SCRIPT FOR OU MODEL (Anna Kostikova and Berto Polerà)

```
#calculates lineage length from a given node DOWN towards the root of the tree
(usefull to calculate lengths for internal nodes and for non-ultrametric trees)
lineage_length<-function(i, chunk_full=0, tree){


        chunk<-tree$edge.length[which(tree$edge[,2]==i)] #get down first branch
        #print(chunk)
        if ( length(tree$edge.length[which(tree$edge[,2]==i)]) == 0){ #check if
it is ancestral state, if it is - return value and quite


                                return(chunk_full)
        }

        else{
                        chunk_full <- chunk_full + chunk # add to a chunk

                        parent<-tree$edge[tree$edge[,2]==i][1] #get parent
                        chunk_full<-lineage_length(parent, chunk_full, tree) #do
recursion to next level down

                        }

}



#not the quickest, but working solution to count number of branches for a
lineage
lineage_branch_num <-function(i,tree,chunk_full=0){



        if ( length(tree$edge.length[which(tree$edge[,2]==i)]) == 0){ #check if
it is ancestral state, if it is - return value and quite
                return(chunk_full)
        }

        else{
                        chunk_full <- chunk_full + 1 # add to a chunk
                        parent<-tree$edge[tree$edge[,2]==i][1] #get parent
                        chunk_full<-lineage_branch_num(parent, tree, chunk_full)
#do recursion to next level down

                        }

}

#get full tree length T
tree_length<-function(tree){

        any_extern_tip<-unique(tree$edge[,1])[1]-1
        T<-lineage_length(any_extern_tip,0,tree)
        return(T)

}
```

```
#calculate exponentials for VCV matrix - call to tree_length function to get T
make_exp<-function(a,tree,alfa){
        T<-tree_length(tree)

        if(a==1){
                varcov<-(1-exp(-2*alfa*T))
        }
        else{
                t1<-T-a
                varcov<-(exp(-2*alfa*t1))*(1-exp(-2*alfa*a))
        }

return(varcov)
}


#calculate full matrix of variance and covariance - call to "lineage length" and
"make_exp" functions
vcv_predicat<-function(tree,alfa){

        my.vcv<-mrca(tree) #construct MRCA matrix for the tree

        for (i in 1:nrow(my.vcv)){

                for (j in 1:ncol(my.vcv)){
                        shared_lineage<-lineage_length(my.vcv[i,j],0,tree) #for
each matrix cell get lineage length
                        calc_exp<-make_exp(shared_lineage,tree,alfa) #and
calculate exponentials
                        my.vcv[i,j]<-calc_exp
                }

        }
        return(my.vcv)
}



#return matrix for all species of lineage lengths (direction - downwards, to the
root)
internal_nodes_lineages<-function(tree){

internal_nodes<-sort(unique(tree$edge[,1])) #get unique  internal nodes and sort
them
len<-length(unique(tree$edge[,1])) #calculate how many of them are in the tree
my.mat<-matrix(nrow=len, ncol=2) #make matrix to store lineage lengths


for (i in 1:length(internal_nodes)){
        my.mat[i,1]<-internal_nodes[i]
        my.mat[i,2]<-lineage_length(internal_nodes[i],0,tree) #calculate lineage
length (call to lineage length function
}

return(my.mat)

}




#for a given lineage produce matrix with (1) branch length (2) regime (3) cumsum
of branch length _before_ given branch (necessary for OU model)
```

```
regime_length<-function(i,my.mat, tree, counter =1, dfn_tmp){


        if ( length(tree$edge.length[which(tree$edge[,2]==i)]) != 0){ #check if
it is ancestral state, if it is - return value and quite

                        my.mat[counter,1]<-
tree$edge.length[which(tree$edge[,2]==i)] #get down first branch
                        parent<-tree$edge[tree$edge[,2]==i][1] #get parent
                        my.mat[counter,2]<- dfn_tmp[,2]
[which( dfn_tmp[,1]==parent)]
                        counter = counter + 1
                        my.mat<-regime_length(parent, my.mat, tree, counter,
dfn_tmp) #do recursion to next level down

        }

        else{

                my.mat[counter,1]<-tree_length(tree)
                my.mat[counter,2]<-0
                if (counter > 2){
                my.mat[,3]<- c(0,cumsum(my.mat[,1])[1:(counter-2)],0) #making up
length before not occupies by a  regime
                }
                else{
                my.mat[,3]<- c(0,0) #this is mean that if branch starts from
root then it will only have two values in matrix - 1 and 0, therefore, before
history is 0 and root histoty 0
                }
                return(my.mat)
        }


}


#calculate exponentials for weight matrix
make_exp_weights<-function(a,tree,alfa){

        T<-tree_length(tree)


        if(a[2]==0){#when node is root
                varcov<-(exp(-alfa*T))
        }

        else{#all other nodes
                varcov<-(exp(-alfa*(a[3])))*(1-exp(-alfa*a[1]))

        }
return(varcov)
}




#find all extant nodes in the tree (by numbers, not species names!)
get_terminal_nodes<-function(tree){

index<-which(!(tree$edge[,2]) %in% unique(tree$edge[,1]))
terminals<-length(index)
for (i in 1:length(index)) {
        terminals[i]<-tree$edge[index[i],2]
```

```
}
return (terminals)
}

#construct a weight matrix for a particular number of selective regimes,
requires alfa, tree and number of selective regimes. Call to functions
get_terminal_nodes and regime_length
#regime dataframe must look as follow:
#   qq1
#t1   1
#t2   2
#t3   1
#t4   3
#where rownames - are species names, and only column is regime states

weigth_matrix<-function(regimes_df, tree, alfa, single_regime=TRUE){



termin_nodes<-get_terminal_nodes(tree)
species<-list()

if (single_regime) {
dfn_tmp<-ace_single_regime(regimes_df,tree) #_do not_ reconstruct ancestral
state, just assign 99 to them

}

else{
dfn_tmp<-ace_reconstruction(regimes_df,tree) #reconstruct species ancestral
states for each node

}

        for (i in 1:length(termin_nodes)){
                nbrlen<-lineage_branch_num(i,tree,0)
                my.mat<-matrix(nrow=(nbrlen+1),ncol=3)
                epoch<-regime_length(i,my.mat, tree, counter =1, dfn_tmp)
                epoch_exp<-apply(epoch, 1, FUN=make_exp_weights, tree=tree,
alfa=alfa)
                names(epoch_exp)<-epoch[,2]
                epoch_2_regimes<-aggregate(epoch_exp,
by=list(Category=names(epoch_exp)), FUN=sum)
                epoch_fin<-epoch_2_regimes[[2]]
                names(epoch_fin)<-epoch_2_regimes[[1]]
                species[[i]]<-epoch_fin

        }

        spec_df<-list_to_df(species)
        spec_df$check<-apply(spec_df,1,FUN=sum)
        return(spec_df)

}


#convert list with components of unequal length into a data frame and replaces
all NA with 0 value (so all species with do not have a particular regime on a
lineage would be equal to zero) _FINAL THING HERE
list_to_df<-function(myList){

dat <- data.frame()
for(i in seq(along=myList)) for(j in names(myList[[i]]))
```

```
dat[i,j] <- myList[[i]][j]
dat[is.na(dat)] <- 0
return(dat)

}

#reconstruct single regime for a tree
ace_single_regime <- function(regimes_input,tree){
#how many internal species
n_dead_taxa<-length(unique(tree$edge[,1]))
#how many extant species
n_extant_taxa<-length(tree$tip.label)

#resort input regimes into tree$tip.label order
reg_extant<-regimes_input[,1][match(tree$tip.label, rownames(regimes_input))]
#make names for extant species
names_extant<-c(1:length(tree$tip.label))

#create empty matrix to store results
dfn<-data.frame(matrix(nrow=n_dead_taxa+n_extant_taxa, ncol=2))

dfn[1:n_extant_taxa,1]<-names_extant
dfn[1:n_extant_taxa,2]<-reg_extant

node.label<-((length(tree$tip)+1):((length(tree$tip)*2)-1))
k = 1
for (l in  (n_extant_taxa+1):nrow(dfn)) {


        dfn[l,2]<-1
        dfn[l,1]<-node.label[k]
        k = k+1

}

colnames(dfn)<-c("node","regime")

return(dfn)


}


#reconstruct regimes given dataframe with regimes where rownames are same as in
tree$tip.label (order is not important)
ace_reconstruction<-function(regimes_input,tree){


#how many internal species
n_dead_taxa<-length(unique(tree$edge[,1]))
#how many extant species
n_extant_taxa<-length(tree$tip.label)

#resort input regimes into tree$tip.label order
reg_extant<-regimes_input[,1][match(tree$tip.label, rownames(regimes_input))]
#make names for extant species
names_extant<-c(1:length(tree$tip.label))

#create empty matrix to store results
dfn<-data.frame(matrix(nrow=n_dead_taxa+n_extant_taxa, ncol=2))

dfn[1:n_extant_taxa,1]<-names_extant
dfn[1:n_extant_taxa,2]<-reg_extant
```

```
#run ace to get ancestral states
test<-ace(reg_extant, tree, type="d", ip=0.00000001)
node.label<-((length(tree$tip)+1):((length(tree$tip)*2)-1))
k = 1
for (l in  (n_extant_taxa+1):nrow(dfn)) {

        index<-c(which(test$lik.anc[k,] == max(test$lik.anc[k,])))
        dfn[l,2]<-index[[1]]
        dfn[l,1]<-node.label[k]
        k = k+1

}

colnames(dfn)<-c("node","regime")

return(dfn)
}

#covert ancestral reconstuctions by node into by branch format
regimes_By_branch<-function(tree, ace_r){

        regime_by_branch<-ace_r[,2][match(tree$edge[,2], ace_r[,1])]
        regs<-cbind(tree$edge,regime_by_branch)

        return(regs)

}
```

#OU model - single regime

```
ou<-function(param, tree, trait, regime){

require(MASS)

sigma<-abs(param[1]) #sd
alfa<-abs(param[2])

n<-length(tree$tip.label)

#construct optima matrix (it is a SINGLE regime model, so matrix is same each
time)
opt<-matrix(nrow=2,ncol=1)

opt[1,1]<-abs(param[3])
opt[2,1]<-abs(param[4])

#construct weight matrix (it is a SINGLE regime model, so matrix is same each
time)
T<-tree_length(tree)
w1<-as.matrix(weigth_matrix(regime, tree, alfa, single_regime=TRUE)[,1:2])
#print(w1)


#construct VCV matrix
vcv <-vcv_predicat(tree,alfa)
#print(vcv)
s_vcv3<- (sigma^2)/(2*alfa) * vcv #multiple by sigma and alpha as in Butler and
King 2004
d_vcv3 <- ginv(s_vcv3)

(-t(trait-w1%*%opt)%*%d_vcv3%*%(trait-w1%*%opt))/2-n*log(2*pi)/2 -
log(abs(det(s_vcv3)))/2
```

```
}

##################################################### END OF FUNCTIONS

#generate dataset
tree<-rcoal(10)
tree<-chronopl(tree,lambda=0.1)
bio1<-rTraitCont(tree, model = "OU", sigma = 0.1, alpha = 0.5)


#fit with GEIGER
fitContinuous(tree, bio1, model="OU")

#make regimes
nsp<-length(tree$tip.label)
regimes<-data.frame(c(rep(1,nsp)))
rownames(regimes)<-tree$tip.label
colnames(regimes)<-c("reg")


#fit with OU code
optim(c(0.2,0.1,0.1,0.1), ou, tree=tree, trait=bio1, regime = regimes,
control=list(fnscale=-1), lower=c(0.000000001,0.000000001,-100,-100),
upper=c(100,100,100,100), method = "L-BFGS-B", hessian = TRUE)


#make 30 tests
res_brown<-matrix(0,nrow=30, ncol=2)
res_brown2<-matrix(0,nrow=30, ncol=2)
for (i in 1:30){
        print(paste("run: ",i,sep=""))
  bio1<-rTraitCont(tree, model = "OU", sigma = 0.1, alpha = 0.3)
  zx<-fitContinuous(tree, bio1, model="OU")
  res_brown[i,1]<-sqrt(zx$Trait1$beta)
  res_brown[i,2]<-zx$Trait1$alpha
  ze<-optim(c(0.1,0.1,0.1,0.1), ou, tree=tree, trait=bio1, regime = regimes,
control=list(fnscale=-1), lower=c(0.000000001,0.000000001,-100,-100),
upper=c(100,100,100,100), method = "L-BFGS-B", hessian = TRUE)
  res_brown2[i,1]<-ze$par[1]
  res_brown2[i,2]<-ze$par[2]
}









#testing weigth matrix
 weigth_matrix(sp2[1], utree2,0.2, single_regime = FALSE)
#0.2 represent alpha to test the matrix (in OU is estimated with MLE)




##TESTING
test1<-rcoal(10)
utest1<-chronopl(test1, lambda=0.1)
nsp<-length(utest1$tip.label)
```

```
regimes<-data.frame(c(rep(1,nsp)))
rownames(regimes)<-utest1$tip.label
colnames(regimes)<-c("reg")

bio1<-rTraitCont(utest1, model = "OU", sigma = 0.1, alpha = 0.2)
fitContinuous(utest1, bio1, model="OU", bounds=list(alpha=c(0.001, 10),
beta=c(0.000001, 100)))
#regimes<-data.frame((c(rep(1,10),rep(2,20))))
#regimes<-data.frame(c(rep(1,nsp/2),rep(2,(nsp-nsp/2))))


weigth_matrix(regimes, utest1, 0.3, single_regime=TRUE)
```