```
#.............Script général pour le calcul de l et m.......................

#Charger le package "ape" permettant de travailler avec des données
#phylogénétiques sur R.

library(ape)

#Charger le bon répertoire courant dans l'onglet fichier (soit le fichier
#où est enregistré le fichier texte contenant les données relatives à l'arbre
#phylogénétique étudié.

arbre<-read.tree("tree")

plot(arbre,main="Arbre phylogénétique")

#Commande donnant un vecteur t contenant la longeur de chaque branche de l'arbre
#(temps):

t<-branching.times(arbre)

#Généralement, à ce stade, on peut également entrer dans R la valeur de T (qui
#nous est en principe fournie avec l'arbre)

#Code pour la fonction de base; l=taux de spéciation, m=taux d'extinction et
#t=temps:

p<-function(l,m,t) {
num<-(l-m)^2*exp((l-m)*t)
den<- (l*exp((l-m)*t)-m)^2
return(num/den)}

#Code pour la fonction contenant les exceptions;
#-première exception: si l=m.
#-seconde exception: dans le cas où l=m et où l'on obtient NaN ou Infinite.

prob<-function(l,m,t) {
if(m==l) {
like<-log(1/((1+l*t)^2))}
else{ like<-log(p(l,m,t))}
if(is.nan(like)| is.infinite(like)) {return(-1000000000)}
else  { return(like)}
}

#Code pour la fonction complète; T=temps entre temps zéro et le premier ancêtre
#commun à toutes les espèces considérées dans l'arbre mais ne figurant pas sur
#celui-ci:

vrais<- function(l,m,t,T) {
a<- prob(l,m,T) + sum(sapply(t, prob, l=l, m=m)) + length(t)*log(l)
return(a)}
```

```
#Création d'une matrice contenant les valeurs de l et m que nous souhaitons tester.
#Il est important de noter que le seul rôle de cette matrice est de nous permettre
#de réaliser une représentation graphique des résultats que nous obtenons grâce
#à la fonction vrais, mais elle n'est pas nécessaire au bon fonctionnement des
#commandes permettant d'évaluer les valeurs de l et m de façon mathématiques (à
#savoir les commandes nlm, optimum et optimum2).

mat<-matrix(numeric(10000),ncol=100)
l<-seq(0.1,10,length.out=100)#la commande length.out=x stipule que la
m<-l#longeur de la séquence générée doit être de x

#Création d'une boucle afin de d'insérer la matrice dans la formule du maximum
#de vraissemblance:

for(i in 1:100) {
for(j in 1:100) {
mat[i,j]<-vrais(l[i], m[j], t, T)#Il faut avoir défini T (valeur fournie avec l'arbre)
}}

#Graphes:

contour(l,m,mat,xlab="l",ylab="m",main="Représentation en relief de la vraisemblance
des valeurs de l et m concernant arbre",col="blue")

persp(l,m,mat,main="Représentation tridimensionnelle de la vraisemblance
des valeurs de l et m concernant arbre",col="green",theta=30,phi=30)

#Commande permettant d'obtenir, au moyen d'un processus itératif (Non Linear Minimization)
#les valeurs les plus probables de l et m pour expliquer les valeurs de t et T que nous
#commaissions initialement:

vrais2<-function(x){
return(-vrais(x[1],x[2],t,T))}

#la fonction nlm() de R donne par défaut un minimum. Comme dans notre cas nous
#recherchon un maximum, il nous faut donc prendre le résultat opposé, c'est
#pourquoi on introduit un signe "-".

nlm(vrais2,c(d,e))

#d et e représentent des valeurs numériques approximatives obtenues sur la base
#du graphe contour() pour respectivement l et m les plus probables.

#Il est possible de compiler les fonctions vrais2 et nlm en une seule commande
#Comme d'habitude t est défini comme le vecteur des longueur de branches et T
#comme la durée entre le temps zéro et le premier ancêtre commun à toutes
#les espèces considérées dans l'arbre mais ne figurant pas sur celui-ci. Start est
#un vecteur de deux valeurs de type "c(x,y)" où x et y sont respectivement des
#estimations de l et m.
```

```
optimum<-function(time,T,start) {
vrais2<-function(x){
return(-vrais(x[1],x[2],time,T))}
return(nlm(vrais2,start))
}

#Afin d'obtenir les valeurs les plus probables de l et m, une autre commande
#que nlm est utilisable. Il s'agit de la commande "optim", une méthode
#d'optimisation non linéaire reposant sur l'algorithme de Nelder-Mead et qui
#permet de minimiser une fonction dans un espace à plusieurs dimensions.

optimum2<-function(time,T,start) {
vrais2<-function(x){
return(-vrais(x[1],x[2],time,T))}
return(optim(start,vrais2,method="L-BFGS-B",lower=c(0,0)))
}

#Comme nous l'avons mentionné, l'utilisation de cette fonction nous donne des
#valeurs pour les taux de spéciation et d'extinction qui sont biaisées par la
#variable T. La solution à ce problème est d'utiliser une formule du maximum
#de vraisemblance qui ne serait plus fonction de T. La fonction d'une telle
#formule nous a été fournie par Nicolas Salamin:

like<-function(times, T=1, like="given", algo="BFGS") {
if(class(times)=="phylo") times<-branching.times(times);

if(like=="data") {
  out<-optim(c(2, 1), function(p) -ldata(T, times, p[1], p[2]), method=algo);
  }
  else {
out<-optim(c(2, 1), function(p) -lgiven(times, p[1], p[2]), method=algo);
out$par<-sort(out$par, decreasing=TRUE);
}

print(list(speciation=out$par[1], extinction=out$par[2], lnL=out$value));
}

prob<-function(t, s, e) {
if(s==e) {
like<-log(1)-2*log(1+s*t);
}
else {
like<-log(((s-e)^2)*exp(s*t-e*t))-log((s*exp(s*t-e*t)-e)^2);
}

  if(is.nan(like) | is.infinite(like)) {
    return(-100000);
  }
  else {
  return(like);
  }
```

```
}

intprob<-function(t, s, e) {
tol<-0.000000001;

if(t>=250) {
if(s>e) {
if(s<tol) {
like<-log(1)-log(tol);
}
else {
like<-log(1)-log(s);
}
}
else {
if(e<tol) {
like<-log(1)-log(tol);
}
else {
like<-log(1)-log(e);
}
}
}
else {
if(s==e) {
like<-log(t)-log(s*t + 1);
}
else {
like<-log(exp(s*t-e*t)-1)-log(s*exp(s*t-e*t)-e);
}
}

return(like);
}

paradis<-function(t, s, e) {
  if(s<0 | e<0) return(-10000000);

par.times<-c(NA, t);
  ntax=length(t)+1;

return(-2*(sum(log((ntax-1):1))+(ntax-2)*log(s)+s*sum(par.times[3:ntax])+ntax*log(1-e)-2*sum(log(exp(s*par.times[2:ntax])-e))));
}

lgiven<-function(t, s, e) {
  if(s<0 | e<0) return(-10000000);

ntimes<-length(t);
T<-1000*max(t);

return(-ntimes*intprob(T, s, e)+sum(sapply(t, prob, s=s, e=e)));
```

```
}

ldata<-function(T, t, s, e) {
  if(s<0 | e<0) return(-10000000);

tol<-0.000000000000001;
ntimes<-length(t);

if(s<tol) {
return(ntimes*log(tol)+prob(tol, e, T)+sum(sapply(t, prob, s=tol, e=e)));
}

  return(ntimes*log(s)+prob(s, e, T)+sum(sapply(t, prob, s=s, e=e)));
}

estimate.lgiven<-function(trees) {
btimes<-lapply(trees, branching.times);
speciation<-vector(length=length(trees));
extinction<-vector(length=length(trees));
for(i in 1:length(trees)) {
out<-sort(optim(c(2,1), function(p) -lgiven(btimes[[i]], p[1], p[2]), upper=c(100,100), lower=c(0,0), method="L-BFGS-B")$par, decreasing=T
speciation[i]<-out[1];
extinction[i]<-out[2];
}

return(data.frame(speciation=speciation, extinction=extinction));
}

estimate.ldata<-function(trees, T) {
  btimes<-lapply(trees, branching.times);
speciation<-vector(length=length(trees));
extinction<-vector(length=length(trees));
for(i in 1:length(trees)) {
out<-sort(optim(c(2,1), function(p) -ldata(T, btimes[[i]], p[1], p[2]), upper=c(100,100), lower=c(0,0), method="L-BFGS-B")$par, decreasing
speciation[i]<-out[1];
extinction[i]<-out[2];
}

return(data.frame(speciation=speciation, extinction=extinction));
}

expectedTimeTransform<-function(T, s, e, n) {
  a<-0;
  b<-(exp(s*T-e*T)-1)/(s*exp(s*T-e*T)-e);
  lim<-(b-a)/n;
  y<-lim*(1:(n-1));
  return((1/(s-e))*log((1-e*y)/(1-s*y)));
}

calc.mat<-function(size, T, t) {
  mat<-matrix(numeric(size*size),nrow=size);
```

```
  l<-seq(0,10,length.out=size);
  m<-l;
  for(i in 1:size) {
    for(j in 1:size) {
      mat[i,j]<-ldata(T,t,l[i],m[j]);
    }
  }

  return(mat);
}

createTree<-function(t=1, l=2, m=1, spp=20, plot=FALSE) {
  if(l<m) {
    stop("Pas d'arbre possible. Le taux de spÃ©ciation est plus petit que le taux d'extinction.\n",call.=FALSE);
  }

  if(l==0) {
    stop("Pas d'arbre possible. Le taux de spÃ©ciation est de zÃ©ro.\n",call.=FALSE);
  }

  require(ape);
  tree<-rtree(spp);
  nonodes<-(spp*2)-1;
  times<-genTimes(t, l, m, spp-1);

  j<-2;
  set<-c(1:spp,(spp+2):nonodes);
  for(i in set) {
    k<-which(tree$edge[,2]==i);
    if(i>spp) {
      tree$edge.length[k]<-times[j];
      j<-j+1;
    }
    else {
      tree$edge.length[k]<-0;
    }
  }

  edge<-tree$edge.length;
  for(i in 1:(nonodes-1)) {
    j<-tree$edge[i,1]; #it's the ancestor, I need to find which line as j as descendant to get its branch length

    if(j==(spp+1)) { #if j is the root, its time is times[1]
      anc<-times[1];
    }
    else { #otherwise, search the corresponding line
      k<-which(tree$edge[,2]==j);
      anc<-edge[k];
    }

    tree$edge.length[i]<-anc-edge[i];
```

```
  }

  if(plot) {
    plot(tree);
    axisPhylo();
  }

  return(tree);
}

genTimes<-function(t=1, l=2, m=1, spp=20) {
  if(l==m) {
    l<-l+0.000000001;
  }

lt<-l*t;
mt<-m*t;

myexp<-exp(lt-mt);

a<-0;
b<-(myexp - 1)/(l*myexp - m);

times<-sort(runif(spp, a, b), decreasing=TRUE);

  return((1/(l-m))*log((1-m*times)/(1-l*times)));
}

plot2D<-function(t, l, m) {
  l<-m<-seq(0.000001, l, length=100);

  mat<-matrix(numeric(10000), nrow=100);

  for(i in 1:100) {
    for(j in (i+1):100) {
      mat[i,j]<-lgiven(t, l[i], m[j]);
    }
  }
}

#Ainsi, le seul paramètre à connaître pour utliser la fonction "like" est le
#vecteur des longueurs de branches t.

.....................PARENTHESE SUR LA GENERATION D'ARBRES.....................

#Génétation d'arbres pour lesquels on peut faire varier l, m, T ainsi que le nombre
#d'espèces afin de constater l'influence de ces paramètres sur les temps t (commande
#fournie par Nicolas Salamin):

createTree<-function(t=1, l=2, m=1, spp=20, plot=FALSE) {
  if(l<m) {
```

```
    stop("Pas d'arbre possible. Le taux de spéciation est plus petit que le taux d'extinction.\n",call.=FALSE);
  }

  if(l==0) {
    stop("Pas d'arbre possible. Le taux de spéciation est de zéro.\n",call.=FALSE);
  }

  require(ape);
  tree<-rtree(spp);
  nonodes<-(spp*2)-1;
  times<-genTimes(t, l, m, spp-1);

  j<-2;
  set<-c(1:spp,(spp+2):nonodes);
  for(i in set) {
    k<-which(tree$edge[,2]==i);
    if(i>spp) {
      tree$edge.length[k]<-times[j];
      j<-j+1;
    }
    else {
      tree$edge.length[k]<-0;
    }
  }

  edge<-tree$edge.length;
  for(i in 1:(nonodes-1)) {
    j<-tree$edge[i,1]; #it's the ancestor, I need to find which line as j as descendant to get its branch length

    if(j==(spp+1)) { #if j is the root, its time is times[1]
      anc<-times[1];
    }
    else { #otherwise, search the corresponding line
      k<-which(tree$edge[,2]==j);
      anc<-edge[k];
    }

    tree$edge.length[i]<-anc-edge[i];
  }

  if(plot) {
    plot(tree);
    axisPhylo();
  }

  return(list(tree=tree, T=times[1]));
}

genTimes<-function(t=1, l=2, m=1, spp=20) {
  if(l==m) {
    l<-l+0.000000001;
```

```
  }

lt<-l*t;
mt<-m*t;

myexp<-exp(lt-mt);

a<-0;
b<-(myexp - 1)/(l*myexp - m);

times<-sort(runif(spp, a, b), decreasing=TRUE);

  return((1/(l-m))*log((1-m*times)/(1-l*times)));
}

#Commande pour générer de tels arbres:

createTree(t=x, l=y, m=z, spp=w, plot=T)

#l=taux de spéciation, m=taux d'extinction, spp=nombre d'espèces
#observées au temps zéro, t=temps séparant le temps zéro du premier
#nœud ancestral n'étant pas compris dans l'arbre considéré (soit T)

#Commande pour faire des graphes ou des ananlyse à partir d'arbre générés
#par la fonction createTree:

arbre<-createTree(t=x, l=y, m=z, spp=w, plot=T)

T<-x

t<-branching.times(arbre$tree)

#...->suite de la procédure calssique avec les formules du maximimum de vraisemblance,
#la matrice et la boucle, puis les les graphes persp() et contour().

.....................Importance de la variable T.................................

#La fonction rep() utilise la fonction createTree() pour générer une série de n arbres
#(où n=nrep) ayant tous des paramètres identiques (l=l, m=m, t=T) mais que l'on peut faire
#varier. Ensuite, cette fonction récupère les longueurs de branches (vecteur t) de chacun
#des n arbres. Elle utilise alors la fonction optimum2() et la valeur de T que nous avons
#définie pour calculer les l et m les plus vraisemblables pour chacun de nos n arbres.
#Ces résultats sont alors stocké dans une matrice de nrep colonnes et de deux lignes
#(la première correspond aux valeurs de l et la seconde aux valeurs de m):

rep<-function(nreps=100,nsp=20,t=2,l=2,m=1) {
mat<-matrix(numeric(2*nreps),nrow=2,ncol=nreps);
for(i in 1:nreps) {
arbre<-createTree(t=t,l=l,m=m,spp=nsp);
a<-branching.times(arbre$tree);
param<-optimum2(a,t,c(l,m))$par
```

```
mat[,i]<-param;
}
return(mat);
}

#On peut facilement enregistrer les matrices générées par la fonction rep(). Cela
#permet alors de réaliser des représentations graphiques sous forme d'histogrammes
#des distributions de l et m relatives à des arbres pour lesquels nous avons défini
#les valeurs de T et du nombre d'espèces (nsp).

m<-rep(nreps=100,nsp=10,t=1)

par(mfrow=c(1,2))

hist(m[1,],xlab="lambda",main="Distribution de lambda lorsque
T=2, nsp=10, l=2 et m=1")#histogramme en fonction de l
hist(m[2,],xlab="mu",main="Distribution de mu lorsque
T=2, nsp=10, l=2 et m=1")#histogramme en fonction de m
```