Thiago Macek Gonçalves Zahn

Mestrando em Evolução Morfológica, Laboratório de Evolução de Mamíferos (LEM), IB-USP.
Trabalhando no projeto “Integração morfológica no crânio e evolução da morfologia craniana em Feliformia”.

Exercícios

exec

Trabalho Final

Proposta A

Lines of Least Evolutionary Resistance (Schluter, 1996, Marroig et al 2005, Marroig et al 2010)

Uma possibilidade interessante para o trabalho final seria gerar uma função que pudesse ser útil para as análises que serão feitas futuramente em meu projeto de mestrado. Algumas das análises que posso vir a realizar estão relacionadas às linhas de menor resistência evolutiva, incluindo quanto estas são conservadas ao longo da filogenia de um grupo, quanto a evolução de um grupo segue estas linhas e quanto isso afeta a velocidade de resposta evolutiva.

As linhas de menor resistência evolutiva (LLER) para um determinado táxon, considerando um dado conjunto de caráteres, se define como a combinação linear destes caráteres que tem maior variância genética (/fenotípica) em uma população. Devido às restrições evolutivas decorrentes da estrutura de covariação entre caráteres, é comum que as alterações que ocorrem no curso da evolução se deem ao longo da direção desta “linha de menor resistência”.

O objetivo desta função seria obter como resposta:

1) Para todos os nós e terminais da filogenia:

- a Linha de Menor Resistência Evolutiva (LLER, correspondente ao primeiro componente principal da matriz de variação/covariação de caracteres de um terminal ou nó);

2) Para todos os ramos da filogenia (i.e. entre cada terminal/nó e seu ancestral)

- os vetores de resposta evolutiva (deltaZ, correspondente à diferença nas médias de cada caráter entre um terminal/nó e seu nó ancestral);
- a “respondabilidade” (quantidade total de diferenciação nos caráteres entre um terminal/nó e seu nó ancestral);
- a “velocidade de evolução” (respondabilidade dividida pelo tamanho do ramo correspondente na filogenia), caso sejam fornecidos tamanhos de ramo na filogenia;

- a correlação vetorial (cosseno do ângulo) entre as LLERs do nó/terminal e de seu ancestral;
- a correlação vetorial (cosseno do ângulo) entre a LLER do ancestral e o vetor de resposta evolutiva;
- a correlação entre [a respondabilidade] e [a correlação vetorial entre LLER do ancestral e vetor de resposta evolutiva];
- a correlação entre [a velocidade de evolução] e [a correlação vetorial entre LLER do ancestral e vetor de resposta evolutiva], caso sejam fornecidos tamanhos de ramo na filogenia.

Os argumentos (objetos) de entrada necessários para conseguir essas informações como resposta seriam:

- Uma filogenia (objeto da classe (phylo)), com ou sem tamanhos de ramo (o que afetaria a possibilidade de obter algumas das respostas, como mencionado acima);
- Uma lista com as matrizes de variação/covariação dos táxons terminais da filogenia, nomeados de forma a que haja correspondência entre cada matriz e sua posição na filogenia;
- Uma lista com os valores médios de cada caráter para os táxons terminais da filogenia, também nomeados de acordo com a filogenia.

Seguem os passos que imagino que seriam necessários:

1. Obter as matrizes de Var/Covar “pooled” para todos os nós internos da filogenia (utilizando médias ponderadas das matrizes) com base na estrutura da filogenia;
2. Obter os valores médios de cada caráter para todos os nós internos da filogenia (provavelmente utilizando também médias ponderadas, apesar de aparentemente haver outras formas) com base na estrutura da filogenia. Tenho a impressão de que já há funções de reconstrução de caráter ancestral que fazem isso;
3. Calcular a LLER para cada nó e terminal a partir das matrizes var/Covar geradas;
4. Calcular vetores de resposta evolutiva a partir dos valores médios gerados;
5. Calcular as correlações vetoriais entre LLERs e entre LLERs e vetores resposta;
6. Calcular as correlações de pearson entre as correlações vetoriais obtidas e respondabilidade/velocidade de evolução.

O output seria, provavelmente, uma lista contendo listas (!!) das informações desejadas para cada um dos nós - 'nomeados' ou 'indexados', de preferência, de forma que seja mantida a correspondência com a filogenia.

Seria bastante útil, mas me parece uma quantidade bastante grande de trabalho. Será que é viável??

Marroig, M.; Cheverud, J. 2005. Size as a line of least evolutionary resistance: diet and adaptive morphological radiation in New World monkeys. Evolution 59(5): 1128-1142

Marroig, M; Cheverud, J. 2010. Size as a line of least evolutionary resistance II: direct selection on size or correlated response due to constraints?

Schluter, D. 1997. Adaptive radiation along genetic lines of least resistance. Evolution 50(5): 1766-1774

Proposta B - Concluída!

Scales

Andei vendo o plano B da Daniela Rossoni, que fez a disciplina há alguns anos e acabou desenvolvendo o outro plano. Conversando com ela sobre a ideia de uma representação matricial no R das notas no braço de um violão (ou outro instrumento de corda) , me interessei pela ideia de, usando esse tipo de representação, gerar uma função que tivesse como resposta disposições de notas de diversas escalas musicais no braço, dadas uma série de condições.

Imaginei incluir no programa as escalas musicais mais conhecidas (modais, tonais menor harmônica e melódica, pentatônicas maior e menor, escala de tons inteiros, cromática) e possivelmente algumas outras menos conhecidas e/ou regionais.

Como entrada para a função imaginei:

- Uma nota inicial/tônica (A,B,C,D,E,F,G ou um dos sustenidos/bemois);
- Um conjunto de notas que deve constar na escala (um vetor); ou
- Um conjunto de notas que NÃO deve constar na escala (a resposta sendo todas as escalas que NÃO incluem aquelas notas);
- Uma região específica do braço do violão (ex. trastes 5 a 10) - caso este argumento seja dado, seriam exibidas como resposta só as disposições de notas, para cada escala, que se encaixam na região delimitada do braço;
- Um conjunto de 'posições específicas no braço'
- Um ou mais tipos de escala específicos (a função retornaria (ou NÃO retornaria)somente aquela(s) escala(s))
- Um ou mais argumentos relacionados ao tipo de escala (ex. “menor”, “maior”, “com 5a aumentada”, talvez outros).

A resposta, como mencionei acima, seria uma lista de matrizes seis (cordas) por xxx (trastes) apresentando as disposições de notas das escalas que respondem aos critérios definidos.

Outro aspecto que imaginei ser possível integrar na função seria uma generalização para 'qualquer instrumento de corda' em vez de só violões. Creio que seja possível fazer isso argumentos para definir o número de cordas, o número de trastes e a afinação de cada corda.

Essa proposta me parece mais... diferente do usual que a anterior. Procurei por pacotes, funções ou classes de objetos relacionados a música no R e não encontrei nada, o que faz parecer interessante tentar algo do tipo. Imagino que dará certo trabalho na hora de codificar as escalas, mas ainda imagino que seja possivelmente mais fácil que a proposta A!

Comentarios

Ambas as propostas representam otimos desafios. Gosta da primeira e parece que ainda nao foi implementada no R (acho!). De qq forma, fica a seu criterio A ou B! — *Alexandre Adalardo de Oliveira 2013/03/25 12:11*

Página de Ajuda / Help

```
scales                package:none              R Documentation

Scales fitting given criteria for a stringed instrument. Position of notes
on a stringed instrument.

Description:

scales produces musical scales for a chosen tonic which fit certain criteria
(notes to include, notes to exclude and type of scale), creates a matrix
indicating the position of notes on the arm of a stringed instrument based
on entry values for number of strings, number of frets and tuning, and
returns a list of matrices with the positions of the notes on the arm
created for each scale fitting the criteria. The resulting matrices
represent tablatures for the scales.
The user may choose to obtain note positions on the fretboard of an
instrument with given numbers of strings and frets and a given tuning
instead.
The function also produces a .txt file with the results in your working
directory. This file is called "scales.txt" if the function is used to
obtain tablatures for scales, or "instrument.txt" if the user chooses to
obtain note positions on the fretboard of an instrument.

Usage:

scales(tonic=1,notes=tonic,except=F,region=c(0,frets),types="all",strings=6,
tune=c(8,3,11,6,1,8),frets=15,grades=T)

Arguments:

tonic   numeric or character.               The tonic (first degree) of the
scales to be returned by the function. May be given as a character
representing the note in letter notation, using '#' for sharp and 'b' for
flat (e.g "A", "F#" or "Gb") or as a number corresponding to a note, as
follows: A=1, A# (or Bb)=2, B (or Cb)=3, C (or B#)=4, C# (or Db)=5, D=6, D#
(or Eb)=7, E (or Fb)=8, F (or E#)=9, F# (or Gb)=10, G=11, G# (or Ab)=12.
                            If you want to obtain note positions on the
fretboard of an instrument only, use the (numeric) value 0 for tonic.
                            The default value is '1', which returns scales
in 'A' ('la').
notes   numeric or character.               Notes which MUST be included in
the scales to be returned by the function. The tonic is obviously always
included in the scales to be returned. May be given as a vector of type
character, representing the notes in letter notation using '#' for sharp and
'b' for flat (e.g. c("B","F")) or as a vector of numbers corresponding to
the notes, as explained in the argument 'tonic', above.
                            The default is to include only the tonic.
```

except numeric, character or FALSE Notes NOT to be included in the scales to be returned by the function. May be given as a vector of type character, representing the notes in letter notation, using '#' for sharp and 'b' for flat (e.g. c("B", "C")) or as a vector of numbers corresponding to the notes, as explained in the argument 'tonic', above.
If except=FALSE, all scales with the given 'tonic' including the notes given in 'notes' will be returned. This is the default for the function.

region numeric vector of length=2 The region of the fretboard in which the scales returned are shown, given as a numeric vector with 2 elements corresponding to the first and last frets of the fretboard which should be shown (0 corresponds to open strings). Useful for users who wish to view possible ways to play scales on a particular region of an instrument.
If the second element in this argument is larger than 'frets', the function will return an error message.

types character Specific scales ("scale types") to be returned by the function. Given as a character vector with the names of all types of scales to be returned. The scales currently included in the function (and which can therefore be included in vectors for this argument) and their names for entry in this argument are:
Greek modes: "ionian" (=major tonal scale), "dorian", "phrygian", "lydian", "mixolydian", "eolian" (=natural minor tonal scale), "locrian";
Other tonal scales: "harmonic", "melodic" (corresponds to ascending melodic minor scale, as the descending melodic minor is equivalent to the eolian mode);
Pentatonic scales: "minorpentatonic", "majorpentatonic";
Other heptatonic scales: "altered", "flamenco", "doubleharmonic"
Hexatonic scales: "augmented", "blues" (corresponds to minor pentatonic with a 'blue note' on the #4th degree), "wholetone"
Chromatic scale: "chromatic"
If types="all", all scales with the given 'tonic' fitting the conditions in the 'notes' and 'exclude' arguments are returned. This is the default for this argument.

strings numeric The number of strings on the instrument for which the result should be shown. The default is 6 strings, as in a guitar (which is used as the 'default instrument' by this function).

tune numeric or character vector of length=strings The tuning to be used for each of the strings on the instrument for which the result should be shown, in order from the first string (i.e. highest pitched or thinnest) to the last string (i.e. lowest pitched or thickest). The tuning may be given as a vector of type character, representing the notes in letter notation using '#' for sharp and 'b' for flat (e.g.

c("E","B","G","D","A","E")) or as a vector of numbers corresponding to the notes, as explained in the argument 'tonic', above.

frets numeric The number of frets on the instrument for which the result should be shown, or the maximum number of frets to be shown in the results. The default value is 15 frets.
Pay attention to the fact that too large a number of frets will cause the resulting matrices not to be shown in a single line (both in the output window and in the .txt files written), which may hamper in reading the results.

degrees logical If TRUE, the scales returned will show the corresponding Scale Degree ("I"=tonic, "II"=supertonic, "III"=mediant, "IV"=subdominant, "V"=dominant, "VI"=submediant, "VII"=leading tone/subtonic) at the positions corresponding to each note in the scale. This is the default for the function.
If FALSE, the scales returned will simply show an "X" in all positions for notes included in the scale.

Details:

The numerical form of entry for the arguments 'tonic', 'notes', 'except' and 'tune' exist mostly as a result of the inner workings of the function. It is recommended (and much more intuitive) to use the character form to input these arguments.
For the matrices showing the scales to be clearly arranged and easily visible (both in the output panel and in the .txt files produced), it is recommended that the R console be expanded to an adequate width prior to running this function.
The "scales.txt" (for scale tablatures) or "instrument.txt" (for note positions on the fretboard of an instrument) files which are produced when running this function are always created with these names, so if you run the function a second time without moving or renaming the file created on the first run, the file produced during the second run will overwrite the file from the first run.

Value:

If 'tonic' is different from 0, scales returns a list with a number of components of class=matrix which corresponds to the number of scales which fit the criteria entered in the arguments 'notes', 'except' and 'types', to a maximum of 18 (the total number of scales currently included in the function).
The name of each component corresponds to the name of the scale it represents, as explained in the argument 'types', above.
If 'tonic'=0, scales returns a single matrix, corresponding to the positions of each note in the fretboard of an instrument with number of strings and frets and tuning as given in the arguments 'strings', 'frets' and 'tune'.
Each matrix in the list (for 'tonic' different from 0) or the single

```
returned matrix (for 'tonic'=0) has nrow=strings, ncol=frets+1 (with open
strings shown as "fret 0"). Row names correspond to the notes for each
string (i.e. argument 'tune'), and column names correspond to the number of
each fret ("fr 0", "fr 1", "fr n").

Warning:

Remember to use 'tonic=0' if you want to obtain just note positions in the
fretboard.
If the user enters 'tonic=0', the function ignores all arguments other than
'strings', 'tune' and 'frets'.
Do not run this function while keeping console width too small. This will
cause the matrices for scales in both the output window and the produced
.txt file to be broken and shown in two or more rows, which hampers reading.
Remember to rename or move the .txt file produced after each run from the
working directory before running this function again, as the function will
overwrite files with the names "scales.txt" or "instrument.txt" after each
run.
The function will return error messages if the user tries to input values
not identified by the function in the arguments 'tonic', 'notes', 'except'
or 'tune'.
An error message will also be returned if no scales currently included in
the function fit the conditions set by the user in the arguments 'notes',
'except' and/or 'types'.


Author(s):

Thiago Macek Gonçalves Zahn

thimacek@gmail.com


References:

Yamaguchi, Masaya. 2006. The Complete Thesaurus of Musical Scales, revised
edition. New York: Masaya Music Services. ISBN 0-9676353-0-6.
http://en.wikipedia.org/wiki/List_of_musical_scales_and_modes


Examples:

scales(tonic="C",types=c("majorpentatonic","minorpentatonic")) ## Returns
tablatures for major and minor pentatonic scales in C ('do') on a guitar
(default instrument), indicating scale degrees.

scales(tonic="D#",except="C",strings=4,tune=c("G","D","A","E")) ## Returns
tablatures for all D# ('re#') scales NOT including C ('do') on a bass guitar
(4 stringed instrument with 'EADG' tuning).

scales(tonic="G",notes=c("C","D"),tune=c("D","B","G","D","G","G")) ##
```

```
Returns tablatures for all G ('sol') scales including C ('do') and D ('re',
dominant) on a guitar with an open G overtones tuning.

scales(tonic=0,strings=4,tune=c("E","A","D","G")) ## Returns position of
notes on the fretboard of a violin (4 stringed instrument with tuning GDAE).
```

Código da função scales

```
scales<-
function(tonic=1,notes=tonic,except=FALSE,region=c(0,frets),types="all",stri
ngs=6,tune=c(8,3,11,6,1,8),frets=15,degrees=TRUE){
  ## necessary subfunctions
  ## Converting - notes to numbers
  codconv<-function(x){
    errors<-rep(0,length(x))
    for(i in 1:length(x)){
      if(x[i]=="A"){x[i]<-1}
      else{
        if(x[i]=="A#"|x[i]=="Bb"|x[i]=="A#/Bb"|x[i]=="Bb/A#"){x[i]<-2}
        else{
          if(x[i]=="B"|x[i]=="Cb"|x[i]=="B/Cb"|x[i]=="Cb/B"){x[i]<-3}
          else{
            if(x[i]=="B#"|x[i]=="C"|x[i]=="B#/C"|x[i]=="C/B#"){x[i]<-4}
            else{
              if(x[i]=="C#"|x[i]=="Db"|x[i]=="C#/Db"|x[i]=="Db/C#"){x[i]<-5}
              else{
                if(x[i]=="D"){x[i]<-6}
                else{
if(x[i]=="D#"|x[i]=="Eb"|x[i]=="D#/Eb"|x[i]=="Eb/D#"){x[i]<-7}
                  else{
if(x[i]=="E"|x[i]=="Fb"|x[i]=="E/Fb"|x[i]=="Fb/E"){x[i]<-8}
                    else{
if(x[i]=="F"|x[i]=="E#"|x[i]=="F/E#"|x[i]=="E#/F"){x[i]<-9}
                      else{
if(x[i]=="F#"|x[i]=="Gb"|x[i]=="F#/Gb"|x[i]=="Gb/F#"){x[i]<-10}
                        else{
                          if(x[i]=="G"){x[i]<-11}
                          else{
if(x[i]=="G#"|x[i]=="Ab"|x[i]=="G#/Ab"|x[i]=="Ab/G#"){x[i]<-12}
                            else{errors[i]<-1}}}}}}}}}}}}
    if(sum(errors)>0){
      cat("\n\nDouble-check, chap. Your sheet included ",sum(errors),"non-
notes.\nNotes must be entered as capital letters from A to G, with '#' and
'b' representing accidents.\n\n")
    }
    if(is.matrix(x)){
      result<-matrix(as.numeric(x),nrow=nrow(x),ncol=ncol(x))
      rownames(result)<-rownames(x)
```

```
      colnames(result)<-paste("fr",0:(ncol(x)-1))
    }
    else{
      result<-as.numeric(x)
    }
    return(list(result,errors))
  }
  ## Converting - "numbers" to notes (character)
  noteconv<-function(x){
    if(max(x)>12){
      cat("\n\nDouble-check, mate, your sheet included non-notes.\n\nNotes
are integer values between 1 and 12")
    }
    else{
      for(i in 1:length(x)){
        if(x[i]==1){
          x[i]<-"A"
        }
        if(x[i]==2){
          x[i]<-"A#/Bb"
        }
        if(x[i]==3){
          x[i]<-"B/Cb"
        }
        if(x[i]==4){
          x[i]<-"C/B#"
        }
        if(x[i]==5){
          x[i]<-"C#/Db"
        }
        if(x[i]==6){
          x[i]<-"D"
        }
        if(x[i]==7){
          x[i]<-"D#/Eb"
        }
        if(x[i]==8){
          x[i]<-"E/Fb"
        }
        if(x[i]==9){
          x[i]<-"F/E#"
        }
        if(x[i]==10){
          x[i]<-"F#/Gb"
        }
        if(x[i]==11){
          x[i]<-"G"
        }
        if(x[i]==12){
          x[i]<-"G#/Ab"
        }
```

```
      }
      return(x)
    }
  }
  ## General function to create instrument fretboards, default is guitar
with 15 frets.
  # Reading "character" tuning vectors
  if(is.character(tune)){
    tune<-codconv(tune)[[1]]
  }
  # The function itself
  instr<-function(strings=strings,tune=tune,frets=frets){
    if(length(tune)!=strings){
      cat("\n\nYou've got untuned strings, mate. \nPlease enter a tune
vector with enough notes for all your chords")
    }
    else{
      fretboard<-
matrix(rep(NA,(frets+1)*strings),ncol=(frets+1),nrow=strings)
      rownames(fretboard)<-noteconv(tune)
      colnames(fretboard)<-paste("fr",0:frets)
      for(i in 0:frets){
        fretboard[,i+1]<-tune+i
      }
      ### correcting 'notes' with values higher than 12
      for(i in 1:strings){
        for(j in 1:(frets+1)){
          if(fretboard[i,j]>12){
            while(fretboard[i,j]>12){
              fretboard[i,j]<-(fretboard[i,j]-12)
            }
          }
          else{
            fretboard[i,j]<-fretboard[i,j]
          }
        }
      }
      return(fretboard)
    }
  }
  ## Coded scales
  # Greek modes (also tonal major=ionian and minor=eolian)
  ionian<-function(x){
    init<-c(x,x+2,x+4,x+5,x+7,x+9,x+11)
    for(i in 1:length(init)){
      if(init[i]>12){
        while(init[i]>12){
          init[i]<-(init[i]-12)
        }
      }
```

```
    else{
      init[i]<-init[i]
    }
  }
  return(init)
}
dorian<-function(x){
  init<-c(x,x+2,x+3,x+5,x+7,x+9,x+10)
  for(i in 1:length(init)){
    if(init[i]>12){
      while(init[i]>12){
        init[i]<-(init[i]-12)
      }
    }
    else{
      init[i]<-init[i]
    }
  }
  return(init)
}
phrygian<-function(x){
  init<-c(x,x+1,x+3,x+5,x+7,x+8,x+10)
  for(i in 1:length(init)){
    if(init[i]>12){
      while(init[i]>12){
        init[i]<-(init[i]-12)
      }
    }
    else{
      init[i]<-init[i]
    }
  }
  return(init)
}
lydian<-function(x){
  init<-c(x,x+2,x+4,x+6,x+7,x+9,x+11)
  for(i in 1:length(init)){
    if(init[i]>12){
      while(init[i]>12){
        init[i]<-(init[i]-12)
      }
    }
    else{
      init[i]<-init[i]
    }
  }
  return(init)
}
mixolydian<-function(x){
  init<-c(x,x+2,x+4,x+5,x+7,x+9,x+10)
  for(i in 1:length(init)){
```

```
    if(init[i]>12){
      while(init[i]>12){
        init[i]<-(init[i]-12)
      }
    }
    else{
      init[i]<-init[i]
    }
  }
  return(init)
}
eolian<-function(x){
  init<-c(x,x+2,x+3,x+5,x+7,x+8,x+10)
  for(i in 1:length(init)){
    if(init[i]>12){
      while(init[i]>12){
        init[i]<-(init[i]-12)
      }
    }
    else{
      init[i]<-init[i]
    }
  }
  return(init)
}
locrian<-function(x){
  init<-c(x,x+1,x+3,x+5,x+6,x+8,x+10)
  for(i in 1:length(init)){
    if(init[i]>12){
      while(init[i]>12){
        init[i]<-(init[i]-12)
      }
    }
    else{
      init[i]<-init[i]
    }
  }
  return(init)
}
## other tonal scales
harmonic<-function(x){
  init<-c(x,x+2,x+3,x+5,x+7,x+8,x+11)
  for(i in 1:length(init)){
    if(init[i]>12){
      while(init[i]>12){
        init[i]<-(init[i]-12)
      }
    }
    else{
      init[i]<-init[i]
```

```
    }
  }
  return(init)
}
melodic<-function(x){
  init<-c(x,x+1,x+4,x+5,x+7,x+8,x+11)
  for(i in 1:length(init)){
    if(init[i]>12){
      while(init[i]>12){
        init[i]<-(init[i]-12)
      }
    }
    else{
      init[i]<-init[i]
    }
  }
  return(init)
}
## pentatonic scales
minorpentatonic<-function(x){
  init<-c(x,x+3,x+5,x+7,x+10)
  for(i in 1:length(init)){
    if(init[i]>12){
      while(init[i]>12){
        init[i]<-(init[i]-12)
      }
    }
    else{
      init[i]<-init[i]
    }
  }
  return(init)
}
majorpentatonic<-function(x){
  init<-c(x,x+2,x+4,x+7,x+9)
  for(i in 1:length(init)){
    if(init[i]>12){
      while(init[i]>12){
        init[i]<-(init[i]-12)
      }
    }
    else{
      init[i]<-init[i]
    }
  }
  return(init)
}
## other miscellaneous scales
altered<-function(x){
  init<-c(x,x+1,x+3,x+4,x+6,x+8,x+10)
  for(i in 1:length(init)){
```

```
      if(init[i]>12){
        while(init[i]>12){
          init[i]<-(init[i]-12)
        }
      }
      else{
        init[i]<-init[i]
      }
    }
    return(init)
  }
  augmented<-function(x){
    init<-c(x,x+3,x+4,x+7,x+8,x+11)
    for(i in 1:length(init)){
      if(init[i]>12){
        while(init[i]>12){
          init[i]<-(init[i]-12)
        }
      }
      else{
        init[i]<-init[i]
      }
    }
    return(init)
  }
  chromatic<-function(x){
    init<-c(x,x+1,x+2,x+3,x+4,x+5,x+6,x+7,x+8,x+9,x+10,x+11)
    for(i in 1:length(init)){
      if(init[i]>12){
        while(init[i]>12){
          init[i]<-(init[i]-12)
        }
      }
      else{
        init[i]<-init[i]
      }
    }
    return(init)
  }
  blues<-function(x){
    init<-c(x,x+3,x+5,x+6,x+7,x+10)
    for(i in 1:length(init)){
      if(init[i]>12){
        while(init[i]>12){
          init[i]<-(init[i]-12)
        }
      }
      else{
        init[i]<-init[i]
      }
```

```
    }
    return(init)
  }
  flamenco<-function(x){
    init<-c(x,x+1,x+4,x+5,x+7,x+8,x+10)
    for(i in 1:length(init)){
      if(init[i]>12){
        while(init[i]>12){
          init[i]<-(init[i]-12)
        }
      }
      else{
        init[i]<-init[i]
      }
    }
    return(init)
  }
  doubleharmonic<-function(x){
    init<-c(x,x+1,x+4,x+5,x+7,x+8,x+11)
    for(i in 1:length(init)){
      if(init[i]>12){
        while(init[i]>12){
          init[i]<-(init[i]-12)
        }
      }
      else{
        init[i]<-init[i]
      }
    }
    return(init)
  }
  wholetone<-function(x){
    init<-c(x,x+2,x+4,x+6,x+8,x+10)
    for(i in 1:length(init)){
      if(init[i]>12){
        while(init[i]>12){
          init[i]<-(init[i]-12)
        }
      }
      else{
        init[i]<-init[i]
      }
    }
    return(init)
  }

  ## converting "tonic", "notes" and "except" arguments to numbers if given
as characters
  # tonic
  if(is.character(tonic)){
    tonic<-codconv(tonic)[[1]]
```

```
  }
  if(is.character(notes)){
    notes<-codconv(notes)[[1]]
  }
  if(is.character(except)){
    except<-codconv(except)[[1]]
  }
  ## Creating standard fretboard with given parameters
  fretboard<-instr(strings,tune,frets)
  ## Returning fretboard as result if user chooses to (by entering tonic=0)
  if(tonic==0){
    instrument<-noteconv(fretboard)
    sink("instrument.txt")
    print(instrument)
    sink()
    return(noteconv(fretboard))
  }
  ## Building list of scales based on given "types"
  if("all" %in% types){scalinit<-list(ionian=ionian(tonic),
                                      dorian=dorian(tonic),
                                      phrygian=phrygian(tonic),
                                      lydian=lydian(tonic),
                                      mixolydian=mixolydian(tonic),
                                      eolian=eolian(tonic),
                                      locrian=locrian(tonic),
                                      altered=altered(tonic),
                                      augmented=augmented(tonic),
                                      chromatic=chromatic(tonic),
                                      flamenco=flamenco(tonic),
                                      harmonic=harmonic(tonic),
                                      doubleharmonic=doubleharmonic(tonic),
                                      melodic=melodic(tonic),
minorpentatonic=minorpentatonic(tonic),
majorpentatonic=majorpentatonic(tonic),
                                      wholetone=wholetone(tonic),
                                      blues=blues(tonic))
  }else{
    scalinit<-list(ionian=if("ionian" %in% types){ionian(tonic)}else{NA},
                   dorian=if("dorian" %in% types){dorian(tonic)}else{NA},
                   phrygian=if("phrygian" %in%
types){phrygian(tonic)}else{NA},
                   lydian=if("lydian" %in% types){lydian(tonic)}else{NA},
                   mixolydian=if("mixolydian" %in%
types){mixolydian(tonic)}else{NA},
                   eolian=if("eolian" %in% types){eolian(tonic)}else{NA},
                   locrian=if("locrian" %in% types){locrian(tonic)}else{NA},
                   altered=if("altered" %in% types){altered(tonic)}else{NA},
                   augmented=if("augmented" %in%
types){augmented(tonic)}else{NA},
                   chromatic=if("chromatic" %in%
```

```
types){chromatic(tonic)}else{NA},
                    flamenco=if("flamenco" %in%
types){flamenco(tonic)}else{NA},
                    harmonic=if("harmonic" %in%
types){harmonic(tonic)}else{NA},
                    doubleharmonic=if("doubleharmonic" %in%
types){doubleharmonic(tonic)}else{NA},
                    melodic=if("melodic" %in% types){melodic(tonic)}else{NA},
                    minorpentatonic=if("minorpentatonic" %in%
types){minorpentatonic(tonic)}else{NA},
                    majorpentatonic=if("majorpentatonic" %in%
types){majorpentatonic(tonic)}else{NA},
                    wholetone=if("wholetone" %in%
types){wholetone(tonic)}else{NA},
                    blues=if("blues" %in% types){blues(tonic)}else{NA})
  }
  scalinit<-scalinit[!is.na(scalinit)]
  ## Removing scales which do not include the notes given in "notes"
argument
  notes<-c(tonic,notes)
  if("all" %in% types){scal_nt<-list(ionian=0,
                                     dorian=0,
                                     phrygian=0,
                                     lydian=0,
                                     mixolydian=0,
                                     eolian=0,
                                     locrian=0,
                                     altered=0,
                                     augmented=0,
                                     chromatic=0,
                                     flamenco=0,
                                     harmonic=0,
                                     doubleharmonic=0,
                                     melodic=0,
                                     minorpentatonic=0,
                                     majorpentatonic=0,
                                     wholetone=0,
                                     blues=0)}else{
                                       scal_nt<-list(ionian=if("ionian" %in%
types){0}else{NA},
                                                     dorian=if("dorian" %in%
types){0}else{NA},
                                                     phrygian=if("phrygian"
%in% types){0}else{NA},
                                                     lydian=if("lydian" %in%
types){0}else{NA},
mixolydian=if("mixolydian" %in% types){0}else{NA},
                                                     eolian=if("eolian" %in%
types){0}else{NA},
                                                     locrian=if("locrian"
%in% types){0}else{NA},
```

```
                                                    altered=if("altered"
%in% types){0}else{NA},
augmented=if("augmented" %in% types){0}else{NA},
chromatic=if("chromatic" %in% types){0}else{NA},
                                                    flamenco=if("flamenco"
%in% types){0}else{NA},
                                                    harmonic=if("harmonic"
%in% types){0}else{NA},
doubleharmonic=if("doubleharmonic" %in% types){0}else{NA},
                                                    melodic=if("melodic"
%in% types){0}else{NA},
minorpentatonic=if("minorpentatonic" %in% types){0}else{NA},
majorpentatonic=if("majorpentatonic" %in% types){0}else{NA},
wholetone=if("wholetone" %in% types){0}else{NA},
                                                    blues=if("blues" %in%
types){0}else{NA})
                                        }
  scal_nt<-scal_nt[!is.na(scal_nt)]
  for(i in 1:length(scal_nt)){
    if(all(notes %in% scalinit[[i]])){
      scal_nt[[i]]<-scalinit[[i]]
    }else{
      scal_nt[[i]]<-NA
    }
  }
  scal_nt<-scal_nt[!is.na(scal_nt)]
  ## Removing notes given in "except"
  if(except %in% notes){cat("\n\n Check your conditions, chap. You're asking
me to include and not include the same note \n\n")
  stop()
  }
  else{
  for(i in 1:length(scal_nt)){
    if(any(except %in% scal_nt[[1]])){
      scal_nt[[i]]<-NA
    }
  }
  }
  scal_fin<-scal_nt[!is.na(scal_nt)]
  ## Returning message if no scales fit conditions
  if(length(scal_fin)==0){
    cat("\n\n Too bad, mate. No scales currently included in the function
fit the conditions entered.\n\n")
    stop()
  }
  ## Results list with size equal to length of final scales vector
  results<-scal_fin
  ## Emptying space for each scale in results list
  for(i in 1:length(scal_fin)){
    results[[i]]<-
```

```
matrix(rep(NA,strings*(frets+1)),nrow=strings,ncol=frets+1,dimnames=list(not
econv(tune),paste("fr",0:frets)))
  }
  ## Creating fretboards with scales marked as "X" or "degrees" (according
to "degrees" argument), saving these to results list
  ## Initial results fretboard with numbers
  for(i in 1:strings){
    for(j in 1:(frets+1)){
      for(k in 1:length(scal_fin)){
        if(fretboard[i,j] %in% scal_fin[[k]]){
          results[[k]][i,j]<-which(scal_fin[[k]]==fretboard[i,j])
        }
        else{
          results[[k]][i,j]<-"-"
        }
      }
    }
  }
  ## Degrees FALSE: scales marked as "X"
  if(degrees==FALSE){
    for(k in 1:length(results)){
      for(i in 1:strings){
        for(j in 1:(frets+1)){
          if(results[[k]][i,j]!="-"){
            results[[k]][i,j]<-"X"
          }
        }
      }
    }
  }
  ## Degrees TRUE: scales marked as degrees relative to tonic (tonic = I)
  grad<-c("I","II","III","IV","V","VI","VII","VIII","IX","X","XI","XII")
  if(degrees==TRUE){
    for(k in 1:length(results)){
      for(i in 1:strings){
        for(j in 1:(frets+1)){
          for(n in 1:length(scal_fin[[k]])){
            if(results[[k]][i,j]==n){
              results[[k]][i,j]<-grad[n]
            }
          }
        }
      }
    }
  }
  ## Cutting resulting fretboards to show only frets given in "region"
argument
  final<-results
  for(k in 1:length(results)){
    final[[k]]<-results[[k]][,c((region[1]+1):(region[2]+1))]
  }
```

```
  ## Print final results to .txt file
  sink("scales.txt")
  print(final)
  sink()
  ## Returning results as object of type (list) for future use
  return(final)
}
```

Arquivos para Download

Código função scales

Help file função scales