Chapter 4: The cophenetic metrics

Lucia Rotger

Contents

Packages required

The functions used in this chapter need these packages to be installed and loaded in the session

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

List of all binary trees

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

```
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 can be consulted in the *List of Trees* folder of the GitHub repository.

General functions

Next functions are needed in some computations of the sections below.

```
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])
}
```

4.4.1 Expected value of D_n^2

The formulas in Theorem 4.29 and 4.30, corresponging to the Yule and uniform models formulas for D_n^2, 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))
}
# [1]  2.666667  9.407407 21.183333
# [4] 38.712000 62.556190 93.172128

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
                                          -as.bigq((n*(n+7))/4)*((big.double.factorial(2*n-2)/big.doubl
}
# [1]   2.666667  10.560000  26.236735
# [4]  52.302343  91.408632 146.247151
```

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 correponding $\mathcal{BT}_n$, under the Yule and uniform models.

First of all, we have considered all the phylogenetic trees in $\mathcal{BT}_n$ for $n = 3,\ldots,8$ obtained with the package *phylonetwork* for *Python* in this section.

Moreover, we have to take into consideration the probabilities of each tree under the Yule model:

```
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)
}
```

Afterwards, we can compute the cophenetic vector with the package *CollessLike* and the expected value and the variance of the square of the cophenetic distance of each n

```
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()
```

Obtaining the following results:

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

agreeing with the figures given by our formulas.

We also have computed the exact values for the variance:

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

In Section 4.5.2 we have made an estimation of their order.

4.5.1 Distribution of cophenetic metrics and their comparison with other metrics

---

4.5.2 On the variance of D_n^2

In order to be able to estimate the asymptotic order of $E(D_n^4)$ and $\sigma^2(D_n^2)$, we have taken the Monte Carlo path. More specifically, 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 $(T, T') \in \mathcal{T}_n \times \mathcal{T}_n$ with this function, using the package *apTreeshape* and convert them into a *phylo* object from package *ape*:

```
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 expression

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

This value is an estimation of $\sigma^2(D_n^2)$ under the corresponding model.

Next instructions show how we can compute these variance of D_n^2 under the Yule and uniform models:

```
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")
```

NOTE: these computations would take a long time to finish, in fact, we have parallelized them with the 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".

Finally, 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 $n = 50, \ldots, 100$. We have only considered the largest values of n because if smaller values were also included in the regression, the regression coefficient was considerably smaller, due to the fact that, for small n, the dominant term is not large enough to significantly stand out from terms of smaller degree.

```
#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)
```

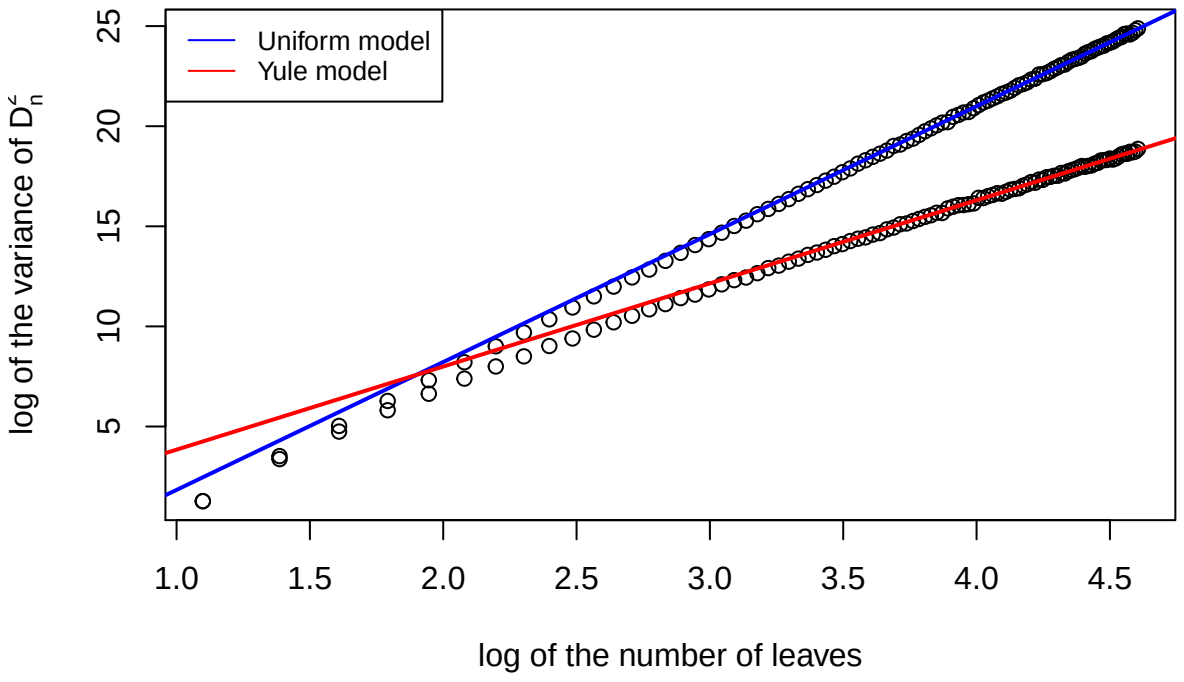
```
##
## Call:
## lm(formula = log(varD2n[48:98, 5]) ~ log(50:100))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.09680 -0.02160  0.00325  0.02406  0.09204
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -4.55620    0.10217  -44.59   <2e-16 ***
## 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
```

The results obtained are summarized in the following table:

	$\log(\overline{\sigma^2}(D_n^2))$	
Model	α	Reg. coefficient R^2
Yule	4.15220	0.99714
Uniform	6.38830	0.99931

Next figure displays $\log(\overline{\sigma^2}(D_n^2))$ under the Yule and uniform models as a function of $\ln(n)$, together with the corresponding regression lines.

```
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)
```



4.5.3 A test on TreeBASE

In this section we report on a very simple experiment to show how $d_{\varphi,2}$ can be used to test evolutionary hypotheses. In this experiment, we have compared the expected value of $d^2_{\varphi,2}$ on $\mathcal{T}_n$ under the uniform and the Yule models with its average value on the set of binary phylogenetic trees with n leaves contained in TreeBASE.

We have considerated all the trees from TreeBASE. The list is available in the *List of Trees* folder of the GitHub repository as a text file or as an `R` object.

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

We have decided to consider only those binary trees contained in TreeBASE with $n \leq 50$ leaves.

```
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)
```

On the other hand, even for those n such that $\text{TreeBASE}_{bin,n}$ is relatively large, in most cases it does not contain many pairs of trees with the same taxa.So, instead of computing the average value of $d^2_{\varphi,2}$ on $\text{TreeBASE}_{bin,n}$ by averaging the values $d^2_{\varphi,2}(T,T')$ for pairs T,T' with exactly the same n taxa, we have

made use of the formula given in Proposition 4.31:

$$E(D_n^2) = 2E(\overline{\Phi}_n^{(2)}) - 2 \cdot \frac{E(S_n)^2}{n} - 4 \cdot \frac{E(\Phi_n)^2}{n(n-1)},$$

as if TreeBASE$_{bin,n}$ was closed under relabelings: that is, we have taken only into account the shapes of the trees contained in it. This is consistent with the fact that our final goal is to test models of evolution that produce tree shapes.

The following functions are needed to do the computations:

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

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


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

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)))
}


Ed2n.v2 = function(EPPhin,n,model="uniform"){
  if (model=="uniform")
    value=2*EPPhin-(2/n)*(EUS(n)^2)-(4/(n*(n-1)))*(EUPhi(n)^2)
  if (model=="yule")
    value=2*EPPhin-(2/n)*(EYS(n)^2)-(4/(n*(n-1)))*(EYPhi(n)^2)
  return(value)
}

ED2n.values = function(n,ESn,EPhin,EPPhin){
  value=2*EPPhin-(2/n)*(ESn^2)-(4/(n*(n-1)))*(EPhin^2)
  return(value)
}

compute.values = function(tree){
  sackin = sackin.index(tree)
  cophen.values = cophen.index2(tree)
```

```
  return(c(sackin,cophen.values))
}
```

Next function is a modification of the function *cophen.index* from the package *CollessLike*. It computes not only the total cophenetic index but also $\overline{\Phi}^{(2)}(T) = \sum_{1 \le i \le j \le n} \varphi_T(i,j)^2$.

```
cophen.index2 = function(tree){
    if(class(tree)=="character")
      tree=read.tree(text = tree)
    if (class(tree)=="phylo")
      tree=graph.edgelist(tree$edge, directed=TRUE)
    if(class(tree)!="igraph")
      stop("Not an igraph object. Please introduce a newick string, an ape tree or an igraph tree.")
    root.node = which(degree(tree,mode="in")==0)
    deg.out = degree(tree,mode="out")
    ########
    if(deg.out[root.node]==1){ #exists a root-edge
      tree = delete.vertices(tree,root.node)
      deg.out = degree(tree,mode="out")
      root.node = which(degree(tree,mode="in")==0)
    }
    leaves = which(deg.out==0)
    root.list = get.shortest.paths(tree,root.node)$vpath
    # COPHENETIC #
    N = length(leaves)
    COPHEN = 0
    COPHEN2 = 0
    for(i in 1:N)
      for(j in i:N){
        aux  = length(intersect(root.list[[leaves[i]]],root.list[[leaves[j]]]))-1
        if(i!=j) COPHEN = COPHEN + aux
        COPHEN2 = COPHEN2 + aux^2
      }
    return(c(COPHEN,COPHEN2))
  }
```

With these instruccion we compute $E(D_n^2)$ with the formula given in Proposition 4.31.

```
indices.tb = list()
values.tb = c()
for(k in 3:50){
  trees = bin.tb.ape[bin.tb.n==k]
  indices.tb[[k]] = t(sapply(trees, compute.values))
  colnames(indices.tb[[k]])=c("Sackin","Cophenetic","overPhi2")
  values = colMeans(indices.tb[[k]])
  formula = ED2n.values(k,values[1],values[2],values[3])
  values.tb = rbind(values.tb,c(k,values,formula))
}
colnames(values.tb) = c("n","Sackin","Cophenetic","overPhi2","E(D2n)")
```

The results of this computations are avalaible in "C4-table-tb-values.txt".

The following code computes $E_Y(D_n^2)$ and $E_U(D_n^2)$ for $n = 3, \ldots, 50$:

```
range.plot = 3:50
eyD2n.values = sapply(range.plot, EYD2n)
euD2n.values = sapply(range.plot, EUD2n)
```

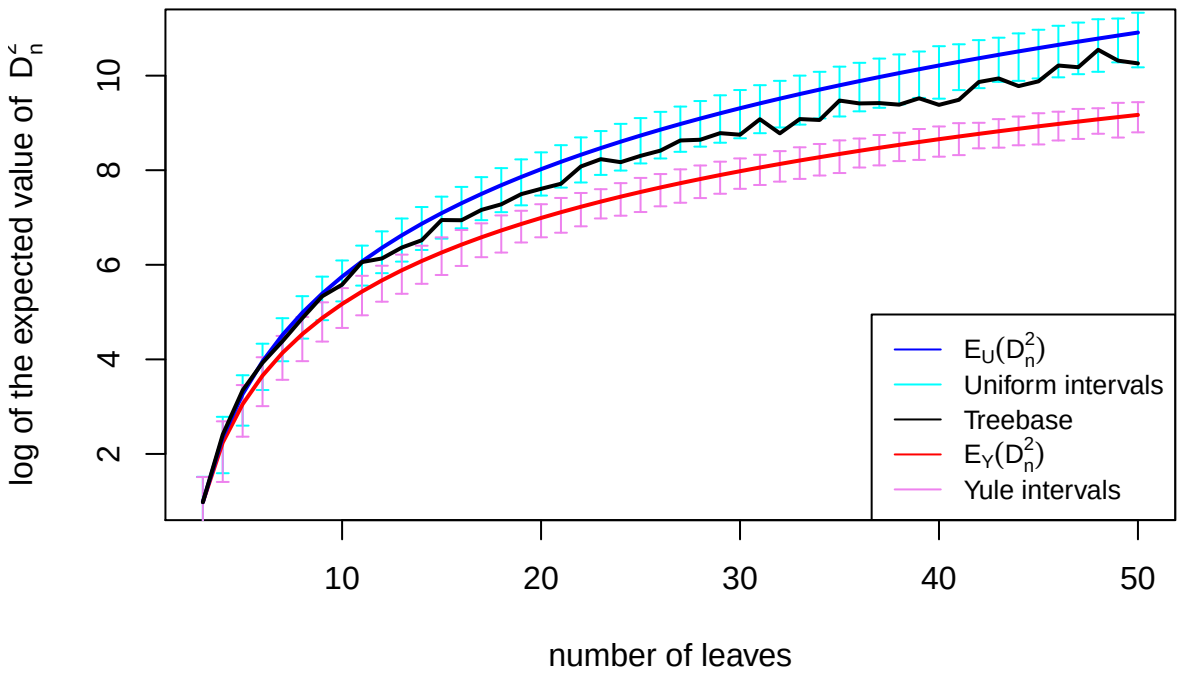
Using the computations of the variance of D_n^2 (from previous section) we can obtain the reference intervals for D_n^2 that will be drawn in the figure.

```
# harmonic -> Hn
# EYD2n -> EY2
# EUD2n -> EU2
#original: formula de EUD2n
# varU2 = function(n) return(varD2n[n-2,4]-EUD2n(n)^2)
# varY2 = function(n) return(varD2n[n-2,2]-EYD2n(n)^2)
# vu2 =  sapply(3:50,varU2)
# vy2 =  sapply(3:50,varY2)
# intU = cbind(3:50,log(euD2n.values-sqrt(vu2)),
#              log(euD2n.values+sqrt(vu2)))
# intY = cbind(3:50,log(eyD2n.values-sqrt(vy2)),
#              log(eyD2n.values+sqrt(vy2)))
#tabla: usa la simulacion de la varianza
#(varD2n) c("n","Yule_EDn4","Yule_varDn2","uniform_EDn4","uniform_varDn2")
# intU = cbind(range.plot,log(euD2n.values-1*sqrt(varD2n[range.plot-2,5])),
#                log(euD2n.values+1*sqrt(varD2n[range.plot-2,5])))
# # intU2 = cbind(range.plot,log(euD2n.values-2*sqrt(varD2n[range.plot-2,5])),
# #                log(euD2n.values+2*sqrt(varD2n[range.plot-2,5])))
# # intU2[c(1,2),2]=0;intU2[48,2]=8
# intY = cbind(range.plot,log(eyD2n.values-1*sqrt(varD2n[range.plot-2,3])),
#                log(eyD2n.values+1*sqrt(varD2n[range.plot-2,3])))
# intY2 = cbind(range.plot,log(eyD2n.values-2*sqrt(varD2n[range.plot-2,3])),
#                log(eyD2n.values+2*sqrt(varD2n[range.plot-2,3])))
# intY2[is.nan(intY2[,2]),2]=0
#
#
# varY.reg  = function(n) return(exp(-0.3085)*n^(4.1522))
# varU.reg  = function(n) return(exp(-4.556)*n^(6.3883))
# intU.reg1 = cbind(range.plot,log(euD2n.values-1*sqrt(sapply(range.plot,varU.reg))),log(euD2n.values+1
# intU.reg1[is.nan(intU.reg1[,2]),2]=0
# intY.reg1 = cbind(range.plot,log(eyD2n.values-1*sqrt(sapply(range.plot,varY.reg))),log(eyD2n.values+1
# intY.reg1[is.nan(intY.reg1[,2]),2]=0
#
# intU.reg2 = cbind(range.plot,log(euD2n.values-2*sqrt(sapply(range.plot,varU.reg))),log(euD2n.values+2
# intU.reg2[is.nan(intU.reg2[,2]),2]=0
# intY.reg2 = cbind(range.plot,log(eyD2n.values-2*sqrt(sapply(range.plot,varY.reg))),log(eyD2n.values+2
# intY.reg2[is.nan(intY.reg2[,2]),2]=0

intU = cbind(range.plot,log(euD2n.values-
               1*sqrt(varD2n[range.plot-2,4]-euD2n.values^2)),
               log(euD2n.values+
               1*sqrt(varD2n[range.plot-2,4]-euD2n.values^2)))
intY = cbind(range.plot,log(eyD2n.values-
               1*sqrt(varD2n[range.plot-2,2]-eyD2n.values^2)),
               log(eyD2n.values+
               1*sqrt(varD2n[range.plot-2,2]-eyD2n.values^2)))
```

```
#
##pint(intY,intU)
# pint(intY.reg1,intU.reg1)
# pdf("pruebasSigma.pdf")
# par(mfrow=c(2,2))
#   pint(intY1,intU1)
# pint(intY2,intU2)
#   pint(intY.reg1,intU.reg1)
# pint(intY.reg2,intU.reg2)
# dev.off()

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 ),int.yule[i-2,2:3],col="violet")
    lines(c(i-epsilon,i+epsilon),rep(int.yule[i-2,2],2),col="violet")
    lines(c(i-epsilon,i+epsilon),rep(int.yule[i-2,3],2),col="violet")
  }
}
```

Next figure plots $\log(E_{TrB}(D_n^2))$ as a function of n (middle, thick black curve). We have added the curves of $\log(E_Y(D_n^2))$ (lower, red curve) and $\log(E_U(D_n^2))$ (upper, blue curve), again as functions of n, and the logarithms of the corresponding reference intervals for D_n^2 (vertical segments).

```
plot(NULL,NULL,xlim=range(range.plot),ylim=c(1,11),
     xlab="number of leaves",ylab=expression("log of the expected value of  "*D[n]^2))
draw.intervals(range.plot,intY,intU)
lines(range.plot,log(eyD2n.values),col="red",lwd=2)
lines(range.plot,log(euD2n.values),col="blue",lwd=2)
lines(range.plot,log(values.tb[,5]),lwd=2)
legend("bottomright", legend=c(expression(E[U]*(D[n]^2)),"Uniform intervals","Treebase",
        expression(E[Y]*(D[n]^2)),"Yule intervals"),col=c( "blue","cyan","black","red",
        "violet"),lty=1,cex=0.8)
```



The graphic shows that the expected value of $d^2_{\varphi,2}$ on (the shapes of) the phylogenetic trees contained in TreeBASE is better explained by the uniform model than by the Yule model.