Luísa Novara Monclar Gonçalves

Graduanda em Ciências Biológicas na USP, começando a Iniciação Científica na área de Ecologia, no LABTROP.

Currículo Lattes de Luísa Novara

Meus Exercícios

Linque para meus exercícios resolvidos aqui.

Trabalho final

Introdução às propostas A e B

A Teoria Neutra da Biodiversidade de Hubbell foi extremamente importante para a Ecologia enquanto ciência, uma vez que ela tenta unificar e simplificar os processos que geram a estrutura e a dinâmica de comunidades no planeta. Essa teoria cria um cenário neutro a partir do qual podemos comparar cenários mais complexos, e, assim, obter informações acerca da importância de alguns fatores na estruturação ou dinâmica da comunidade que estamos estudando; muitas vezes, podemos também nos ater ao cenário neutro, e tecer conclusões mais simples acerca de padrões que antes julgávamos demasiado complexos ou multicausais.

Proposta A

O primeiro passo para explorarmos as possibilidades que o uso de um modelo neutro em Ecologia pode nos dar é compreender como funciona um cenário baseado em seus pressupostos. Para isso, podemos criar uma função que simule uma dinâmica neutra de comunidades, a partir dos pressupostos de Hubbell. Essa função poderia ter como argumentos o número inicial de espécies presentes na comunidade, o número de indivíduos por espécie, o número de propágulos total produzido por cada indivíduo e o número de ciclos que o usuário gostaria de executar. A cada ciclo, seria selecionado aleatoriamente um indivíduo para morrer (a probabilidade de morte é o inverso do tempo de vida, que seria calculado como o esforço reprodutivo total dividido pelo número de propágulos produzidos a cada ciclo) e, para substitui-lo, seria escolhido, também aleatoriamente, um indivíduo do banco de propágulos. Ao final do número de ciclos determinado, observaríamos o número de espécies final presente na comunidade e o número de indivíduos por espécie. Se possível, poderíamos também incluir no cenário neutro de Hubbell um trade-off entre esforço reprodutivo instantâneo e tempo de vida. Nesse caso, seria possível observar se, com a aplicação desse trade-off, a dinâmica neutra selecionaria, ao longo de vários ciclos, indivíduos que produzissem menos propágulos por ciclo e vivessem por mais ciclos (análogos a estrategistas k) ou indivíduos que produzissem mais propágulos por ciclo e vivessem por menos ciclos (análogos a estrategistas r). Esse trade-off poderia ser modelado com o acréscimo de um argumento na função, que seria a proporção

de variação no número de propágulos produzidos por ciclo. Poderíamos também acrescentar o sistema de reprodução sexuada com herança de caracteres contínuos (no caso, o caráter contínuo seria a média do número de prole produzida por ciclo). Com essa função, eu poderia executar diversas simulações e responder à pergunta principal da minha IC: a dinâmica neutra é seletiva? O output dessa função com o acréscimo desse último argumento seria diferente, uma vez que o objetivo seria observar se indivíduos que produzem mais ou menos propágulos por ciclo estariam sendo selecionados.

Proposta B

Outra maneira de nos utilizarmos da Teoria Neutra da Biodiversidade é compararmos o cenário real da estrutura de uma dada comunidade de estudo com um cenário neutro gerado pelos nossos dados. Para isso, podemos criar uma função que calcula o quanto da estrutura observada (diversidade e abundância) de nossa comunidade se assemelha a um cenário neutro (gerado a partir do valor de diversidade de espécies presentes na comunidade original). A função se daria por comparação das curvas de distribuição de abundância das espécies e de SAR geradas para o cenário real e o cenário neutro. No caso dos gráficos de distribuição de abundância das espécies, poderíamos comparar, entre ambos os cenários, cada valor de número de espécies referente a um ranking de abundância, acusando diferença significativa ou não significativa entre eles. Para os gráficos de SAR, poderíamos comparar as curvas como um todo ou segmento a segmento, o que facilitaria na identificação das principais diferenças entre os dois cenários. Ao utilizarmos uma função como esta, há a possibilidade de chegarmos à conclusão de que processos estocásticos simples de nascimento e morte de indivíduos são capazes de gerar uma estrutura de comunidade semelhante àquela de que estudamos, em vez de partir de processos mais complexos para explicar nosso cenário. Podemos chegar também à conclusão de que nosso cenário não poderia ser gerado a partir de uma dinâmica neutra, e, a partir disso, hipotetizar e testar a importância de outros fatores nessa determinação. De uma forma ou de outra, a comparação com o modelo neutro é bastante útil para conhecermos melhor a estrutura da comunidade que estamos estudando.

Comentários

A proposta A é boa, mas um pouco além do que pensamos para o trabalho final. É muito mais trabalhosa do que provavelmente imaginou. Sugiro que faça uma função que rode apenas o início da proposta. Uma dinâmica de comunidade neutra com taxas per capitas de mortalidade e nascimento aleatórios iguais para as espécies. A principio, os resultados de uma simulações como esta não são interessantes pois devem levar a comunidades de uma unica espécie pela deriva ecológica. O que importa é que define a estrutura básica para a criação da função proposta da teoria Neutra o que já é um trabalho muito bacana e um grande desafio!

- Faltou definir os parâmetros de entrada do cenário inicial da simulação e o objeto de saída...

Corrija sua proposta ajustada às sugestões

— *Alexandre Adalardo de Oliveira 2014/04/25 10:52*

Proposta A ajustada

O primeiro passo para explorarmos as possibilidades que o uso de um modelo neutro em Ecologia pode nos dar é compreender como funciona um cenário baseado em seus pressupostos. Para isso, podemos criar uma função que simule uma dinâmica neutra de comunidades, a partir dos pressupostos de Hubbell. Essa função teria como argumentos o número inicial de espécies presentes na comunidade, o número de indivíduos por espécie, o número de ciclos que o usuário gostaria de executar e o número de mortes por ciclo. A cada ciclo, seria selecionado aleatoriamente um ou mais indivíduos (sem reposição) para morrer. Para substitui-los, seria escolhido, também aleatoriamente, um ou mais indivíduos (com reposição) que atue como parental único daquele que irá ocupar o local vago. Ao final do número de ciclos determinado, observaríamos o número de espécies presente na comunidade. Poderiam, também, ser gravados esses valores para cada ciclo rodado, a fim de avaliar a dinâmica temporal da simulação. Também seria gerado um gráfico com a distribuição de abundâncias final das espécies. Em termos mais precisos, os parâmetros de entrada do cenário inicial da simulação seriam o número de espécies da comunidade, o número de indivíduos por espécie, o número de ciclos que o usuário gostaria de simular e o número de mortes aplicado em cada ciclo. O objeto de saída seria composto pelo valor final do número de espécies presentes na comunidade, por um gráfico com a relação entre o número de ciclos rodados e o número de espécies existentes na comunidade e por um gráfico de distribuição de abundâncias das espécies após os ciclos terem sido rodados.

Página de Ajuda (Help)

```
hubbells.game       package : nenhum            R Documentation

Dinâmica neutra baseada na Teoria Neutra da Biodiversidade, de Hubbell.

Description :

hubbells.game() realiza simulações de uma dada comunidade (cujas condições
iniciais são determinadas pelo
usuário), que é submetida a uma dinâmica neutra baseada somente em eventos
estocásticos de morte e nascimento de
indivíduos que compõem um jogo de soma zero. A função retorna um gráfico com
a distribuição de abundância final
da comunidade e outro com a variação da riqueza ao longo dos ciclos, além da
média do número de espécies final de
todas as simulações.

Usage :

hubbells.game(s, a, nc, dc, nsim, details=FALSE)
```

```
Arguments :

s   Numérico. Número inicial de espécies (riqueza) existente na comunidade.
Deve ser maior ou igual a 1.
a   Numérico. Número inicial de indivíduos (abundância) por espécie. Deve
ser maior ou igual a 1.
nc  Numérico. Número de ciclos a serem rodados. Deve ser maior ou igual a 0.
dc  Numérico. Número de mortes e nascimentos por ciclo. Deve ser maior que o
número total de indivíduos presentes na comunidade, dado por s*a.
nsim    Numérico. Número de simulações (réplicas) a serem rodadas. Deve ser
maior ou igual a 1.
details Lógico. O array que contém a identidade de todos os indivíduos a
cada ciclo (para todas as simulações) deve ser retornado ?

Details :

Em 2001, Hubbell organizou a Teoria Neutra da Biodiversidade, uma proposta
para explicar a riqueza e a
abundância de espécies de comunidades (e, por vezes, sua estrutura espacial)
a partir de processos estocásticos
de migração, especiação e sucessão de eventos de morte e nascimento de
indivíduos. A função hubbells.game()
utiliza apenas este último processo para determinar a dinâmica temporal da
comunidade ; desta forma, gera um
modelo não-espacial. Na dinâmica, os indivíduos compõem um jogo de soma zero
(o número de indivíduos total da
comunidade não se altera) e sua reprodução é assexuada, já que não há
formação de pares para gerar novos
indivíduos. A característica herdada de parental para prole é sua
identidade, isto é, a espécie a que pertence.
Desta forma, ao final da dinâmica, podemos observar qual foi a variação
temporal na riqueza e na abundância das
espécies presentes na comunidade.

Value :

A função hubbells.game() retorna, no console, a média, utilizando-se todas
as simulações rodadas, da riqueza
final da comunidade. Isto é, calcula o número de espécies presentes na
comunidade após os ciclos serem rodados
para todas as simulações e retorna a média desses valores.
Na janela gráfica, a função retorna dois gráficos. O primeiro é a
distribuição de abundância da comunidade em
todas as simulações ; a cada simulação é atribuída uma cor diferente. O
segundo gráfico apresenta a variação da
riqueza da comunidade ao longo dos ciclos rodados, novamente para todas as
simulações.

Warning :
```

```
A função é interrompida e mensagens de erro são retornadas em casos de :
ausência dos argumentos s e/ou a ;
argumentos s, a e/ou nsim com valor menor ou igual a zero ; argumentos nc
e/ou dc com valor menor que zero,
argumento dc igual a zero quando argumento nc for maior que zero ; e
argumento dc com valor maior ou igual ao
número de indivíduos da comunidade.
Mensagens de aviso são retornadas em casos de : ausência do argumento nc,
então será utilizado nc igual a zero ;
ausência do argumento dc, então será utilizado dc igual a 1 ; ausência do
argumento nsim, então será utilizado
nsim igual a 1. Será enviada também uma mensagem de aviso quando o usuário
determinar nc igual a zero, caso ele o
tenha feito por engano. Ademais, é retornada uma mensagem de aviso que
informa o usuário quanto à aplicação de um
resíduo aleatório nos valores originais das variáveis abundância e riqueza
nos gráficos gerados, para evitar que
linhas ou pontos com mesmo valor de x e/ou y apareçam sobrepostos,
dificultando a visualização das informações.

Author(s) :

Luísa Novara
luisanovara@gmail.com

References :

Hubbell, S.P. (2001) The Unified Neutral Theory of Biodiversity and
Biogeography, Princeton University Press.
Rosindell, J.; Hubbell, S.P.; Etienne, R.S. (2011) The Unified Neutral
Theory of Biodiversity and Biogeography at Age Ten. Trends in Ecology &
Evolution. 26, 340–348.
See Also :

sample() para amostragem aleatória com ou sem reposição

Examples :

# Exemplo 1 : Comunidade com riqueza 50 e abundância por espécie 3 submetida
a 10 ciclos com 5 mortes e 5 nascimentos cada, replicado 10 vezes. Não foi
solicitado o retorno do array com a identidade das espécies ao longo dos
ciclos.
hubbells.game(50,3,10,5,10,F)
# ou
hubbells.game(50,3,10,5,10) # o default do argumento details é FALSE
## Observação : faça das duas formas e perceba que, ainda que as condições
iniciais sejam as mesmas em ambos os casos, surgem resultados distintos,
Isso ocorre porque a dinâmica neutra é estocástica.

# Exemplo 2 : Comunidade com riqueza 20 e abundância por espécie 4 submetida
a 1 ciclo com 15 mortes e 15 nascimentos, replicado 5 vezes.
```

```
hubbells.game(20,4,1,15,5,F)

# Exemplo 3 : Mesma comunidade do exemplo anterior submetida à mesma dinâmica. Agora, é solicitado o retorno do array.
hubbells.game(20,4,1,15,5,T) # Observe que a identidade dos indivíduos muda do ciclo 0 (condição inicial) para o ciclo 1. Isso ocorreu porque foram selecionados aleatoriamente 15 indivíduos para serem eliminados e, dos indivíduos que restaram na comunidade, foram selecionados 15 para gerar um indivíduo de identidade equivalente a sua para ocupar os lugares vagos.

# Exemplo 4 : Comunidade com riqueza 10 e abundância por espécie 5 submetida a nenhum ciclo, replicado 2 vezes. Note que, ainda que tenha sido definido um valor para o número de mortes e nascimentos por ciclo, ele não é utilizado.
hubbells.game(10,5,0,1,2) # os resultados finais reproduzem as condições iniciais, já que foi definido valor zero para o número de ciclos rodados
hubbells.game(10,5,0,0,2) # esta linha de comando é funcionamente equivalente à anterior
## Entretanto, se for definido um valor maior que zero para o número de ciclos a serem rodados, deve ser definido também um valor maior que zero para o número de mortes e nascimentos por ciclo
hubbells.game(10,5,1,0,2) # erro na função

# Exemplo 5 : Comunidade com riqueza 5 e abundância por espécie 2 submetida a 3 ciclos com 12 mortes e 12 nascimentos cada, com 1 réplica. A função é interrompida, já que o número de mortes/nacimentos por ciclo é maior do que o número total de indivíduos da comunidade.
hubbells.game(5,2,3,12,1) # dessa forma, a comunidade seria levada à extinção no primeiro ciclo
hubbells.game(5,2,3,8,1) # diminuindo o número de mortes por ciclo para um valor menor que o número total de indivíduos (no caso, 10), a função é executada corretamente
```

Código da Função

```
hubbells.game <- function(s, a, nc, dc, nsim, details=F) # s = número de espécies inicial, a = número inicial de indivíduos por espécie, nc = número de ciclos, dc = número de mortes por ciclo, nsim = número de simulações
{
  # Verificando se há argumentos faltantes
  if(missing(s)) stop("É obrigatória a definição de um valor para o número inicial de espécies (s).") # caso o argumento "s" esteja faltante, a função é interrompida e é enviada uma mensagem de erro
  if(missing(a)) stop("É obrigatória a definição de um valor para a abundância inicial das espécies (a).") # caso o argumento "a" esteja faltante, a função é interrompida e é enviada uma mensagem de erro
  if(missing(nc)){ # caso o argumento "nc" esteja faltante
    nc=0 # será utilizado nc = 0
```

```
    warning("Não foi definido um valor para o número de ciclos a serem
rodados (nc). Logo, foi utilizado nc = 0.") # e uma mensagem de aviso será
retornada
    }
  if(missing(dc)){ # caso o argumento "dc" esteja faltante
    dc=1 # será utilizado dc = 1
    warning("Não foi definido um valor para o número de mortes por ciclo
(dc). Logo, foi utilizado dc = 1, presente no modelo neutro clássico de
Hubbell.") # e uma mensagem de aviso será retornada
    }
  if(missing(nsim)){ # caso o argumento "nsim" esteja faltante
    nsim=1 # será utilizado nsim = 1
    warning("Não foi definido um valor para o número de simulações a serem
rodadas (nsim). Logo, foi utilizado nsim = 1.")   # e uma mensagem de aviso
será retornada
  }
  # Verificando se os valores determinados pelo usuário para os argumentos
são lógicos
  if(s<=0) stop("O número inicial de espécies deve ser maior que zero.") #
caso o valor do argumento "s" seja menor ou igual a zero, a função é
interrompida e é enviada uma mensagem de erro
  if(a<=0) stop("A abundância inicial das espécies deve ser maior que
zero.") # caso o valor do argumento "a" seja menor ou igual a zero, a função
é interrompida e é enviada uma mensagem de erro
  if(nc==0){ # caso o valor do argumento "nc" seja igual a zero
    warning("O número de ciclos (nc) definido foi zero.") # é enviada uma
mensagem de aviso
  }
  if(nc<0) stop("O número de ciclos (nc) definido deve ser maior ou igual a
zero.") # caso o valor do argumento "nc" seja menor que zero, a função é
interrompida e é enviada uma mensagem de erro
  if(dc==0 & nc>0) stop("O número de mortes por ciclo (dc) deve ser maior
que zero.") # caso o valor do argumento "dc" seja zero, a função é
interrompida e é enviada uma mensagem de erro
  if(dc<0) stop("O número de mortes por ciclo (dc) definido deve ser maior
ou igual a zero.") # caso o valor do argumento "dc" seja menor que zero, a
função é interrompida e é enviada uma mensagem de erro
  if(nsim<=0) stop("O número de simulações deve ser maior que zero.") # caso
o valor do argumento "nsim" seja menor ou igual a zero, a função é
interrompida e é enviada uma mensagem de erro
  if(dc>=(s*a)) stop ("O número de mortes por ciclo deve ser menor do que o
número de indivíduos total da comunidade.") # caso o número de de mortes por
ciclo determinado seja maior ou igual ao número de indivíduos presentes na
comunidade, a função é interrompida e é enviada uma mensagem de erro
  # Após todas as verificações: Cálculo inicial
  n <- s*a # número total de indivíduos na comunidade (valor fixo)
  # Criando array para guardar os resultados
  resultado.array <-
array(data=,dim=c((nc+1),n,nsim),dimnames=list(paste("ciclo",0:(nc),sep="
"),paste("ind",1:n,sep=" "),paste("simulação",1:(nsim),sep=" "))) # cria
array cuja primeira dimensão é composta por linhas referentes ao número de
```

```
ciclos rodados, cuja segunda dimensão é composta por colunas referentes aos indivíduos presentes na comunidade (as linhas e colunas formam matrizes) e cuja terceira dimensão é composta por nsim matrizes, referentes ao número de simulações rodadas.
  # Condições iniciais
  vetor.esp <- rep(1:s, each=a) # vetor com as espécies de todos os indivíduos no ciclo 0 (retiradas das condições iniciais determinadas pelo usuário)
  # Iniciando: caso o número de ciclos seja zero
  if(nc==0)
  {
    ## Guardando as condições iniciais na primeira linha das matrizes do array
      resultado.array[1,,] <- vetor.esp # o vetor com a identidade dos indivíduos (espécies a que pertencem) é guardado na linha 1 de todas as colunas em todas as matrizes do array
    ## Alterando parâmetros para construção de gráficos
      par(mfrow=c(1,2)) # altera os parâmetros gráficos para plotarmos 2 gráficos na mesma janela, um ao lado do outro
    ## Criando o gráfico de distribuição de abundâncias
    abund.list <- list() # cria lista vazia para guardar a distribuição de abundância final de cada simulação
    for (i in 1:nsim){ # cria loop com contador de 1 até o número de simulações determinado pelo usuário (nsim)
    abund.list[[i]] <- table(table(resultado.array[(nc+1),,i])) # guarda, na lista criada, o valor de distribuição de abundância final de cada simulação
    }
    for(i in 1:nsim){ # cria loop com contador de 1 até o número de simulações determinado pelo usuário (nsim)
      names(abund.list[[i]]) <- jitter(as.numeric(names(abund.list[[i]]))) # substitui os valores de abundância calculados por valores acrescidos de um ruído, para evitar sobreposições ao plotarmos o gráfico de distribuição de abundância
    }
    plot(abund.list[[1]],lty=2,xlab="Abundância Final",ylab="Frequência de Espécies",main="Distribuição de Abundância",col=1,ylim=c((min(sapply(abund.list,min))),(max(sapply(abund.list,max)))),xlim=c((round(min(sapply(X=abund.list,FUN=function(X){min(as.numeric(names(X)))}))-1),(round(max(sapply(X=abund.list,FUN=function(X){max(as.numeric(names(X)))}))+1)),axes=F) # plota a distribuição de abundância final da primeira simulação, presente na primeira posição da lista criada
    if(nsim>1){ # caso o número de simulações seja maior que 1, devemos continuar plotando
    for(i in 2:nsim){ # cria loop com contador de 2 até o número de simulações determinado pelo usuário
      lines(abund.list[[i]],lty=2,col=i) # plota as distribuições de abundância finais das demais simulações na mesma janela gráfica. Cada simulação tem uma cor própria.
    }
    }
```

```
axis(1,at=c(round((min(sapply(X=abund.list,FUN=function(X){min(as.numeric(na
mes(X)))}))):round((max(sapply(X=abund.list,FUN=function(X){max(as.numeric(
names(X)))}))),labels=c(round((min(sapply(X=abund.list,FUN=function(X){min
(as.numeric(names(X)))}))):round((max(sapply(X=abund.list,FUN=function(X){m
ax(as.numeric(names(X)))}))) # plota o eixo x
axis(2,at=c((min(sapply(abund.list,min))):(max(sapply(abund.list,max)))),lab
els=c((min(sapply(abund.list,min))):(max(sapply(abund.list,max))))) # plota
o eixo y
    ## Criando o gráfico de riqueza x nc
    s.final <- c() # cria um vetor vazio para guardar a riqueza final de
cada simulação
    for(i in 1:nsim){ # cria loop com contador de 1 até o número de
simulações determinado
      s.final[i] <- length(unique(resultado.array[(nc+1),,i])) # extrai do
array a riqueza final de cada simulação e guarda no vetor criado
    }
    s.final.med <- mean(s.final) # guarda em um objeto a média da riqueza
final de espécies da comunidade
      plot(s.final[1],type="p",main="Variação da
Riqueza",ylab="Riqueza",xlab="Ciclo",col=1,ylim=c(s,s),axes=F) # plota a
riqueza em função do número de ciclos rodados. Não há necessidade de plotar
a riqueza final de todas as simulações, já que ela é a mesma para todas.
      if(nsim>1){ # caso o número de simulações seja maior que 1, devemos
continuar plotando
      for(i in 2:nsim){ # cria loop com contador de 2 até o número de
simulações determinado pelo usuário
        points(jitter(s.final[i]),lty=2,col=i) # plota as distribuições de
abundância finais das demais simulações na mesma janela gráfica (com adição
de ruído, para evitar sobreposição gráfica dos pontos). Cada simulação tem
uma cor própria.
        }
      }
      axis(side=1,at=c(0:2),labels=c(-1:1)) # plota o eixo x do gráfico
axis(side=2,at=c(0,s.final.med,(2*s.final.med)),labels=c("",s.final.med,""))
# plota o eixo y do gráfico
      axis(side=3,at=c(0,2),labels=c("","")) # plota a borda superior do
gráfico
      axis(side=4,at=c(0,(2*s.final.med)),labels=c("","")) # plota a borda
direita do gráfico
    ## Retornando aos parâmetros usuais
      par(mfrow=c(1,1)) # retorna os parâmetros gráficos para o default do R
    ## Mensagem de aviso acerca dos resíduos gerados para construção do
gráfico
    warning("A fim de evitar sobreposição de linhas ou pontos com mesmo
valor de x e/ou y nos gráficos, estes foram plotados a partir de um resíduo
aleatório gerado para cada valor original de x ou y.") # retorna mensagem de
aviso acerca dos resíduos associados aos valores de x ou y nos gráficos
    ## Outputs
      ### Opção sem retorno do array
      if(details==F){ # não retorna o array "resultado.array"
      cat("Média da riqueza final\n") # retorna no console a frase "Média do
```

```
número de espécies final"
      return(s.final.med) # retorna a média da riqueza final
      }
      ### Opção com retorno do array
      if(details==T){ # retorna o array "resultado.array"
      resulta=list("Média da riqueza final"=s.final.med,"Indentidade dos
indivíduos ao final de cada ciclo"=resultado.array) # cria uma lista,
chamada "resulta", com a média da riqueza final e com o array
"resultado.array"
      return(resulta) # retorna a lista "resulta"
      }
  }
  # Iniciando: caso o número de ciclos seja maior que zero
  else
  {
  ## Guardando as condições iniciais na primeira linha das matrizes do array
    resultado.array[1,,] <- vetor.esp
  ## Rodando os ciclos
    for(k in 1:nsim){ # cria loop com contador de 1 até o número de
simulações determinado pelo usuário (nsim)
    for(i in (2:(nc+1))){ # cria loop com contador de 2 até o número de
ciclos determinado pelo usuário (nc) + 1
  ## Morte
    ind.morto <- sample(x=n,size=dc,replace=F) # seleção de dc indivíduos
que irão morrer (sem reposição) a partir de n indivíduos, com igual
probabilidade
    vetor.esp[c(ind.morto)] <- NA # morte dos indivíduos selecionados,
atribuindo NA a suas identidades
  ## Nascimento
    esp.ind.novo <- sample(x=na.omit(vetor.esp),size=dc,replace=T) # seleção
da espécie dos indivíduos novos a partir das espécies dos indivíduos
restantes (com reposição)
    vetor.esp[is.na(vetor.esp)] <- esp.ind.novo # substituição das espécies
dos indivíduos mortos pelas espécies dos indivíduos novos
    resultado.array[i,,k] <- vetor.esp # guarda o resultado nas linhas de 2
até (nc+1) (na linha 1, já estão gravadas as condições iniciais) das
matrizes do array
    }
    }
  ## Alterando parâmetros para construção de gráficos
    par(mfrow=c(1,2)) # altera os parâmetros gráficos para plotarmos 2
gráficos na mesma janela, um ao lado do outro
  ## Criando o gráfico de distribuição de abundâncias
  abund.list <- list() # cria lista vazia para guardar a distribuição de
abundância final de cada simulação
  for (i in 1:nsim){ # cria loop com contador de 1 até o número de
simulações determinado pelo usuário
    abund.list[[i]] <- table(table(resultado.array[(nc+1),,i])) # guarda, na
lista criada, o valor de distribuição de abundância final de cada simulação
  }
```

```
  for(i in 1:nsim){ # cria loop com contador de 1 até o número de simulações
determinado pelo usuário (nsim)
    names(abund.list[[i]]) <- jitter(as.numeric(names(abund.list[[i]]))) #
substitui os valores de abundância calculados por valores acrescidos de um
ruído, para evitar sobreposições ao plotarmos o gráfico de distribuição de
abundância
  }
  plot(abund.list[[1]],lty=2,xlab="Abundância Final",ylab="Frequência de
Espécies",main="Distribuição de
Abundância",col=1,ylim=c((min(sapply(abund.list,min))),(max(sapply(abund.lis
t,max)))),xlim=c((round(min(sapply(X=abund.list,FUN=function(X){min(as.numer
ic(names(X)))})))-1),(round(max(sapply(X=abund.list,FUN=function(X){max(as.n
umeric(names(X)))})))+1)),axes=F) # plota a distribuição de abundância final
da primeira simulação, presente na primeira posição da lista criada
  if(nsim>1){ # caso o número de simulações seja maior que 1, devemos
continuar plotando
    for(i in 2:nsim){ # cria loop com contador de 2 até o número de
simulações determinado pelo usuário
      lines(abund.list[[i]],lty=2,col=i) # plota as distribuições de
abundância finais das demais simulações na mesma janela gráfica. Cada
simulação tem uma cor própria.
    }
  }
axis(1,at=c(round((min(sapply(X=abund.list,FUN=function(X){min(as.numeric(na
mes(X)))}))))):round((max(sapply(X=abund.list,FUN=function(X){max(as.numeric(
names(X)))}))))),labels=c(round((min(sapply(X=abund.list,FUN=function(X){min
(as.numeric(names(X)))}))))):round((max(sapply(X=abund.list,FUN=function(X){m
ax(as.numeric(names(X)))})))))) # plota eixo x
axis(2,at=c(1:(max(sapply(abund.list,max)))),labels=c(1:(max(sapply(abund.li
st,max))))) # plota eixo y
## Criando o gráfico de riqueza x nc
    s.final <- c() # cria um vetor vazio para guardar a riqueza final de
cada simulação
    for(i in 1:nsim){ # cria loop com contador de 1 até o número de
simulações determinado (nsim)
        s.final[i] <- length(unique(resultado.array[(nc+1),,i])) # extrai do
array a riqueza final de cada simulação e guarda no vetor criado
    }
    s.final.med <- mean(s.final) # guarda em um objeto a média da riqueza
final de espécies da comunidade
    s.final.mat <- matrix(data=,nrow=(nc+1),ncol=nsim) # cria uma matriz
vazia com nc+1 linhas e nsim colunas para guardar as riquezas de todas as
simulações a cada ciclo rodado
    for(k in 1:nsim){ # cria loop com contador de 1 até o número de
simulações determinado pelo usuário (nsim)
      for(j in 1:(nc+1)) { # cria loop com contador de 1 até o número de
ciclos determinado pelo usuário mais 1 (nc+1)
        s.final.mat[j,k] <- (length(unique(resultado.array[j,,k]))) # extrai
do array a riqueza da comunidade a cada ciclo rodado em todas as simulações
e guarda na matriz criada
      }
```

```
    }
    matplot((s.final.mat),type="l",main="Variação da
Riqueza",ylab="Riqueza",xlab="Ciclo",col=1:nsim,lty=2,xlim=c(1,(nc+1)),ylim=
c(1,s),axes=F) # plota a riqueza em função do número de ciclos rodados para
todas as simulações
    axis(1,at=c((-(nc+1)):(2*(nc+1))),labels=c((-(nc+2)):((2*(nc+1))-1))) #
plota eixo x do gráfico
    axis(2,at=c(-1:(2*s)),labels=c(-1:(2*s))) # plota eixo y do gráfico
    axis(3,at=c((-(nc+1)),(2*(nc+1))),labels=c("","")) # plota borda
superior do gráfico
    axis(4,at=c(-1,2*s),labels=c("","")) # plota borda direita do gráfico
  ## Retornando aos parâmetros usuais
    par(mfrow=c(1,1)) # retorna os parâmetros gráficos para o default do R
  ## Mensagem de aviso acerca dos resíduos gerados para construção do
gráfico
    warning("A fim de evitar sobreposição de linhas ou pontos com mesmo
valor de x e/ou y nos gráficos, estes foram plotados a partir de um resíduo
aleatório gerado para cada valor original de x ou y.") # retorna mensagem de
aviso acerca dos resíduos associados aos valores de x ou y nos gráficos
  ## Outputs
    ### Opção com retorno do array
      if(details==T){ # retorna o array "resultado.array"
      resulta=list("Média da riqueza final"=s.final.med,"Indentidade dos
indivíduos ao final de cada ciclo"=resultado.array) # cria uma lista,
chamada "resulta", com a média da riqueza final e com o array
"resultado.array"
      return(resulta) # retorna a lista "resulta"
      }
    ### Opção sem retorno do array
      if(details==F){ # não retorna o array "resultado.array"
      cat("Média da riqueza final\n") # retorna no console a frase "Média da
riqueza final"
      return(s.final.med) # retorna a média da riqueza final
      }
  }
} # Fim da função hubbells.game()
```

Arquivos da Função

Código de hubbells.game Help de hubbells.game

From:
http://ecor.ib.usp.br/ - **ecoR**

Permanent link:
http://ecor.ib.usp.br/doku.php?id=05_curso_antigo:r2015:alunos:trabalho_final:luisa.goncalves:start

Last update: **2020/08/12 06:04**