Trabalho Final

Função `bm.multiphylo`

```
bm.multiphylo <- function(trees, matrix, subsample = NULL, stype = NULL,
delta, criterion, Barplot = FALSE) {
  ## Testa se os pacotes requeridos estão instalados (código modificado de
https://stackoverflow.com/questions/9341635/check-for-installed-packages-bef
ore-running-install-packages)
  for (i in c("ape", "geiger")) { # Cria um ciclo que itera sobre o vetor
com os nomes dos pacotes
    if (i %in% installed.packages()) { # Procura cada um dos pacotes na
lista de pacotes instalados no computador
      cat (i, "is already installed in this computer\n") # Se encontrar o
pacote procurado, imprime na tela uma mensagen
    } else {
        stop (i, "is not installed\n") # Se não encontra o pacote, para e
imprime na tela uma mensagen
      }
  }
  ## Carrega os pacotes requeridos
  library(ape)
  library(geiger)
  ## Leia os dados morfológicos (o objeto `matrix`) e extrai o número de
dimensões
  if (ncol(matrix) > 2) { # Testa se o objeto `matrix` tem mais de dois
colunas
    stop ("The object `matrix` must contain only two columns, or you must
       to specify the column to be analyzed\n") # Se o objeto `matrix` tem
mais de dois colunas, para e imprime uma mensagen na tela
  } else {
      chtaxa <- as.character(matrix[ ,1]) # Cria um vetor com os nomes dos
táxons inclusos no objeto `matrix`
    }
  ## Leia o arquivo multiphylo contendo as árvores filogenéticas (o objeto
`trees`)
  if (class(trees) == "multiPhylo") { # Testa se o objeto `trees` é de clase
multiPhylo
    trtaxa <- trees$TipLabel$tip.label # Cria um vetor com os nomes dos
táxons inclusos no objeto `trees`
  } else {
      stop ("The object `trees` must be of class multiphylo, with a sample
of trees
            in parenthetical notation\n") # Para a função se o objeto
`trees` não é de clase multiPhylo
    }
  ## Testa se as árvores fologenéticas no objeto `trees` estão enraizadas
  if (min(is.rooted(trees)) == 0) { # Testa se as árvores filogenéticas
```

```
estão enraizadas
    stop ("The phylogenetic trees must be rooted\n") # Para a função se pelo
menos uma das árvores filogenéticas não está enraizada
  }
  ## Amostras de árvores filogenéticas
  if (!(is.null(subsample))) { # Testa se o usuário deu um valor para o arg
`subsample`
    if (subsample >= 2 & subsample < length(trees)) { # Testa se o valor do
argumento subsample está entre 2 e o número total de árvores
      stype <- match.arg(arg = stype, choices = c("random", "first",
"last")) # Estabelece os valores possíveis do argumento `stype`
      if (stype == "random") {
        trees <- sample(x = trees, size = subsample) # Se stype = "random" o
subsample é ao acaso
      }
      if (stype == "first") {
        trees <- trees[1:subsample] # Se stype = "first" o subsample extrai
as n primeiras árvores
      }
      if (stype == "last") {
        trees <- trees[(length(trees) - subsample):(length(trees) -1)] # Se
stype = "last" o subsample extrai as n últimas árvores
      }
    } else {
      stop ("subsample must be numeric between 2 and the total of trees in
the object trees -1 \n") # Se o valor do argumento `subsample` está errado
imprime na tela uma mensagem
    }
  }
  ## Compara os nomes dos táxons que estão na matriz morfológica (`matrix`)
e nas árvores filogenéticas (`trees`)
  if (sum(sort(chtaxa) != sort(trtaxa)) > 0){ # Testa se os táxons nos
objetos `matrix` e `trees` são iguais, para isso primeiro ordena os nomes
alfabeticamente
    cat ("The following taxa in `matrix` were not found in the object
`trees`.
         They will be removed for the analysis:\n",
         chtaxa[chtaxa %in% trtaxa == FALSE], "\n") # Imprime na tela os
nomes dos táxons que estão no objeto `matrix` mas não no objeto `trees`
    cat ("The following taxa in `trees` were not found in the object
`matrix`.
         They will be removed for the analysis:\n",
         trtaxa[trtaxa %in% chtaxa == FALSE], "\n") # Imprime na tela os
nomes dos táxons que estão no objeto `trees` mas não no objeto `matrix`
    ## Remove os táxons incompativeis da `matrix`
    chtaxa.remove <- chtaxa[chtaxa %in% trtaxa == FALSE] # Cria um vetor com
os nomes dos táxons que estão no objeto `matrix` mas não no objeto `trees`
    for (i in 1:length(chtaxa.remove)){ # Cria um contador desde 1 até o
número de elementos no vetor `chtaxa.remove`
      matrix <- matrix[-which(matrix[, 1] == chtaxa.remove[i]), ] # Remove
```

```
do objeto `matrix` cada um dos táxons que não estão no objeto `trees`
    }
    ## Remove os táxons incompativeis do objeto `trees` (Código modificado
de http://blog.phytools.org/)
    trees <- lapply (X = trees, FUN = drop.tip, tip = c(trtaxa[trtaxa %in%
chtaxa == FALSE])) # Aplica a função `drop.tip` sobre cada uma das árvores
do objeto `trees`
    class(trees) <- "multiPhylo" # Muda a clase do novo objeto `trees` de
lista para multiphylo
    cat ("trees:", length(row.names(summary(trees))), "phylogenetic trees
with",
         length(trees[[1]]$tip.label), "taxa\n") # Imprime na tela o número
de árvores no objeto `trees` e o novo número de táxons que contem cada uma
das árvores
  } else {
      ## Sumário dos dados que serão analisados
      # Árvores filogenéticas
      cat ("trees:", length(row.names(summary(trees))), "phylogenetic trees
with",
           length(trees$TipLabel$tip.label), "taxa\n") # Imprime na tela o
número de árvores no objeto `trees` e o número de táxons que contem cada uma
das árvores
    }
  ## Sumário dos dados que serão analisados
  # Matriz de dados morfológicos
  matrix[ ,2] <- as.factor(matrix[ ,2]) # Muda a clase do carater analisado
  cat ("matrix: 1 character and", dim(matrix)[[1]], "coded taxa\n") #
  ch <- setNames(object = matrix[ , 2], nm = matrix[ , 1]) # Cria um objeto
contendo só os estados do carater
  cat ("The character has the following states:\n", levels(ch), "\n") #
Extrai a lista de estados do carater e os imprime na tela
  ## Use a função `fitDiscrete` para testar o melhor modelo de evolução de
caracteres para cada uma das árvores do objecto `trees`
  models <- c("ER", "ARD", "SYM") # Cria um objeto com os nomes dos modelos
de evolução que vão ser testados
  fitmodels <- data.frame(Tree = rep(1:length(trees), each = 3), # Cria um
data frame para inserir o resultado que sai do for
                          Model = rep(c("ER", "ARD", "SYM"), times =
length(trees)),
                          AIC = rep(NA, times = (length(trees))*3),
                          AICc = rep(NA, times = (length(trees))*3))
  iter <- 1 # Cria um objeto que aumenta em 1 com cada ciclo, permite
inserir resultados nas colunas do data frame
  for (i in 1:length(trees)) { # Cria um contador de 1 até o número de
árvores no objeto `trees`
    cat("Running", i, "trees out of", length(trees),"\n") # Imprime na tela
uma mensagem com o número de ciclos por vez
    for(j in models) { # Cria um contador que itera sobre o vetor dos
modelos
      tree <- trees[[i]] # Seleciona uma árvore do objeto `trees`
      testfit <- fitDiscrete(phy = tree, dat = ch, model = j) # Aplica a
```

```
função `fitDiscrete` sobre a árvore selecionada
      fitmodels[iter, "AIC"] <- testfit$opt$aic # Extrai o valor de AIC do
objeto testfit e o salva no data frame
      fitmodels[iter, "AICc"] <- testfit$opt$aicc # Extrai o valor de AICc
do objeto testfit e o salva no data frame
      iter <- iter + 1 # Aumenta em 1 o valor do objeto `iter`. Permite
inserir resultados na próxima linha do data frame
    }
  }
  ## Calcula a diferença (delta) entre os modelos de evolução
  minAIC <- aggregate(fitmodels$AIC ~ fitmodels$Tree, FUN = min) # Extrai o
mínimo valor de AIC para cada árvore
  minAIC <- rep(minAIC$`fitmodels$AIC`, each = 3) # Cria um vetor repitindo
mínimo valor de AIC para cada árvore
  minAICc <- aggregate(fitmodels$AICc ~ fitmodels$Tree, FUN = min) # Extrai
o mínimo valor de AICc para cada árvore
  minAICc <- rep(minAICc$`fitmodels$AICc`, each = 3) # Cria um vetor
repitindo mínimo valor de AICc para cada árvore
  fitmodels$delta_AIC <- fitmodels$AIC - minAIC # Calcula a diferença
(delta) entre o menor valor de AIC e os restantes; salva o resultado numa
nova coluna do data frame `firmodels`
  fitmodels$delta_AICc <- fitmodels$AICc - minAICc # Calcula a diferença
(delta) entre o menor valor de AICc e os restantes; salva o resultado numa
nova coluna do data frame `firmodels`
  ## Seleciona o melhor modelo usando os criterios AIC e AICc
  bestModel_AIC <- fitmodels[which(fitmodels$delta_AIC == 0), c("Tree",
"Model")] # Extrai o nome do modelo com o valor mais baixo de AIC (o melhor
modelo) para cada árvore
  bestModel_AICc <- fitmodels[(which(fitmodels$delta_AICc == 0)), c("Tree",
"Model")] # Extrai o nome do modelo com o valor mais baixo de AIC (o melhor
modelo) para cada árvore
  ## Calcula os "Akaike weights" para cada um dos modelos usando o valor de
AICc (Uma pior versão da função aic.w de `phytools`)
  fitmodels$Tree <- as.factor(fitmodels$Tree) # Muda a clase da coluna Tree
do data frame `fitmodels`
  b <- rep(NA, times = length(levels(fitmodels$Tree))*3) # Cria um vetor
para salvar o resultado do for
  iter = 1 # # Cria um objeto que aumenta em 1 com cada ciclo, permite
inserir resultados no vetor `b`
  for (i in levels(fitmodels$Tree)) { # Cria um contador que itera sobre o
número de árvores
    for (j in models) { # Cria um contador que itera sobre os modelos
      a <- fitmodels[which(fitmodels$Tree == i & fitmodels$Model == j),
"AICc"] # Seleciona o valor de AICc para cada árvore e cada modelo, e o
salva no objeto `a`
      b[iter] <- (exp(1))^(- a/2) # Calcula uma parte da expresão matemática
que calcula os pesos dos modelos (ver documentação)
      iter = iter + 1 # Aumenta em 1 o valor do objeto `iter`. Permite
inserir resultados no vetor `b`
    }
```

```
  }

  fitmodels$b <- b # Cria uma nova coluna no data frame `fitmodels` para
salvar o conteudo do vetor `b`
  sumation <- aggregate(x = fitmodels$b, by = list(Tree = fitmodels$Tree),
FUN = sum) # Soma os valores de `b` para cada árvore
  sumation <- rep(sumation$x, each = 3) # Repite os resultados da soma
anterior para criar um vetor do mesmo comprimento que as colunas no data
frame `fitmodels`
  fitmodels$AICcw <- b / sumation # Calcula os pesos dos modelos (ver
documentação) e salva-los numa nova coluna do data frame `fitmodels`
  ## Seleciona o melhor modelo usando o criterio AICcw
  max_AICcw <- aggregate(x = fitmodels$AICcw, by = list(Tree =
fitmodels$Tree), FUN = max) # Extrai o maior valor de AICcw (o melhor
modelo) para cada árvore
  bestModel_AICcw <- fitmodels[which(fitmodels$AICcw %in% max_AICcw$x), 1:2]
# Extrai o nome do modelo com o maior valor AICcw (o melhor modelo) para
cada árvore
  ## Data frame com o melhor modelo selecionado para cada árvore
  bestModels <- data.frame(Tree = levels(fitmodels$Tree), AIC =
bestModel_AIC$Model, # Cria um data frame com os nomes dos melhores modelos
para cada árvore de acordo a cada um dos criterios
                           AICc = bestModel_AICc$Model, AICcw =
bestModel_AICcw$Model)
  ## Re-organiza o data frame fitmodels
  fitmodels <- data.frame(Tree = fitmodels$Tree, Model = fitmodels$Model,
AIC = fitmodels$AIC, # Reordena as colunas do data frame fitmodels
                          AICc = fitmodels$AICc, AICcw = fitmodels$AICcw,
delta_AIC = fitmodels$delta_AIC,
                          delta_AICc = fitmodels$delta_AICc)
  ## Compara os valores de AIC e AICc entre modelos e seleciona um modelo de
acordo com o valor do arg `delta`
  if (class(delta) == "numeric") { # Testa se a clase do objeto `delta` é
numeric
    deltaAIC <- fitmodels[which(fitmodels$delta_AIC <= delta), c(1,2,6)] #
Seleciona o modelo que tem um valor de delta menor ou igual ao valor de
delta_AIC inserido pelo usuário
    deltaAICc <- fitmodels[which(fitmodels$delta_AICc <= delta), c(1,2,7)] #
Seleciona o modelo que tem um valor de delta menor ou igual ao valor de
delta_AICc inserido pelo usuário
    ## Troca o nome de cada modelo por o número relativo de parámetros de
cada um deles para AIC
    deltaAIC$param <- rep(NA, times = length(deltaAIC$Model)) # Cria uma
nova coluna no data frame `deltaAIC` para salvar os resultados do for
    models <- c("ER", "SYM", "ARD") # # Cria um objeto com os nomes dos
modelos de evolução ordenados pela sua complexidade
    for (i in 1:3) { # Cria um contador de 1 até o número de modelos
      deltaAIC$param[which(deltaAIC$Model == models[i])] = i # Escreve 1, 2
ou 3 na nova coluna do `deltaAIC` para cada um dos modelos de acordo com sua
complexidade
    }
```

```
    min_AICparam <- aggregate(x = deltaAIC$param, by = list(Tree =
deltaAIC$Tree), FUN = min) # Extrai o menor valor de parámetros (o modelo
mais ssimples) para cada árvore
    selectModel_AIC <- deltaAIC[which(deltaAIC$param %in% min_AICparam$x),
c("Tree", "Model")] # Extrai o nome do modelo mais simples de acorco com o
AIC para cada árvore
    ## Troca o nome de cada modelo por o número relativo de parámetros de
cada um deles para AICc
    deltaAICc$param <- rep(NA, times = length(deltaAICc$Model)) # Cria uma
nova coluna no data frame `deltaAIC` para salvar os resultados do for
    for (i in 1:3) { # Cria um contador de 1 até o número de modelos
      deltaAICc$param[which(deltaAICc$Model == models[i])] = i # Escreve 1,
2 ou 3 na nova coluna do `deltaAIC` para cada um dos modelos de acordo com
sua complexidade
    }
    min_AICcparam <- aggregate(x = deltaAICc$param, by = list(Tree =
deltaAICc$Tree), FUN = min) # Extrai o menor valor de parámetros (o modelo
mais ssimples) para cada árvore
    selectModel_AICc <- deltaAICc[which(deltaAICc$param %in%
min_AICcparam$x), c("Tree", "Model")] # Extrai o nome do modelo mais simples
de acorco com o AICc para cada árvore
  } else {
      stop ("arg `delta` must be numeric") # Se o valor do argumento `delta`
inserido pelo usuário não é numérico, para a função e imprime uma mensagem
na tela
    }
  ## Sumárico gráfico
  if (Barplot == TRUE) { # Testa se o usuário quer gerar uma saída gráfica
    bestModels$Tree <- as.factor(bestModels$Tree) # Confere que a coluna
Tree seja de clase factor
    b <- c(as.character(bestModels$AIC), as.character(bestModels$AICc),
as.character(bestModels$AICcw)) # Cria um vetor com o melhor modelo para
cada árvore e cada criterio
    c <- data.frame(criterion = rep(c("AIC", "AICc", "AICcw"), each =
length(levels(bestModels$Tree))),
                    models = b)
    freqModels <- table(c) # Cria uma tabela para contar as veces que cada
modelo foi o melhor modelo ajustado para a amostra de árvores
    x11() # Abre um dispositivo gráfico
    par(mfrow = c(2,1), mar = c(5, 6, 4, 4))
    color.names = c("black","grey40","grey80") # Estabelece as cores de cada
categoría para o gráfico
    barplot(t(freqModels), beside = TRUE, ylim = c(0, max(freqModels)+1),
xlab = "Akaike criteria for model selection",
            ylab = "Model Frequency", col = color.names, axis.lty = "solid",
main = "Best Model using Akaike criteria") # Faz um gráfico de barplot para
resumir a contagem dos modelos
    legend("topright", legend = rownames(t(freqModels)), fill = color.names,
title = "Models", cex = 0.5) # Insere uma legenda
    criterion <- match.arg(arg = criterion, choices = c("AIC", "AICc",
```

```
"AICcw")) # Estabelece os valores possíveis do argumento `criterion`
    if (criterion == "AIC") {
      freqModels2 <- table(selectModel_AIC$Model)
    }
    if (criterion == "AICc") {
      freqModels2 <- table(selectModel_AICc$Model)
    }
    if (criterion == "AICcw") {
      freqModels2 <- table(bestModel_AICcw$Model)
    }
    barplot(freqModels2, beside = TRUE, ylim = c(0, max(freqModels2)+1),
main = "Selected Model" , xlab = "Evaluated model for character evolution",
            ylab = "Model Frequency", col = color.names, axis.lty = "solid")
# Faz um gráfico de barplot para resumir a contagem dos modelos
  }

  ## Saída
  final_AIC <- list(matrix = matrix, trees = trees, fitModels = fitmodels,
bestModels = bestModels,
                    selectModel = selectModel_AIC) # Cria  uma lista com os
elementos que serão retornados pela função, caso o usuário use `criterion` =
AIC
  final_AICc <- list(matrix = matrix, trees = trees, fitModels = fitmodels,
bestModels = bestModels,
                     selectModel = selectModel_AICc) # Cria  uma lista com
os elementos que serão retornados pela função, caso o usuário use
`criterion` = AICc
  final_AICcw <- list(matrix = matrix, trees = trees, fitmodels = fitmodels,
bestModels = bestModels) # Cria  uma lista com os elementos que serão
retornados pela função, caso o usuário use `criterion` = AICcw
  if (criterion == "AIC") return(final_AIC) # Para a função e retorna uma
das lista já ditas
  if (criterion == "AICc") return(final_AICc) # Para a função e retorna uma
das lista já ditas
  if (criterion == "AICcw") return(final_AICcw) # Para a função e retorna
uma das lista já ditas
}
```

Função function.r

Documentação documentation.txt

Dados morfológicos morphodata.csv

Árvores filogenéticas (mudar extensão a .tre) multiphylo.txt