tree)))[deg.out>0]
    decendents = neighborhood(tree,1,int.nodes,mode = "out")
    fun.nodes.deltas = function(nodes){
      aux = neighborhood(tree,length(deg.out)-1,nodes,
                          mode = "out")[[1]]
      return(sum(f.size(deg.out[aux])))
    }
    fun.children = function(children){
      children = children[-1]
      result =  unlist(lapply(children,fun.nodes.deltas))
      return(result)
    }
    deltas = lapply(decendents,fun.children)
    Vdiss = lapply(deltas, diss)
    COLLESS = sum(unlist(Vdiss))
    if(norm){
      if(case.norm==0) warning("Indices can not be normalized")
      else{
        N = length(which(deg.out==0))
        # ln MDM
        if(case.norm==1)
          max.cl = (f.size(0) + f.size(2))*(N-1)*(N-2)/4
        # ln var
        if(case.norm==2)
          max.cl = (f.size(0) + f.size(2) )^2*(N-1)*(N-2)*
```

```
                     (2*N-3)/12
        # ln sd
        if(case.norm==3)
          max.cl = (f.size(0) + f.size(2))*(N-1)*(N-2)/
                     (2*sqrt(2))
        # e^n var
        if(case.norm==8)  max.cl = (f.size(N-1)+N-2)^2/2
        if(N==4){
          # e^n MDM
          if(case.norm==4)  max.cl = (f.size(2)+1)*3/2
          # e^n sd
          if(case.norm==12) max.cl =(f.size(2)+1)*3/sqrt(2)
        }
        else{
          # e^n MDM
          if(case.norm==4)  max.cl = (f.size(N-1)+N-2)/2
          # e^n sd
          if(case.norm==12) max.cl = (f.size(N-1)+N-2)/sqrt(2)
        }
        COLLESS  = COLLESS /max.cl
      }
    }
    return(COLLESS)
  }
```

The next function assembles the three previous functions. So, it computes the three indices and returns a single array with the three values. The results can be normalized. If `binary.Colless=TRUE`, it computes the classical Colless index for binary trees (previously checking that the input tree is binary).

```
balance.indices <- function(tree,norm=FALSE,binary.Colless=FALSE){
    if(class(tree)=="character")
      tree=read.tree(text = tree)
    if (class(tree)=="phylo")
      tree=graph.edgelist(tree$edge, directed=TRUE)
    if(class(tree)!="igraph")
      stop("Not an igraph object. Please introduce a newick
           string, an ape tree or an igraph tree.")
    root.node = which(degree(tree,mode="in")==0)
    deg.out = degree(tree,mode="out")
    # COLLESS.MDM.LN
    D.MDM = function(xx)
        return(sum(abs(xx-median(xx)))/length(xx))
    f.ln   = function(n)  return(log(n+exp(1)))
    int.nodes = (1:length(V(tree)))[deg.out>0]
    decendents = neighborhood(tree,1,int.nodes,mode = "out")
    fun.nodes.deltas = function(nodes){
      aux = neighborhood(tree,length(deg.out)-1,nodes,
                          mode = "out")[[1]]
      return(sum(f.ln(deg.out[aux])))
```

```
  }
  fun.children = function(children){
    children = children[-1]
    result =  unlist(lapply(children,fun.nodes.deltas))
    return(result)
  }
  deltas = lapply(decendents,fun.children)
  Vdis = lapply(deltas, D.MDM)
  COLLESS = sum(unlist(Vdis))
  if(deg.out[root.node]==1){ #exists root-edge
    tree = delete.vertices(tree,root.node)
    deg.out = degree(tree,mode="out")
    root.node = which(degree(tree,mode="in")==0)
  }
  leaves = which(deg.out==0)
  root.list = get.shortest.paths(tree,root.node)$vpath
  # SACKIN #
  depths = unlist(lapply(root.list,
                         function(xx){length(xx)-1}))
  SACKIN=sum(depths[leaves])
  # COPHENETIC #
  N = length(leaves)
  COPHEN = 0
  for(i in 1:(N-1))
    for(j in (i+1):N){
      aux  = length(intersect(root.list[[leaves[i]]],
                              root.list[[leaves[j]]]))-1
      COPHEN = COPHEN + aux
    }
  result = c("Colles-Like"=COLLESS,"Sackin"=SACKIN,
             "Cophenetic"=COPHEN)
  if(binary.Colless){
    if(sum(!(deg.out %in% c("0","2")))==0)
      result[1] = result[1]/((log(0+exp(1))+log(2+exp(1)))/2)
    else warning("The tree introduced is not binary,
                 Colless-like index for multifurcated trees is
                 computed.")
  }
  else{
    if(norm){
      max.cl = ( log(0+exp(1)) + log(2+exp(1)) )*(N-1)*(N-2)/4
      max.s = N*(N-1)/2 + N-1
      max.c = N*(N-1)*(N-2)/6
      result[1] = result[1]/max.cl
      result[2] = (result[2]-N)/(max.s-N)
      result[3] = result[3]/max.c
    }
  }
  return(result)
  }
```

The following function computes the total cophenetic vector of a tree.

```
cophen.vector <-  function(tree,set.of.labels=NULL){
    if(class(tree)=="character")
      tree=read.tree(text = tree)
    if (class(tree)=="phylo"){
      if(is.null(set.of.labels))
        set.of.labels = tree$tip.label
      tree=graph.edgelist(tree$edge, directed=TRUE)
    }
    if(class(tree)!="igraph")
      stop("Not an igraph object. Please introduce a newick
           string, an ape tree or an igraph tree.")
    if(is.null(set.of.labels))
      stop("Please insert the set of labels or a phylo object")
    root.node = which(degree(tree,mode="in")==0)
    deg.out = degree(tree,mode="out")
    if(deg.out[root.node]==1){ #exists a root-edge
      tree = delete.vertices(tree,root.node)
      deg.out = degree(tree,mode="out")
      root.node = which(degree(tree,mode="in")==0)
    }
    leaves = which(deg.out==0)
    if(length(set.of.labels)!=length(leaves))
      stop("Please insert the correct set of labels or a phylo
           object")
    root.list = get.shortest.paths(tree,root.node)$vpath
    # COPHENETIC #
    N = length(leaves)
    COPHEN = c()
    ordered.leaves=order(set.of.labels)
    for(i in 1:(N-1)){
      leaf.i = ordered.leaves[i]
      COPHEN = c(COPHEN,length(root.list[[leaf.i]])-1)
      for(j in (i+1):N){
        leaf.j = ordered.leaves[j]
        aux  = length(intersect(root.list[[leaves[leaf.i]]],
                                root.list[[leaves[leaf.j]]]))-1
        COPHEN = c(COPHEN,aux)
      }
    }
    COPHEN = c(COPHEN,length(root.list[[ordered.leaves[N]]])-1)
    return(COPHEN)
  }
```

Given `alpha`, `gamma` and the number of leaves `n`, the following function generates a random phylogenetic tree with $n \geqslant 3$ leaves with the probability distribution defined by the α-γ-model.

```r
a.g.model <-
  function(n,alpha,gamma){
    if(n<3)
      stop("n<3")
    else{
      if((alpha>1)||(alpha<0)||(gamma>1)||(gamma<0))
        stop("alpha and gamma must been between 0 and 1")
      else{
        if(alpha<gamma)
          stop("alpha < gamma")
        else{
          edge.matrix = matrix(c(1,2,2,3,2,4),byrow=T,ncol=2)
          n.nodes = 4
          n.edges = 3
          for(n.leaves in 3:n){#Add new leaf
            #Asign probabilities
            probabilities = rep(0,n.nodes+n.edges)
            degrees = rep(0,n.nodes)
            degree.table = table(edge.matrix[,1])
            degrees[as.numeric(names(degree.table))]=degree.table

            leaves = which(degrees==0)
            leaf.edge =  which(edge.matrix[,2]%in%leaves)

            probabilities[1:n.edges + n.nodes] = gamma
            probabilities[leaf.edge+n.nodes] = 1-alpha
            probabilities[which(degrees>1)]=(degrees[degrees>1]-1)*alpha-gamma
            probabilities = probabilities/(n.leaves-alpha)

            random = sample(c(1:n.nodes,1:n.edges+n.nodes),
                            1,prob=probabilities)

            if(random<=n.nodes){#a node is selected
              edge.matrix = rbind(edge.matrix,c(random,n.nodes+1))
              n.nodes = n.nodes+1
              n.edges = n.edges+1
            }
            else{#an edge is selected
              random = random - n.nodes
              edge.matrix = rbind(edge.matrix,
                              c(edge.matrix[random,1],n.nodes+1))
              edge.matrix = rbind(edge.matrix,
                              c(n.nodes+1,edge.matrix[random,2]))
              edge.matrix = rbind(edge.matrix,
                              c(n.nodes+1,n.nodes+2))
              edge.matrix = edge.matrix[-random,]
              n.nodes = n.nodes+2
              n.edges = n.edges+2
            }
          }
          tree = graph.edgelist(edge.matrix)
          deg.out = degree(tree,mode="out")
          root.node = which(degree(tree,mode="in")==0)
          if(deg.out[root.node]==1){ #Erase the root-edge
            tree = delete.vertices(tree,root.node)
          }
          return(tree)
        }
      }
    }
  }
```

The following function generates a list of trees with the probability distribution defined by an α-γ-model and then it computes their Colless-like, Sackin and total cophenetic indices.

```
indices.simulation <-
function(n,alpha=NA,gamma=NA,repetitions=1000,norm=FALSE){
    only.one=FALSE
    if(is.na(alpha)){
        parameters = expand.grid(seq(0,1,0.1),seq(0,1,0.1),n)
        parameters = parameters[which(parameters[,1]>=
                                          parameters[,2]),]
    }
    else{
      if(is.na(gamma)){
        parameters = expand.grid(alpha,seq(0,alpha,0.1),n)
        parameters = parameters[which(parameters[,1]>=
                                          parameters[,2]),]
      }
      else{
        if((alpha>1)||(alpha<0)||(gamma>1)||(gamma<0))
          stop("alpha and gamma must been between 0 and 1")
        else
          if(alpha<gamma)
            stop("alpha < gamma")
          else{
            parameters = c(alpha,gamma,n)
            only.one = TRUE
          }
      }
    }
    generator = function(idx,n,alpha,gamma){
      return(a.g.model(n,alpha,gamma))
    }
    iterate.ford = function(tab){ #tab = [alpha,gamma,n]
        if(tab[1]>=tab[2]){
            alpha = tab[1]
            gamma = tab[2]
            n = tab[3]
            print(paste("n :",n," alpha :",alpha,
                        " gamma :",gamma))
            tree.list = lapply(1:repetitions,generator,n,
                               alpha,gamma)
            result = matrix(unlist(lapply(tree.list,
                    balance.indices,norm=norm)),ncol=3,byrow=T)
            colnames(result) = c("COLLES.MDM.LN","SACKIN",
                                 "COPHENETIC")
        }
        return(result)
    }
    if(only.one){
      result = iterate.ford(parameters)
    }
    else{
      parameters2=lapply(1:(dim(parameters)[1]),
                         function(i) as.numeric(parameters[i,]))
      result = lapply(parameters2,iterate.ford)
      paste.param = function(tab){
        return(paste("a",tab[1],"g",tab[2],sep=""))
        }
      names(result) = apply(parameters,1,paste.param)
    }
    return(result)
}
```

The following function, given α,γ and a phylogenetic tree, plots the distribution of the normalized versions of the Colless-like, Sackin and total cophenetic indices under the α-γ-model on $\mathcal{T}_n$. It also computes the percentiles of the indices of the tree under this α-γ-model. Two plots are available: one represents the percentile plots of the normalized balance indices (`percentile.plot=TRUE`), and the other represents the density plots of the normalized balance indices (`percentile.plot=FALSE`). In order to compute the distribution and percentiles, this function needs a database of trees generated under the α-γ-model. Our database is available on the *GitHub*. The trees stored in our database have between 3 and 50 leaves and the values of the parameters `alpha` and `gamma` are in $\{0, 0.1, \ldots, 1\}$ such that `gamma` $\leqslant$ `alpha`. If the introduced parameters are not in the list, a new computation is done with them and a new dataset of trees is generated, and their indices are also computed. The number of trees generated can be modified by the parameter `repetitions` (see `indices.simulation` for more information). This computation may take some time, therefore you can compute the trees separately with `indices.simulation`, save their values and then call this function by setting them as the value of parameter `set.indices`.

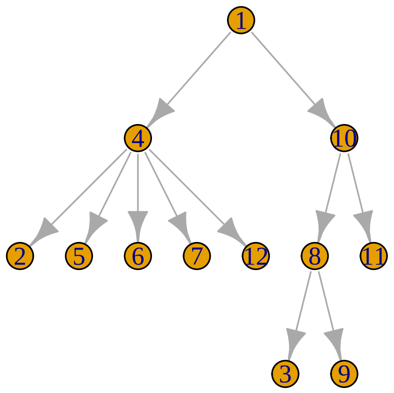
```
distribution <- function(tree,alpha=NA,gamma=NA,set.indices=NULL,
           new.simulation=FALSE,repetitions=1000,
           legend.location="topright",cex=0.75,
           percentile.plot=FALSE,db.path=getwd() ){
    ## Class of object "tree"
    if(class(tree)=="character")
      tree=read.tree(text = tree)
    if (class(tree)=="phylo")
      tree=graph.edgelist(tree$edge, directed=TRUE)
    if(class(tree)!="igraph")
      stop("Not an igraph object. Please introduce a newick
           string, an ape tree or an igraph tree.")
    n = sum(degree(tree,mode="out")==0)
    ## parameters alpha & gamma
    if(new.simulation){
      print("This process might take a long time. If you want
            to save the indices simulation, please run
            'indices.simulation' directly and then call
            'distribution' by setting the resulting table as
            the parameter 'set.indices'")
      print("Remember, our indices data base is available to
            download at: https://github.com/LuciaRotger/
            CollessLike/tree/master/CollessLikeDataBase")
      warning("New simulation required")
      indices.list=indices.simulation(n,alpha,gamma,repetitions)
      txt = bquote(paste("Parameters: ",alpha," = ",.(alpha),
                         ", ",gamma," = ",.(gamma)))
    }
    else{
      if(is.null(set.indices)){
        if(alpha<gamma){
          print("Remember, our indices data base is available
                to download at: https://github.com/LuciaRotger/
                CollessLike/tree/master/CollessLikeDataBase")
          stop("alpha < gamma")
        }
        else{
          if((alpha>1)||(alpha<0)||(gamma>1)||(gamma<0)){
            print("Remember, our indices data base is available
                  to download at:https://github.com/LuciaRotger/
                  CollessLike/tree/master/CollessLikeDataBase")
            stop("alpha and gamma must been between 0 and 1")
          }
          else{
            txt = bquote(paste("Parameters: n = ",.(n),",
                               ",alpha," = ",.(alpha),",
                               ",gamma," = ",.(gamma)))
            if(paste("n",n,sep="")%in%dir(db.path)){
              file = paste("CollessLikeDataBase_n",n,"_a",
                  alpha*100,"_g",gamma*100, "_r5000.txt",sep="")
              folder = paste(db.path,"n",n,"/",sep="")
              if(file %in% dir(folder)){
                indices.list=read.table(file=paste(folder,file,
                                          sep=""), header=TRUE)
              }
              else {
                print("Remember, our indices data base is
                      available to download at:https://github.
                      com/LuciaRotger/CollessLike/tree/master/
                      CollessLikeDataBase")
                stop(paste("The file '",file,
                    "' is not located at '",folder,"'",sep=""))
              }
            }
            else {
              print("Remember, our indices data base is
                    available to download at: https://github.
                  com/LuciaRotger/CollessLike/tree/master/
                  CollessLikeDataBase")
              stop(paste("The folder 'n",n,
                   "' is not located at '",db.path,"'",sep=""))
            }
          }
        }
      }
      else{
        print("Indices Database introduced by user")
        txt=""
        indices.list = set.indices
      }
    }
    if(max(indices.list)>1){
      ## maximum
      max.cl = ( log(0+exp(1)) + log(2+exp(1)) )*(n-1)*(n-2)/4
      max.s = n*(n-1)/2 + n-1
      max.c = n*(n-1)*(n-2)/6
      indices.list[,1] = round(indices.list[,1]/max.cl,4)
      indices.list[,2] = round((indices.list[,2]-n)/(max.s-n),4)
      indices.list[,3] = round(indices.list[,3]/max.c,4)
    }
    # densities
    d.cl= density(indices.list[,1])
    d.s = density(indices.list[,2])
    d.c = density(indices.list[,3])
    xlim = range(c(0,1))
    ylim = range(c(0,d.cl$y,d.s$y,d.c$y))

    #tree
    tree.indices = balance.indices(tree)
    tree.indices = round(c(tree.indices[1]/max.cl,
                           (tree.indices[2]-n)/(max.s-n),
                           tree.indices[3]/max.c),4)

    f.cl=approxfun(d.cl$x,d.cl$y)
    f.s=approxfun(d.s$x,d.s$y)
    f.c=approxfun(d.c$x,d.c$y)
```

```r
tree.densities = round(c(f.cl(tree.indices[1]),
                         f.s(tree.indices[2]),
                         f.c(tree.indices[3])),4)
tree.densities[is.na(tree.densities)]=0

a.cl = cumsum(d.cl$y)
a.cl=a.cl/max(a.cl)
a.s= cumsum(d.s$y)
a.s= a.s/max(a.s)
a.c= cumsum(d.c$y)
a.c= a.c/max(a.c)
#tree index plots percs
percs = c(a.cl[which(d.cl$x/max(d.cl$x)>tree.indices[1])[1]],
          + a.s[which(d.s$x/max(d.s$x)>tree.indices[2])[1]],
          + a.c[which(d.c$x/max(d.c$x)>tree.indices[3])[1]])
percs[is.na(percs)]=1
percs = round(percs,4)

print(paste("Tree with n=",n," leaves",sep=""))
print(paste("Colles-like: ",tree.indices[1],
            " (density:", tree.densities[1] ,"),
            Percentile:", percs[1] ,sep=""))
print(paste("Sackin: ",tree.indices[2],
            " (density:", tree.densities[2] ,"),
            Percentile:", percs[2] ,sep=""))
print(paste("Cophenetic: ",tree.indices[3],
            " (density:", tree.densities[3] ,"),
            Percentile:", percs[3],sep=""))

#plots
par(xpd=FALSE)

if(!percentile.plot){
  plot(-1,-1 , xlab = "", ylab="Distribution of indices",
       xlim = xlim, ylim = ylim,xaxs = 'i', yaxs='i',
       main = 'Distribution of indices',
       panel.first = grid() )

  polygon(d.cl, density = -1, col=rgb(1,0,0,0.2),
          border = "red",lwd = 1)
  polygon(d.s, density = -1, col=rgb(0,1,0,0.2),
          border = "green",lwd = 1)
  polygon(d.c, density = -1, col=rgb(0,0,1,0.2),
          border = "blue",lwd = 1)
  legend(legend.location,c("Colles-Like","Sackin",
          "Cophenetic"), fill = c(rgb(1,0,0,0.2),
          rgb(0,1,0,0.2),rgb(0,0,1,0.2),bty ='n',
          border = NA),cex=cex)
```

```
    lines(rep(tree.indices[1],2),c(0,tree.densities[1]),
            col=rgb(1,0,0),lwd =2)
    lines(rep(tree.indices[2],2),c(0,tree.densities[2]),
            col=rgb(0,1,0),lwd =2)
    lines(rep(tree.indices[3],2),c(0,tree.densities[3]),
            col=rgb(0,0,1),lwd =2)

    lines(xlim,c(0,0),lwd=2)

    points(tree.indices[1],tree.densities[1],pch=21,
            bg=rgb(1,0,0))
    points(tree.indices[2],tree.densities[2],pch=21,
            bg=rgb(0,1,0))
    points(tree.indices[3],tree.densities[3],pch=21,
            bg=rgb(0,0,1))
  }
  else{
    plot(-1,-1 , xlab = "", ylab="Percentiles",
         xlim = xlim, ylim = c(0,1),xaxs = 'i', yaxs='i',
         main = 'Percentile Plot', panel.first = grid() )

    lines(d.cl$x/max(d.cl$x),a.cl, col=rgb(1,0,0))
    lines( d.s$x/max(d.s$x), a.s, col=rgb(0,1,0))
    lines( d.c$x/max(d.c$x), a.c, col=rgb(0,0,1))

    legend("topleft",c("Colles-Like","Sackin","Cophenetic"),
           fill = c(rgb(1,0,0,0.2),rgb(0,1,0,0.2),
           rgb(0,0,1,0.2),bty ='n',border = NA),cex=cex)

    lines(rep(tree.indices[1],2),c(0,percs[1]),
            col=rgb(1,0,0),lwd =2)
    lines(rep(tree.indices[2],2),c(0,percs[2]),
            col=rgb(0,1,0),lwd =2)
    lines(rep(tree.indices[3],2),c(0,percs[3]),
            col=rgb(0,0,1),lwd =2)

    lines(xlim,c(0,0),lwd=1)

    points(tree.indices[1],percs[1],pch=21,bg=rgb(1,0,0))
    points(tree.indices[2],percs[2],pch=21,bg=rgb(0,1,0))
    points(tree.indices[3],percs[3],pch=21,bg=rgb(0,0,1))

  }
  mtext( txt , line = 0.3)
  mtext("Normalized indices",line = 2.5,side = 1)
  mtext(bquote(paste("Percentiles: ",P[C],"=",.(percs[1]),
            ", ", P[S],"=",.(percs[2]),",",P[Phi],
            "=",.(percs[3]) )),line = 4,side = 1)
  return(percs)
  }
```

For instance, we have generated the random tree depicted in Fig. 4.10 under the α-γ-model with $\alpha = 0.7$ and $\gamma = 0.4$ the following commands (using `set.seed(1000)` for reproducibility)

```
set.seed(1000)
tree=a.g.model(8,0.7,0.4)
plot(tree,layout=layout.reingold.tilford(tree,root=1))
```



We can compute the three balance indices (Colles-like, Sackin and total cophenetic) on this tree and their normalized values:

```
balance.indices(tree)
```

```
## Colles-Like      Sackin  Cophenetic
##    1.746074   18.000000   14.000000
```

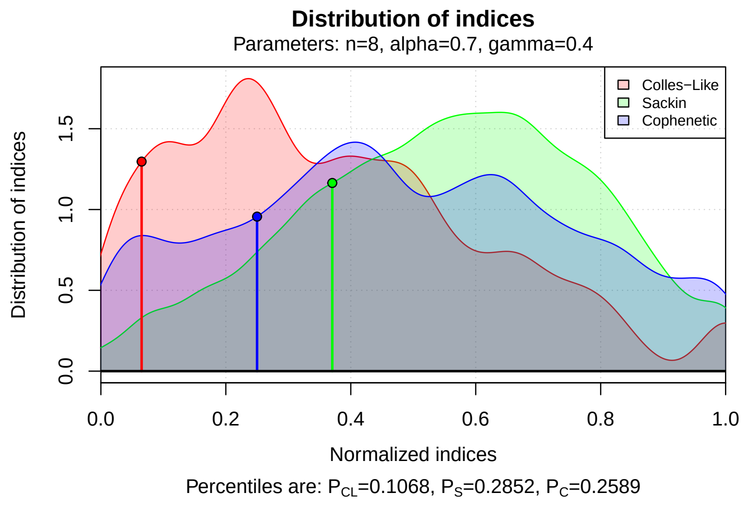
```
balance.indices(tree,norm = TRUE)
```

```
## Colles-Like      Sackin  Cophenetic
##   0.0651759   0.3703704   0.2500000
```

Then, Fig. 4.11, displaying the estimation of the density function of the three balance indices under the α-γ-model with $\alpha = 0.7$ and $\gamma = 0.4$ on $\mathcal{T}_8$, has been generated as follows:

```
database.location = "D:/Recerca/sep_2015/sep_2016/CollessLike/CollessLikeDataBase/"
distribution(tree,0.7,0.4,db.path = database.location)
```

```
## [1] "Tree with n=8 leaves"
## [1] "Colles-like: 0.0652 (density:1.2966), Percentile:0.1068"
## [1] "Sackin: 0.3704 (density:1.1642), Percentile:0.2852"
## [1] "Cophenetic: 0.25 (density:0.9561), Percentile:0.2589"
```

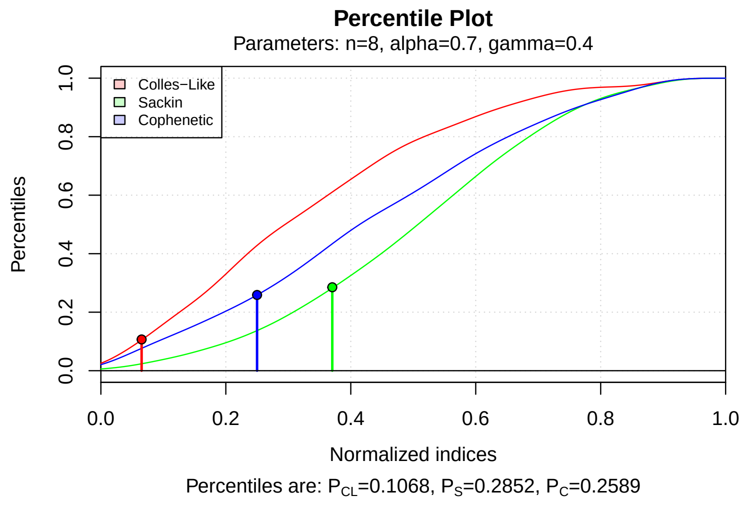


```
## [1] 0.1068 0.2852 0.2589
```

Fig. 4.12, which shows a percentile plot of the three balance indices under the α-γ-model for $\alpha = 0.7$ and $\gamma = 0.4$ on $\mathcal{T}_8$ and the estimated percentiles of the balance indices of the tree, has been produced with the following command:

```
distribution(tree,0.7,0.4,db.path = database.location,percentile.plot = TRUE)
```

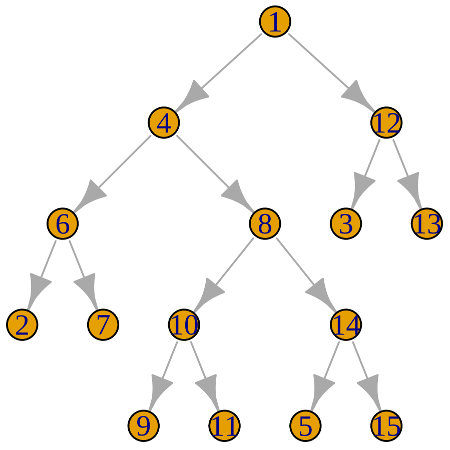
```
## [1] "Tree with n=8 leaves"
## [1] "Colles-like: 0.0652 (density:1.2966), Percentile:0.1068"
## [1] "Sackin: 0.3704 (density:1.1642), Percentile:0.2852"
## [1] "Cophenetic: 0.25 (density:0.9561), Percentile:0.2589"
```



```
## [1] 0.1068 0.2852 0.2589
```

The unlabeled tree of Fig. 4.13 has been generated (with `set.seed(1000)` using $n = 8$ and $\alpha = \gamma = 0.5$, which corresponds to the uniform model. The information on it and Fig. 4.14 have been obtained with the following code:

```
set.seed(1000)
tree.uni=a.g.model(8,0.5,0.5)
plot(tree.uni,layout=layout.reingold.tilford(tree.uni,root=1))
```



```
balance.indices(tree.uni)
```

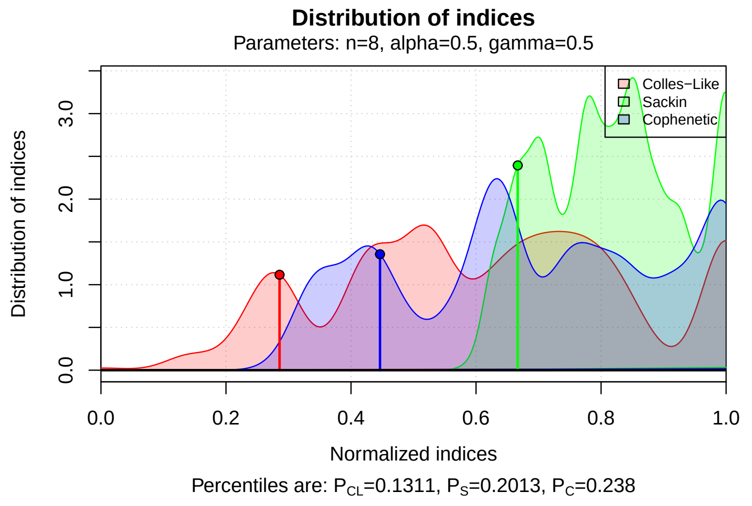
```
## Colles-Like      Sackin  Cophenetic
##    7.654334   26.000000   25.000000
```

```
balance.indices(tree.uni,norm = TRUE)
```

```
## Colles-Like      Sackin  Cophenetic
##   0.2857143   0.6666667   0.4464286
```

```
distribution(tree.uni,0.5,0.5,db.path = database.location)
```

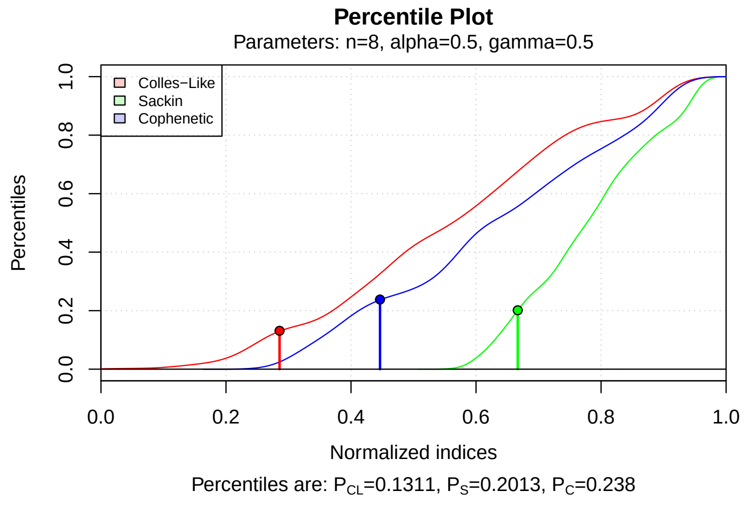
```
## [1] "Tree with n=8 leaves"
## [1] "Colles-like: 0.2857 (density:1.1156), Percentile:0.1311"
## [1] "Sackin: 0.6667 (density:2.3944), Percentile:0.2013"
## [1] "Cophenetic: 0.4464 (density:1.3557), Percentile:0.238"
```



```
## [1] 0.1311 0.2013 0.2380
```

```
distribution(tree.uni,0.5,0.5,db.path = database.location, percentile.plot = TRUE)
```

```
## [1] "Tree with n=8 leaves"
## [1] "Colles-like: 0.2857 (density:1.1156), Percentile:0.1311"
## [1] "Sackin: 0.6667 (density:2.3944), Percentile:0.2013"
## [1] "Cophenetic: 0.4464 (density:1.3557), Percentile:0.238"
```



```
## [1] 0.1311 0.2013 0.2380
```

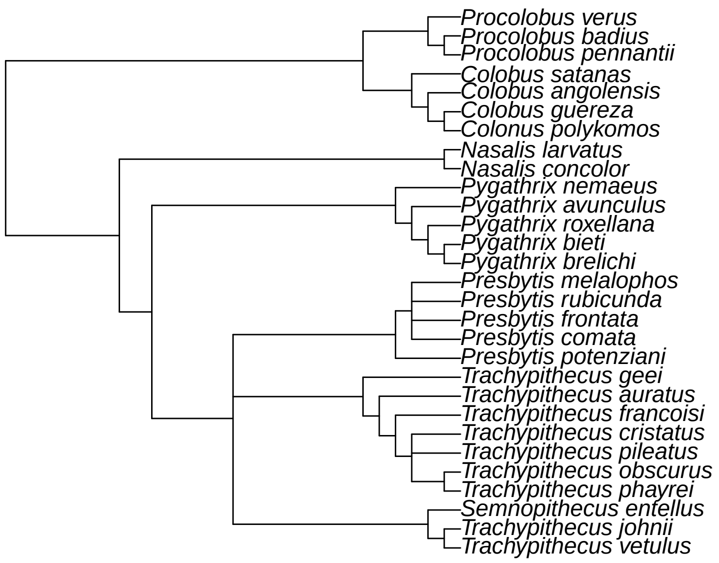
A.6.2 A real example

The tree considered in Section 4.5.1 is the following:

```
t1t2="((((Colonus_polykomos,Colobus_guereza)
,Colobus_angolensis),Colobus_satanas),(
(Procolobus_pennantii,Procolobus_badius),Procolobus_verus))"
t3 = "(Nasalis_concolor,Nasalis_larvatus)"
t4 = "((((Pygathrix_brelichi,Pygathrix_bieti)
,Pygathrix_roxellana),Pygathrix_avunculus),Pygathrix_nemaeus)"
t5 = "(Presbytis_potenziani,
(Presbytis_comata,Presbytis_frontata,Presbytis_rubicunda,Presbytis_melalophos))"
t6 = "(((((Trachypithecus_phayrei,Trachypithecus_obscurus)
,Trachypithecus_pileatus,Trachypithecus_cristatus)
,Trachypithecus_francoisi),Trachypithecus_auratus)
,Trachypithecus_geei)"
t7 = "((Trachypithecus_vetulus,Trachypithecus_johnii)
,Semnopithecus_entellus)"
t8 = paste("(",t7,",",t6,",",t5,")",sep="")
t9 = paste("(",t8,",",t4,")",sep="")
t10 = paste("(",t9,",",t3,")",sep="")
all.txt= paste("(",t10,",",t1t2,");",sep="")
tree=read.tree(text = all.txt)
```

The following command depicts this tree in Fig. 4.15:

```
plot(tree)
```



The three balance indices of this tree and their normalized values are obtained as follows:

```
balance.indices(tree)
```

```
## Colles-Like      Sackin  Cophenetic
##     81.4844    161.0000    655.0000
```

```
balance.indices(tree, norm = T)
```

```
## Colles-Like      Sackin  Cophenetic
##   0.1689766   0.3259259   0.1792556
```

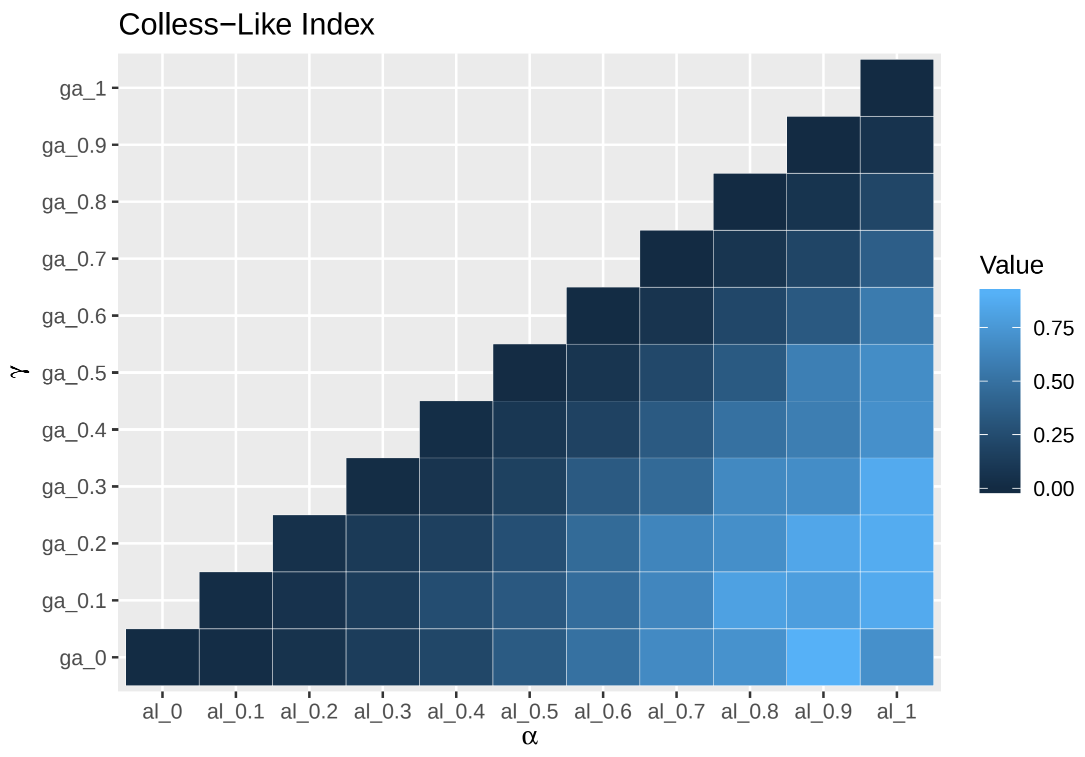
To establish a relationship of the previous tree with the α-γ-model we have computed the percentile of the tree for every $(\alpha, \gamma) \in \{0, 0.1, 0.2, \ldots, 0.9, 1\}^2$ with $\gamma \leqslant \alpha$, in order to check the values of the parameters (α, γ) for which the tree has higher values.

```
percentile.matrix = matrix(NA,nrow = 11,ncol = 11,
           dimnames = list(paste("al",seq(0,1,0.1),sep="_"),
                             paste("ga",seq(0,1,0.1),sep="_")))
 for(a in seq(0,1,0.1)){
   for(g in seq(0,a,0.1)){
     pers = distribution(tree,a,g,db.path = database.location)
     percentile.matrix[a*10+1,g*10+1] = pers[1]
   }
 }
```

The results are available at "C4-real-example-percentiles.txt" and the percentile plots at "C4-real-example-percentile-plots.pdf".

The heatmap of the Fig. 4.16 is obtained with the following code:

```
require(ggplot2)
require(reshape2)
m1 = melt(percentile.matrix[,],na.rm=T)
names(m1)=c("Alpha","Gamma","Value" )
a.g.range=seq(0,1,by = 0.1)
gp1=ggplot(data=m1,aes(x=Alpha,y=Gamma,fill=Value))+
              geom_tile(color="white")
gp1 + labs(title = "Colless-Like Index", x=bquote(alpha),
              y=bquote(gamma))
```



The parameters yielding the highest percentiles are

alpha	gamma	Percentile
0.9	0.0	0.9031
1.0	0.2	0.8725
1.0	0.1	0.8620
1.0	0.3	0.8600

A.6.3 Computation of the mean and variance

First of all, we upload the treeBASE database

```
load("./treeBASE-database.RData",verbose = T)
```

```
## Loading objects:
##   tb.ape
```

Then, we compute each one of the three indices of all the trees and also its normalized version. The third vector is the number of leaves of each tree.

```
tb.idx = t(sapply(tb.ape,balance.indices))
tb.idx.norm = t(sapply(tb.ape,balance.indices,norm = TRUE))
tb.n = t(sapply(tb.ape,Ntip))
```

The results are available in the files “C4-tb-indices.txt”,“C4-tb-indices-norm.txt” and “C4-tb-n.txt”.

Now, we can compute the means and variance of every index

```
tb.means = list(c(),c())
tb.vars = list(c(),c())
for(n in 3:max(tb.n)){
  number.trees = which(tb.n==n)
  if(length(number.trees)>0){
    aux = list(tb.idx[number.trees,],
               tb.idx.norm[number.trees,])
    if(!is.null(dim(aux[[1]]))){
      means = list(colMeans(aux[[1]]),colMeans(aux[[2]]))
      vars = list(apply(aux[[1]],2,var),apply(aux[[2]],2,var))
    }
    else{
      means= aux
      vars = list(c(0,0,0),c(0,0,0))
    }
    num = length(number.trees)
    tb.means[[1]] = rbind(tb.means[[1]],c(n,means[[1]],num))
    tb.vars[[1]] = rbind(tb.vars[[1]],c(n,vars[[1]],num))
    tb.means[[2]] = rbind(tb.means[[2]],c(n,means[[2]],num))
    tb.vars[[2]] = rbind(tb.vars[[2]],c(n,vars[[2]],num))
  }
}
```

The results of these computations are available in “C4-tb-means-ALL.txt”,“C4-tb-means-norm-ALL.txt”, “C4-tb-vars-ALL.txt” and “C4-tb-vars-norm-ALL.txt”.

We have chosen the trees with $n < 300$ number of leaves and we have considered only those with more than 30 trees

```
final.pos = which(tb.means[[1]][,1]==300)
morethan30=which(tb.means[[1]][1:final.pos,5]>30)
tb.means.reg = tb.means[[1]][morethan30,]
tb.means.reg.norm = tb.means[[2]][morethan30,]
tb.vars.reg = tb.vars[[1]][morethan30,]
tb.vars.reg.norm = tb.vars[[2]][morethan30,]
```

These results are available in “C4-tb-means-regression.txt”, “C4-tb-means-regression-norm.txt”, “C4-tb-vars-regression.txt” and “C4-tb-varss-regression-norm.txt”.

We have computed the regressions for the Colless-like index. These are the regressions of its mean values:

```
reg.cl=summary(lm(log(tb.means.reg[,2])~log(tb.means.reg[,1])))
reg.cl
```

```
##
## Call:
## lm(formula = log(tb.means.reg[, 2]) ~ log(tb.means.reg[, 1]))
```

```
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.58088 -0.05385  0.01801  0.09498  0.33918
##
## Coefficients:
##                        Estimate Std. Error t value Pr(>|t|)
## (Intercept)            -0.62477    0.07181    -8.7 9.17e-14 ***
## log(tb.means.reg[, 1])  1.58463    0.01860    85.2  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1495 on 96 degrees of freedom
## Multiple R-squared:  0.9869, Adjusted R-squared:  0.9868
## F-statistic:  7260 on 1 and 96 DF,  p-value: < 2.2e-16
```

```
reg.cl.norm=summary(lm(log(tb.means.reg.norm[,2])~
                         log(tb.means.reg.norm[,1])))
reg.cl.norm
```

```
##
## Call:
## lm(formula = log(tb.means.reg.norm[, 2]) ~ log(tb.means.reg.norm[,
##     1]))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.48405 -0.06863  0.00563  0.06702  0.50399
##
## Coefficients:
##                             Estimate Std. Error t value Pr(>|t|)
## (Intercept)                  0.51414    0.06350   8.097 1.77e-12 ***
## log(tb.means.reg.norm[, 1]) -0.56860    0.01645 -34.573  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1322 on 96 degrees of freedom
## Multiple R-squared:  0.9257, Adjusted R-squared:  0.9249
## F-statistic:  1195 on 1 and 96 DF,  p-value: < 2.2e-16
```

These are the regressions of the variances of the values of the Colless-like index:

```
reg.cl.var=summary(lm(log(tb.vars.reg[,2])~log(tb.vars.reg[,1])))
reg.cl.var
```

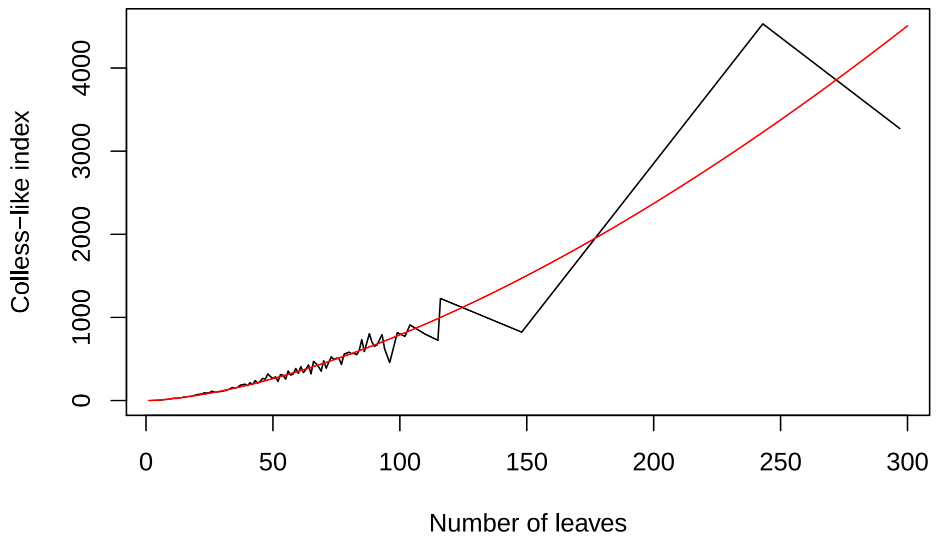
```
##
## Call:
## lm(formula = log(tb.vars.reg[, 2]) ~ log(tb.vars.reg[, 1]))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.1727 -0.1281  0.0881  0.2379  1.9181
##
## Coefficients:
##                        Estimate Std. Error t value Pr(>|t|)
## (Intercept)           -2.57579    0.24489  -10.52   <2e-16 ***
## log(tb.vars.reg[, 1])  3.12798    0.06342   49.32   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5099 on 96 degrees of freedom
## Multiple R-squared:  0.962,  Adjusted R-squared:  0.9616
## F-statistic:  2432 on 1 and 96 DF,  p-value: < 2.2e-16
```

```
reg.cl.var.norm=summary(lm(log(tb.vars.reg.norm[,2])~
                             log(tb.vars.reg.norm[,1])))
reg.cl.var.norm
```

```
##
## Call:
## lm(formula = log(tb.vars.reg.norm[, 2]) ~ log(tb.vars.reg.norm[,
##     1]))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.7861 -0.1327  0.0120  0.1928  2.2477
##
## Coefficients:
##                             Estimate Std. Error t value Pr(>|t|)
## (Intercept)                 -0.29796    0.22059  -1.351     0.18
## log(tb.vars.reg.norm[, 1]) -1.17848    0.05713 -20.628   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4593 on 96 degrees of freedom
## Multiple R-squared:  0.8159, Adjusted R-squared:  0.814
## F-statistic: 425.5 on 1 and 96 DF,  p-value: < 2.2e-16
```

The following code produces Fig. 4.17(a):

```
plot(tb.means.reg[,1],tb.means.reg[,2],type = "l",
        xlab="Number of leaves",ylab="Colless-like index")
lines(1:300,exp(reg.cl$coefficients[1,1])*(1:300)^
        reg.cl$coefficients[2,1],col="red")
```



As to the Sackin index, these are the regressions of its mean values:

```
reg.sa=summary(lm(log(tb.means.reg[,3])~log(tb.means.reg[,1])))
reg.sa
```

```
##
## Call:
## lm(formula = log(tb.means.reg[, 3]) ~ log(tb.means.reg[, 1]))
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.309558 -0.024226  0.008616  0.048661  0.232089
##
## Coefficients:
##                        Estimate Std. Error t value Pr(>|t|)
## (Intercept)             0.37285    0.03870   9.635 9.03e-16 ***
## log(tb.means.reg[, 1])  1.43583    0.01002 143.267  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.08057 on 96 degrees of freedom
## Multiple R-squared:  0.9953, Adjusted R-squared:  0.9953
## F-statistic: 2.053e+04 on 1 and 96 DF,  p-value: < 2.2e-16
```

```
reg.sa.norm=summary(lm(log(tb.means.reg.norm[,3])~
                         log(tb.means.reg.norm[,1])))
reg.sa.norm
```

```
##
```

```
## Call:
## lm(formula = log(tb.means.reg.norm[, 3]) ~ log(tb.means.reg.norm[,
##     1]))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.33568 -0.03219  0.00722  0.05739  0.24989
##
## Coefficients:
##                             Estimate Std. Error t value Pr(>|t|)
## (Intercept)                  0.83896    0.04166   20.14   <2e-16 ***
## log(tb.means.reg.norm[, 1]) -0.53463    0.01079  -49.55   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.08675 on 96 degrees of freedom
## Multiple R-squared:  0.9624, Adjusted R-squared:  0.962
## F-statistic:  2455 on 1 and 96 DF,  p-value: < 2.2e-16
```

These are the regressions of the variances of the values of the Sackin index:

```
reg.sa.var=summary(lm(log(tb.vars.reg[,3])~log(tb.vars.reg[,1])))
reg.sa.var
```

```
##
## Call:
## lm(formula = log(tb.vars.reg[, 3]) ~ log(tb.vars.reg[, 1]))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.5707 -0.1466  0.1062  0.2778  1.8728
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)           -3.43962    0.26752  -12.86   <2e-16 ***
## log(tb.vars.reg[, 1])  3.22249    0.06929   46.51   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.557 on 96 degrees of freedom
## Multiple R-squared:  0.9575, Adjusted R-squared:  0.9571
## F-statistic:  2163 on 1 and 96 DF,  p-value: < 2.2e-16
```

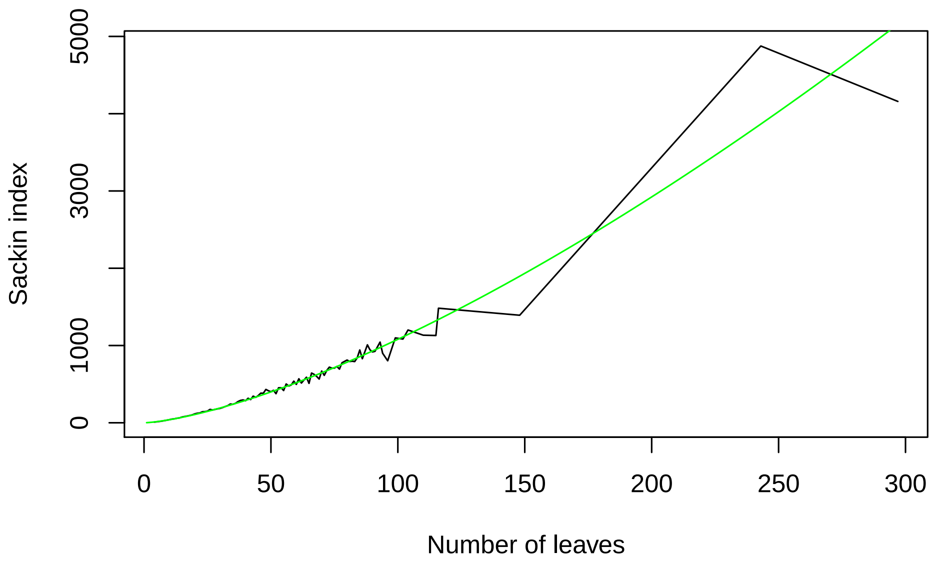
```
reg.sa.var.norm=summary(lm(log(tb.vars.reg.norm[,3])~
                             log(tb.vars.reg.norm[,1])))
reg.sa.var.norm
```

```
##
## Call:
## lm(formula = log(tb.vars.reg.norm[, 3]) ~ log(tb.vars.reg.norm[,
##     1]))
```

```
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.3981 -0.1552  0.0853  0.2259  2.0207
##
## Coefficients:
##                            Estimate Std. Error t value Pr(>|t|)
## (Intercept)                 -1.4750     0.2490  -5.923 4.92e-08 ***
## log(tb.vars.reg.norm[, 1])  -0.9082     0.0645 -14.081  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5185 on 96 degrees of freedom
## Multiple R-squared:  0.6738, Adjusted R-squared:  0.6704
## F-statistic: 198.3 on 1 and 96 DF,  p-value: < 2.2e-16
```

The following code produces Fig. 4.17(b):

```
plot(tb.means.reg[,1],tb.means.reg[,3],type = "l",
        xlab="Number of leaves",ylab="Sackin index")
lines(1:300,exp(reg.sa$coefficients[1,1])*(1:300)^
        reg.sa$coefficients[2,1],col="green")
```



Finally, these are the regressions of the means values for the total cophenetic index:

```
reg.co=summary(lm(log(tb.means.reg[,4])~log(tb.means.reg[,1])))
reg.co
```

```
##
```

```
## Call:
## lm(formula = log(tb.means.reg[, 4]) ~ log(tb.means.reg[, 1]))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.61240 -0.04981  0.03907  0.09208  0.34377
##
## Coefficients:
##                        Estimate Std. Error t value Pr(>|t|)
## (Intercept)            -1.66353    0.07471  -22.27   <2e-16 ***
## log(tb.means.reg[, 1])  2.54769    0.01935  131.67   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1556 on 96 degrees of freedom
## Multiple R-squared:  0.9945, Adjusted R-squared:  0.9944
## F-statistic: 1.734e+04 on 1 and 96 DF,  p-value: < 2.2e-16
```

```
reg.co.norm=summary(lm(log(tb.means.reg.norm[,4])~
                         log(tb.means.reg.norm[,1])))
reg.co.norm
```

```
##
## Call:
## lm(formula = log(tb.means.reg.norm[, 4]) ~ log(tb.means.reg.norm[,
##     1]))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.51558 -0.05255 -0.00597  0.06874  0.50857
##
## Coefficients:
##                             Estimate Std. Error t value Pr(>|t|)
## (Intercept)                  0.81751    0.05723   14.29   <2e-16 ***
## log(tb.means.reg.norm[, 1]) -0.60554    0.01482  -40.85   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1192 on 96 degrees of freedom
## Multiple R-squared:  0.9456, Adjusted R-squared:  0.945
## F-statistic:  1669 on 1 and 96 DF,  p-value: < 2.2e-16
```

These are the regressions of the variances of the values of the total cophenetic index:

```
reg.co.var=summary(lm(log(tb.vars.reg[,4])~log(tb.vars.reg[,1])))
reg.co.var
```

```
##
## Call:
## lm(formula = log(tb.vars.reg[, 4]) ~ log(tb.vars.reg[, 1]))
##
```

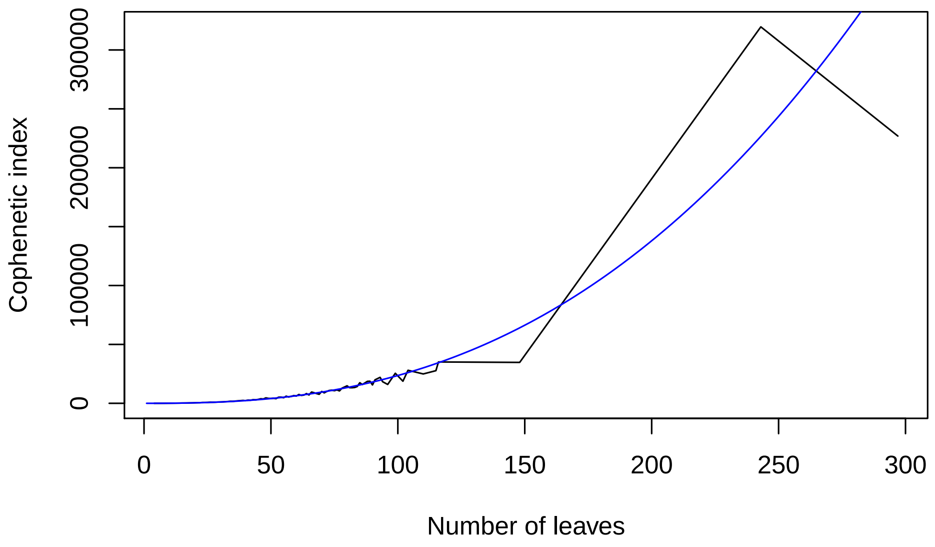
```
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.6544 -0.1626  0.1001  0.2883  1.9319
##
## Coefficients:
##                        Estimate Std. Error t value Pr(>|t|)
## (Intercept)           -5.50441    0.28383  -19.39   <2e-16 ***
## log(tb.vars.reg[, 1])  5.20711    0.07351   70.83   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.591 on 96 degrees of freedom
## Multiple R-squared:  0.9812, Adjusted R-squared:  0.981
## F-statistic:  5018 on 1 and 96 DF,  p-value: < 2.2e-16
```

```r
reg.co.var.norm=summary(lm(log(tb.vars.reg.norm[,4])~
                             log(tb.vars.reg.norm[,1])))
reg.co.var.norm
```

```
##
## Call:
## lm(formula = log(tb.vars.reg.norm[, 4]) ~ log(tb.vars.reg.norm[,
##     1]))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.2678 -0.1254  0.0406  0.2242  2.2615
##
## Coefficients:
##                             Estimate Std. Error t value Pr(>|t|)
## (Intercept)                -0.54233    0.24535   -2.21   0.0294 *
## log(tb.vars.reg.norm[, 1]) -1.09934    0.06354  -17.30   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5108 on 96 degrees of freedom
## Multiple R-squared:  0.7572, Adjusted R-squared:  0.7546
## F-statistic: 299.3 on 1 and 96 DF,  p-value: < 2.2e-16
```

And the code producing Fig. 4.17(c):

```r
plot(tb.means.reg[,1],tb.means.reg[,4],type = "l",
        xlab="Number of leaves",ylab="Cophenetic index")
lines(1:300,exp(reg.co$coefficients[1,1])*(1:300)^
        reg.co$coefficients[2,1],col="blue")
```



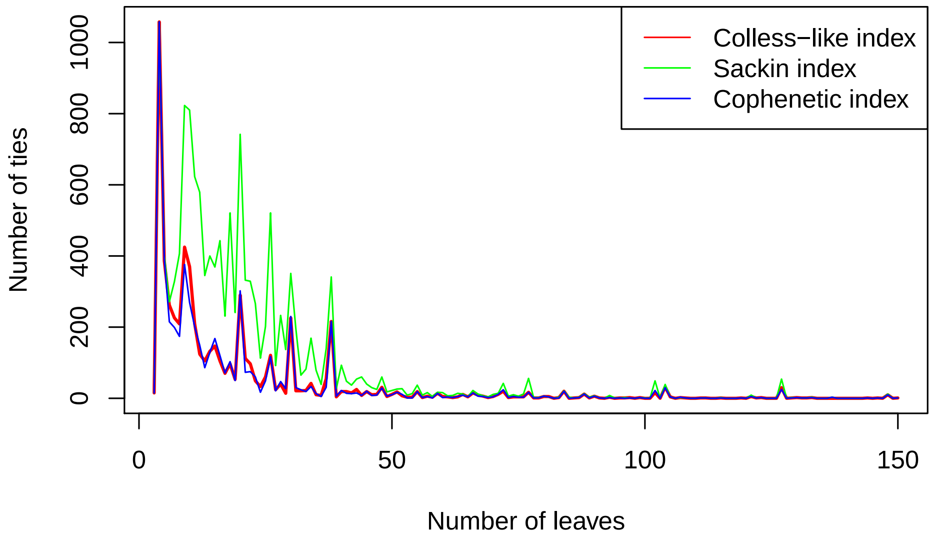
A.6.4 Computation of the number of ties

In this section we compute the number of ties of the three balance indices under consideration. For every number of leaves n and for every index, we have computed the numbers of pairs of trees with n leaves in TreeBASE having the same value of the corresponding index:

```
ties.cl=c()
ties.sa=c()
ties.co=c()
for(i in 3:max(tb.n)){
  aux=tb.idx[tb.n==i,]
  ties.cl=rbind(ties.cl,c(i,sum(choose(table(aux[,1]),2))))
  ties.sa=rbind(ties.sa,c(i,sum(choose(table(aux[,2]),2))))
  ties.co=rbind(ties.co,c(i,sum(choose(table(aux[,3]),2))))
}
```

The following commands produce Fig. 4.18:

```
plot(3:150,ties.cl[1:148,2],type="l",xlab="Number of leaves",
        ylab="Number of ties",col="red",lwd=2)
lines(3:150,ties.sa[1:148,2],col="green",lwd=1)
lines(3:150,ties.co[1:148,2],col="blue",lwd=1,lty=1)
legend("topright",legend=c("Colless-like index",
          "Sackin index","Cophenetic index"),
          lty=c("solid","solid","solid"),
          col=c("red","green","blue"))
```



A.6.4 Computation of Spearman's rank correlation

The global Spearman's rank correlation coefficient of $\mathfrak{C}$ and S is

```
cor(tb.idx[,1],tb.idx[,2],method="spearman")
```

```
## [1] 0.97645
```

And the global Spearman's rank correlation coefficient of of $\mathfrak{C}$ and Φ is

```
cor(tb.idx[,1],tb.idx[,3],method="spearman")
```

```
## [1] 0.9618565
```

We compute now the Spearman rank correlation coefficient of the indices on all trees in TreeBASE grouping them by their number of leaves n. As usual, we have considered only those numbers of leaves with more than 30 trees:

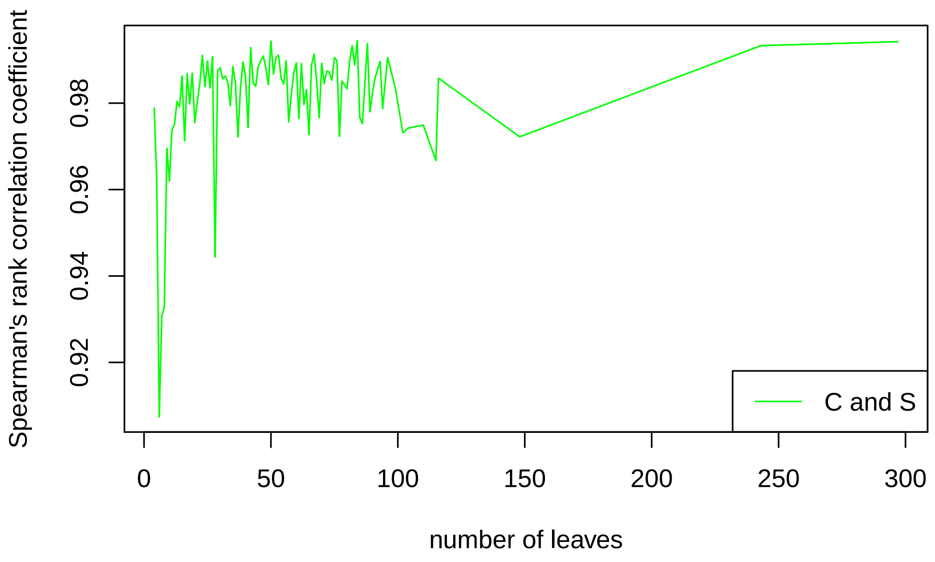
```
spearman.sackin=c()
for(i in 1:max(tb.n)){
  aux=tb.idx[tb.n==i,]
  if(dim(aux)[1]>30){
    aux2=rank(aux[,1])
    aux3=rank(aux[,2])
    spearman.sackin=rbind(spearman.sackin,
          c(i,cor(aux2,aux3,method="spearman")))
  }
}
colnames(spearman.sackin)=c("Num.Leaves","SpearmanRank")
```

```
write.table(spearman.sackin,file="./C4-tb-spearman-CL-S.txt",
              col.names=T,row.names=F)

spearman.coph=c()
for(i in 1:max(tb.n)){
  aux=tb.idx[tb.n==i,]
  if(dim(aux)[1]>30){
    aux2=rank(aux[,1])
    aux3=rank(aux[,3])
    spearman.coph=rbind(spearman.coph,
          c(i,cor(aux2,aux3,method="spearman")))
  }
}
colnames(spearman.sackin)=c("Num.Leaves","SpearmanRank")
write.table(spearman.coph,file="./C4-tb-spearman-CL-C.txt",
              col.names=T,row.names=F)
```

All the values are available in the files ''C4-tb-spearman-CL-S.txt''and''C4-tb-spearman-CL-C.txt".

Fig. 4.19 is produced with the following commands:

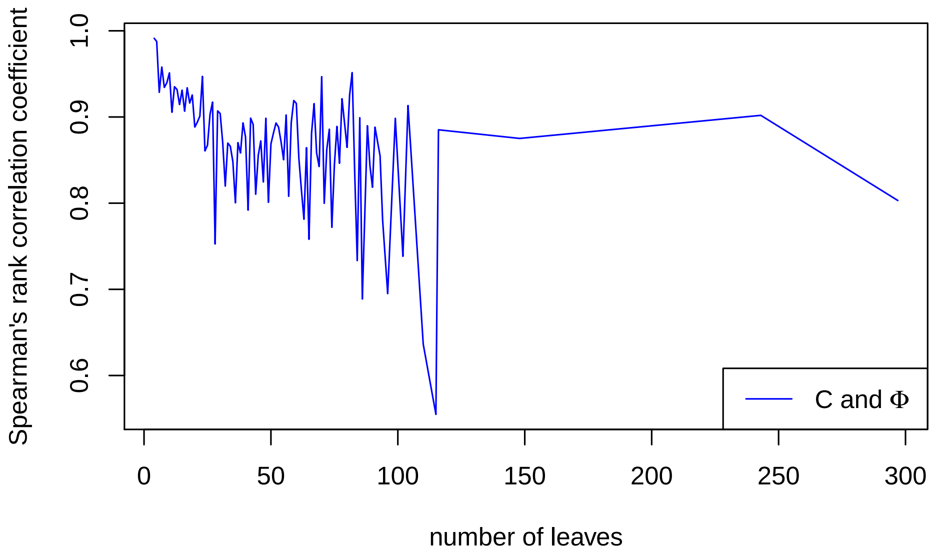
```
plot(spearman.sackin[,1],spearman.sackin[,2],type="l",
      col="green",xlab="number of leaves",
      ylab="Spearman's rank correlation coefficient")
legend("bottomright",legend="C and S"
         #"Colless-like and Sackin indices"
         , lty= "solid"  ,col= "green" )
```



```r
summary(lm(spearman.sackin[,2]~spearman.sackin[,1]))
```

```
##
## Call:
## lm(formula = spearman.sackin[, 2] ~ spearman.sackin[, 1])
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.069523 -0.004901  0.003692  0.007571  0.013644
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)          9.764e-01  2.149e-03 454.235   <2e-16 ***
## spearman.sackin[, 1] 8.648e-05  2.988e-05   2.894   0.0047 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.0128 on 96 degrees of freedom
## Multiple R-squared:  0.08025,    Adjusted R-squared:  0.07067
## F-statistic: 8.377 on 1 and 96 DF,  p-value: 0.004704
```

```r
plot(spearman.coph[,1],spearman.coph[,2],type="l",
      col="blue",xlab="number of leaves",
      ylab="Spearman's rank correlation coefficient")
legend("bottomright",legend= bquote(paste("C and ",Phi," "))
         #"Colless-like and Cophenetic indices"
         , lty= "solid" ,col=  "blue" )
```



We also can compute the regression of the results

```r
summary(lm(spearman.sackin[,2]~spearman.sackin[,1]))
```

```
##
## Call:
## lm(formula = spearman.sackin[, 2] ~ spearman.sackin[, 1])
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.069523 -0.004901  0.003692  0.007571  0.013644
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)          9.764e-01  2.149e-03 454.235   <2e-16 ***
## spearman.sackin[, 1] 8.648e-05  2.988e-05   2.894   0.0047 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.0128 on 96 degrees of freedom
## Multiple R-squared:  0.08025,    Adjusted R-squared:  0.07067
## F-statistic: 8.377 on 1 and 96 DF,  p-value: 0.004704
```

```r
summary(lm(spearman.coph[,2]~spearman.coph[,1]))
```

```
##
## Call:
## lm(formula = spearman.coph[, 2] ~ spearman.coph[, 1])
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.27550 -0.01853  0.01222  0.03897  0.15033
##
## Coefficients:
##                      Estimate Std. Error t value Pr(>|t|)
## (Intercept)         0.9012445  0.0110186  81.793  < 2e-16 ***
## spearman.coph[, 1] -0.0006160  0.0001532  -4.022 0.000115 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.06563 on 96 degrees of freedom
## Multiple R-squared:  0.1442, Adjusted R-squared:  0.1353
## F-statistic: 16.17 on 1 and 96 DF,  p-value: 0.0001151
```

A test on the distribution of TreeBASE

```
## Loading objects:
##   tb.ape
```

In this case, we focus our study with the normalized Colless-like index of the TreeBASE, so we consider only the following array of its normalized indices

```
tb.colless=tb.idx.norm[,1]
```

The following functions extract the data from our database and perform for every pair of `alpha` and `gamma` a `chisq.test`. Then, we will study in detail each interesting case.

```
read.idx = function(n,alpha,gamma,norm=T){
  file = paste("CollessLikeDataBase_n",n,"_a",alpha*100,"_g",
               gamma*100,"_r5000.txt",sep="")
  folder = paste(database.location,"n",n,"/",sep="")
  if(file %in% dir(folder))
    indices.list=read.table(file=paste(folder,file,sep=""),
                            header=TRUE)
  if(norm) indices.list = indices.list[,1]/(( log(0+exp(1)) +
                           log(2+exp(1)) )*(n-1)*(n-2)/4)
  else indices.list = indices.list[,1]
  return(indices.list)
}

pis.ag = function(alpha,gamma,intervals,dens=F){
  all.trees.study = unlist(lapply(3:50,read.idx,alpha=alpha,
                           gamma=gamma))
  brs = seq(0,1,length.out = intervals+1)
  fe = hist(all.trees.study,breaks = brs,plot=F)
  pis = fe$counts/sum(fe$counts)
  if(dens){
    dens = density(all.trees.study)
    new.pi = c()
    for(i in 1:intervals){
      aux=integrate(splinefun(dens$x,dens$y), brs[i],
                    brs[i+1])$value
      new.pi= c(new.pi,aux)
    }
    new.pi=new.pi/sum(new.pi)
    return(list(pis,new.pi,dens))
  }
  else return(pis)
}

parameters.study = function(fo, intervals,dens=F,info=FALSE){
  ntb = sum(fo$counts)
  parameters = expand.grid(seq(0,1,0.1),seq(0,1,0.1))
  parameters = parameters[which(parameters[,1]>=parameters[,2]),]
  pvalues=c()
  all.info=list()
  for(i in 1:65){
    if(i!=11){
      pis = pis.ag(parameters[i,1],parameters[i,2],intervals,
                  dens = dens)
```

```r
      table.info=grouping(fo$counts,pis*ntb,fo$breaks)
      out.test=chisq.test(table.info[,1],p=table.info[,2]/ntb)
      pvalues=c(pvalues,out.test$p.value)
      print(paste("a:",parameters[i,1],", g:",parameters[i,2],
                    "-->",out.test$p.value))
      if(info)print(out.test)
      all.info[[i]]=list(table.info,out.test)
    }
    else{
      print("a: 1 , g: 0  --> ERROR")
      pvalues=c(pvalues,-1)
    }
  }
  print("a: 1 , g: 1  --> ERROR")
  parameters=cbind(parameters,c(pvalues,-1))
  return(list(parameters,all.info))
}

grouping = function(xx,yy,breaks){
  ois = xx[]
  eis = yy[]
  to.modify = which(eis<5)
  while(length(to.modify)>0){
    to.modify = to.modify[1]
    N = length(eis)
    if(to.modify>1){
      if(to.modify<N){
        aux1 = eis[to.modify-1]+eis[to.modify]
        aux2 = eis[to.modify]+eis[to.modify+1]
        if(aux1<aux2){
          eis[to.modify-1] = aux1
          ois[to.modify-1] = ois[to.modify-1]+ois[to.modify]
          breaks = breaks[-to.modify]
        }
        else{
          eis[to.modify+1] = aux2
          ois[to.modify+1] = ois[to.modify]+ois[to.modify+1]
          breaks = breaks[-(to.modify+1)]
        }
        eis = eis[-to.modify]
        ois = ois[-to.modify]
      }
      else{ ## to.modify=N
        eis[N-1] = eis[N]+eis[N-1]
        eis = eis[-N]
        ois[N-1] = ois[N]+ois[N-1]
        ois = ois[-N]
        breaks = breaks[-(N)]
      }
    }
    else{ ## to.modify=1
      eis[2] = eis[1]+eis[2]
      eis = eis[-1]
      ois[2] = ois[1]+ois[2]
      ois = ois[-1]
      breaks = breaks[-2]
    }
    to.modify=which(eis<5)
  }
  N=length(breaks)
  return(cbind(ois,eis,linf=breaks[1:(N-1)],lsup=breaks[2:N]))
}
```

The following code executes the study

```
intervals=100
fo = hist(tb.colless,breaks = intervals,plot=F)
ntb = sum(fo$counts)
results.study=parameters.study(fo,intervals)
```

```
## [1] "a: 0 , g: 0 --> 0"
## [1] "a: 0.1 , g: 0 --> 0"
## [1] "a: 0.2 , g: 0 --> 0"
## [1] "a: 0.3 , g: 0 --> 0"
## [1] "a: 0.4 , g: 0 --> 0"
## [1] "a: 0.5 , g: 0 --> 0"
## [1] "a: 0.6 , g: 0 --> 0"
## [1] "a: 0.7 , g: 0 --> 0"
## [1] "a: 0.8 , g: 0 --> 0"
## [1] "a: 0.9 , g: 0 --> 0"
## [1] "a: 1 , g: 0  --> ERROR"
## [1] "a: 0.1 , g: 0.1 --> 0"
## [1] "a: 0.2 , g: 0.1 --> 0"
## [1] "a: 0.3 , g: 0.1 --> 0"
## [1] "a: 0.4 , g: 0.1 --> 0"
## [1] "a: 0.5 , g: 0.1 --> 0"
## [1] "a: 0.6 , g: 0.1 --> 0"
## [1] "a: 0.7 , g: 0.1 --> 0"
## [1] "a: 0.8 , g: 0.1 --> 0"
## [1] "a: 0.9 , g: 0.1 --> 0"
## [1] "a: 1 , g: 0.1 --> 0"
## [1] "a: 0.2 , g: 0.2 --> 0"
## [1] "a: 0.3 , g: 0.2 --> 0"
## [1] "a: 0.4 , g: 0.2 --> 0"
## [1] "a: 0.5 , g: 0.2 --> 0"
## [1] "a: 0.6 , g: 0.2 --> 0"
## [1] "a: 0.7 , g: 0.2 --> 0"
## [1] "a: 0.8 , g: 0.2 --> 0"
## [1] "a: 0.9 , g: 0.2 --> 0"
## [1] "a: 1 , g: 0.2 --> 0"
## [1] "a: 0.3 , g: 0.3 --> 0"
## [1] "a: 0.4 , g: 0.3 --> 0"
## [1] "a: 0.5 , g: 0.3 --> 0"
## [1] "a: 0.6 , g: 0.3 --> 0"
## [1] "a: 0.7 , g: 0.3 --> 0"
## [1] "a: 0.8 , g: 0.3 --> 0"
## [1] "a: 0.9 , g: 0.3 --> 0"
## [1] "a: 1 , g: 0.3 --> 0"
## [1] "a: 0.4 , g: 0.4 --> 0"
## [1] "a: 0.5 , g: 0.4 --> 0"
## [1] "a: 0.6 , g: 0.4 --> 0"
## [1] "a: 0.7 , g: 0.4 --> 1.01304353682268e-113"
## [1] "a: 0.8 , g: 0.4 --> 0"
## [1] "a: 0.9 , g: 0.4 --> 0"
## [1] "a: 1 , g: 0.4 --> 0"
## [1] "a: 0.5 , g: 0.5 --> 0"
## [1] "a: 0.6 , g: 0.5 --> 0"
## [1] "a: 0.7 , g: 0.5 --> 0"
## [1] "a: 0.8 , g: 0.5 --> 2.42464512709513e-168"
## [1] "a: 0.9 , g: 0.5 --> 0"
## [1] "a: 1 , g: 0.5 --> 0"
## [1] "a: 0.6 , g: 0.6 --> 0"
## [1] "a: 0.7 , g: 0.6 --> 0"
## [1] "a: 0.8 , g: 0.6 --> 0"
## [1] "a: 0.9 , g: 0.6 --> 0"
## [1] "a: 1 , g: 0.6 --> 0"
## [1] "a: 0.7 , g: 0.7 --> 0"
## [1] "a: 0.8 , g: 0.7 --> 0"
## [1] "a: 0.9 , g: 0.7 --> 0"
## [1] "a: 1 , g: 0.7 --> 0"
## [1] "a: 0.8 , g: 0.8 --> 0"
## [1] "a: 0.9 , g: 0.8 --> 0"
## [1] "a: 1 , g: 0.8 --> 0"
## [1] "a: 0.9 , g: 0.9 --> 0"
## [1] "a: 1 , g: 0.9 --> 0"
## [1] "a: 1 , g: 1  --> ERROR"
```

The following cases are those with p-value different of 0

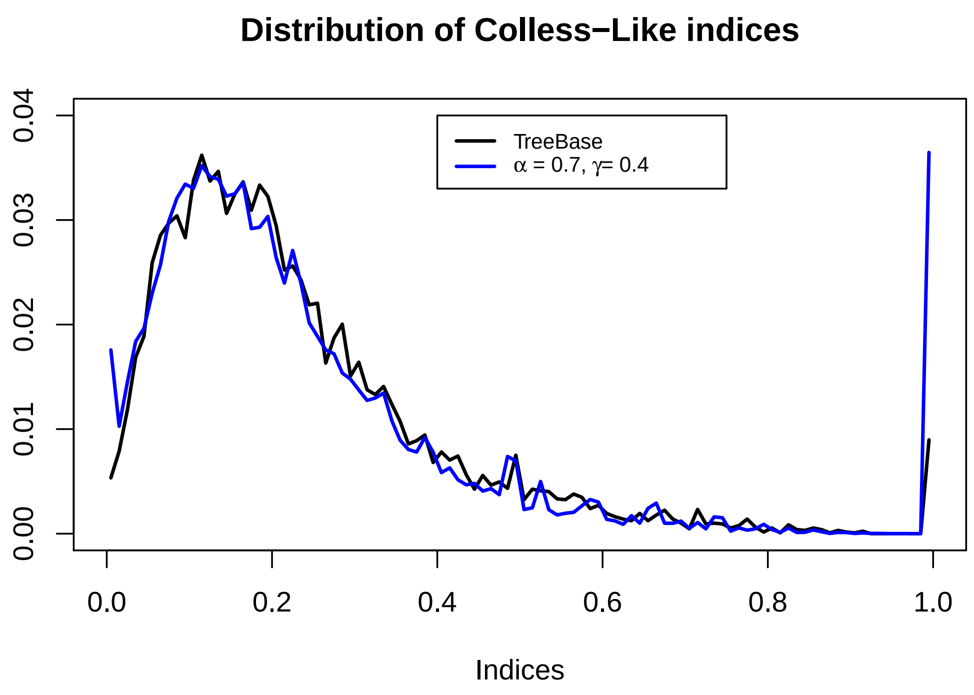
```
results.study[[1]][which(results.study[[1]][,3]>0),]
```

```
##    Var1 Var2 c(pvalues, -1)
## 52  0.7  0.4  1.013044e-113
## 64  0.8  0.5  2.424645e-168
```

```
p42 = pis.ag(0.7,0.4,intervals)
p49 = pis.ag(0.8,0.5,intervals)
```

Although the p-value is very small, we plot the results in Fig. 4.21 as follows:

```
plot(-1,-1,xlim =c(0,1),ylim=c(0,0.04),xlab="Indices",ylab="",
     pch=20,col="white",
     main="Distribution of Colless-Like indices")
lines(fo$mids,fo$counts/ntb,lwd=2)
lines(fo$mids,p42,pch=20,col="blue",lwd=2)
legend(c(0.4,0.75),c(0.04,0.033),legend = c("TreeBase",
        expression(paste(alpha," = 0.7, ",gamma,"= 0.4"))),
        col=c("black","blue"),lwd=2 ,cex=0.75)
```



Besides the whole TreeBASE as explained above, we have also considered differents subsets of it

```
tb.kind=function(tr)return(tr$kind)
kind=unlist(lapply(tb.ape ,tb.kind))
tb.type=function(tr)return(tr$type)
type=unlist(lapply(tb.ape ,tb.type))
idx.spe=which(kind=="Species Tree")
idx.gen=which(kind=="Gene Tree")
idx.spe.con=intersect(idx.spe,which(type=="Consensus"))
idx.spe.sin=intersect(idx.spe,which(type=="Single"))
tb.spe=tb.colless[idx.spe]
tb.spe.n = tb.n[idx.spe,]
tb.spe.con=tb.colless[idx.spe.con]
tb.spe.con.n = tb.n[idx.spe.con,]
tb.spe.sin=tb.colless[idx.spe.sin]
tb.spe.sin.n = tb.n[idx.spe.sin,]
```

```
erase.attributes = function(tree){
  tree$S.id = NULL
  tree$Tr.id = NULL
  tree$type = NULL
  tree$kind = NULL
  tree$quality = NULL
  return(tree)
}
tb.ape.aux = lapply(tb.ape,erase.attributes)
repetitions = duplicated(tb.ape.aux)
pos.repetitions = (1:length(repetitions))[repetitions]
tb.ape.no.reps = tb.ape[!repetitions]

tb.qua=function(tr)return(tr$quality)
qua=unlist(lapply(tb.ape,tb.qua))
idx.qua=which(qua=="Species Tree")
tb.spe=tb.colless[setdiff(idx.spe,pos.repetitions)]
tb.spe.n = tb.n[setdiff(idx.spe,pos.repetitions), ]
tb.spe.con=tb.colless[setdiff(idx.spe.con,pos.repetitions)]
tb.spe.con.n = tb.n[setdiff(idx.spe.con,pos.repetitions), ]
tb.spe.sin=tb.colless[setdiff(idx.spe.sin,pos.repetitions)]
tb.spe.sin.n = tb.n[setdiff(idx.spe.sin,pos.repetitions), ]
```

We have repeated the study explained above for these subsets of TreeBASE, comparing the distribution of the normalized Colless-like indices of their trees with the estimated theoretical distributions by means of goodness-of-fit tests, and the results have been the same, that is, all p-values have also turned out to be negligible.