# Renato Chaves de Macedo Rego

Mestrando em Ecologia, Instituto de Biociências, USP.

### Exercícios

exec

### Propostas para o trabalho final

Proposta A:

Em meu projeto de Iniciação Científica, desenvolvi experimentos para avaliar a interação entre machos e fêmeas de uma espécie de aranha. Para tanto, necessitei produzir séries de casais conforme os testes foram realizados. A cada rodada de testes, cada indivíduo entrou em contato com somente um outro indivíduo, sendo os dois obrigatoriamente de sexos opostos. De modo a tornar esta definição de casais mais adequada, eu tive que desenvolvê-la de forma completamente aleatória. Embora, a situação pareça ser resolvida com um simples sorteio entre machos e fêmeas, os resultados da primeira rodada de testes tinham implicações sobre a segunda rodada, uma vez que todos os casais gerados deveriam ser diferentes dos anteriores e eu deveria aproveitar o máximo possível os animais que tinha a disposição. Explicando melhor, eu não poderia correr o risco de misturar os casais A, B, C, D e E da primeira rodada de testes e deixar o casal F novamente junto (pois este casal não poderia participar dos experimentos, e o n amostral seria menor do que o possível com a oferta de animais). Esta situação se assemelha muito à criação de tabelas de campeonatos esportivos, em que todos os times de um grupo jogam contra todos os times de outro grupo, sem que qualquer duelo se repita. Imagino que a necessidade de se produzir estas tabelas ocorra em várias outras instâncias, inclusive em trabalhos experimentais como o que desenvolvi, justificando-se o desenvolvimento de uma função para produção destes “casais”/ “confrontos”. A idéia é que baste ao usuário do R determinar quais são os objetos a serem combinados e definir a qual variável cada um deles pertence (ex.: posicionando as fêmeas em uma das duas listas disponíveis na função e dispondo os machos na outra lista). A função geraria as combinações randomicamente, quantas vezes fossem solicitadas pelo usuário. Sendo importante ressaltar que há um limite de rodadas, pois, se há 20 machos e 20 fêmeas, cada animal só pode formar casal com 20 outros indivíduos. Se o usuário pedir que a função gere 21 rodadas, ocorrerá repetição de combinações.

Proposta B:

Gerar uma função que permita ao usuário jogar vinte-e-um contra o R. Para isto, eu necessitaria criar um objeto em que estão concatenadas todas as cartas do baralho. Do montante disponível, a função sortearia alguns valores a cada rodada, que são as cartas do jogador (usuário) e da mesa (R). Cada rodada representa um ponto para o vencedor (empates são nulos). O usuário tem a opção de dobrar a aposta (vitória=2 pontos), antes que a próxima carta seja sorteada e defina o vencedor. Após o usuário responder ao comando “levantar aposta ou não”, a carta é sorteada e uma simples soma dos valores das cartas na mesa define o vencedor. Claro que para tudo isso, é necessário atribuir valores

numéricos às cartas com figuras. Conforme o avanço das rodadas, soma-se o placar no duelo entre R e Usuário.

#### Comentários das propostas (Leo)

As propostas são interessantes. Considero a proposta A mais útil e simples que a primeira. Na verdade existem diferenças entre uma tabela de campeonato de futebol e a sua proposta (de parear casais). Em um campeonato você deve parear uma única lista entre ela mesma. Enquanto nos casais você tem duas listas diferentes. Para resolver isto basta um argumento onde o usuário especifica quais são os objetos que serão pareados (x = ..., y = ...). O desafio será criar um algoritmo que evite que o pareamento se repita em diferentes rodadas., ou no caso do campeonato de futebol, que algum time forme um par com ele mesmo.

Como dados de entrada a função receberá uma ou duas listas de igual tamanho. E como função de saída? Será uma lista com os pares para cada rodada?

#### Resposta (Renato)

Sim, a proposta é produzir uma lista com os pares para cada rodada.

#### Atualizando Resposta (Renato)

Com o tempo, percebi que a função ficaria melhor se ela retornasse uma matriz com os pareamentos, pois assim fica bem mais fácil de o usuário manipular os resultados. Desenhei o código de um modo que fica bem direto explicitar e delimitar que resultados o programa vai gerar, mesmo quando o usuário utilizar amostras muito grandes. Extrapolei um pouco a proposta inicial. A função que desenvolvi também permite a entrada de listas de tamanhos diferentes. A função randomiza a distribuição dos elementos de uma lista (argumento y) em função dos elementos da outra lista (argumento x). Caso o usuário queira aleatorizar a ordem dos elementos de ambas as listas, basta usar a função sample sobre a lista que será o argumento 'x' e, após isto, usar a função pareamento. A decisão de aleatorizar somente a ordem dos elementos do argumento y foi consciente (fazer com que a função aleatorizasse os dois conjuntos seria muito simples). Se eu aleatorizasse os elementos de ambas as listas, ficaria muito mais difícil para o usuário acessar determinados pareamentos mais desejados, até porque, muitas vezes, a disposição dos elementos nas listas apresenta uma sequência lógica. Se eu randomizo ambas as sequências, fica mais difícil para o usuário se localizar, ainda mais quando os comprimentos dos vetores são muito grandes.

#### Trabalho Final - Página de Ajuda

```
 pareamento                package: nenhum           R Documentation

Produz pareamentos aleatórios entre os elementos de dois vetores fornecidos
pelo usuário.
```

```
Descrição: a função gera pareamentos randômicos, retornando o resultado
através de uma matriz.
Cada pareamento envolve um elemento do argumento ‘x’ e um elemento do
argumento ‘y’.
As combinações são produzidas independentemente da diferença dos
comprimentos de ‘x’ e ‘y’.

Uso:
pareamento(x,y ...)

## Default method:
pareamento(x, y, linhas=1:length(x), rodadas=1:length(y))

Argumentos:

x - uma lista ou matriz com o primeiro conjunto de elementos que serão
pareados pela função. Deve ter comprimento superior a 1.
y - uma lista ou matriz com o segundo conjunto de elementos que serão
pareados pela função. Deve ter comprimento superior a 1.
linhas – argumento que especifica os elementos de ‘x’ (as linhas da matriz)
que serão retornados pela função.
rodadas – argumento que especifica as rodadas de pareamento (colunas da
matriz) que serão retornadas pela função.
nomes.linhas – opcional para nomeação complementar das linhas da matriz
produzida.
No modo default, as linhas são nomeadas somente pelos elementos de‘x’, na
ordem em que eles foram fornecidos.

Valores:

A função retorna uma matriz em que as linhas são representadas pelos
elementos que compõem o vetor ‘x’ fornecido e as colunas representam as
rodadas de pareamentos produzidos.
Os elementos da matriz resultante são os mesmos do vetor ‘y’ fornecido, mas
dispostos randomicamente.
Caso ‘x’ tenha comprimento superior ao de ‘y’, a diferença de comprimento
dos dois é representada pelo acréscimo de linhas nulas (“NULL”) à matriz.
Se ‘y’ for maior, a diferença de comprimento dos vetores ‘x’ e ‘y’ é
representada pela presença de elementos vazios "Ø" na matriz.

Autor:

Renato Chaves de Macedo Rego
renato.rego@usp.br

Exemplos:

## 'x' e 'y' de mesmo comprimento, sem especificar os outros argumentos
x=c("a","b","c","d","e","f","g","h","i","j")
y=c("k","l","m","n","o","p","q","r","s","t")
pareamento(x,y)
```

```
## length(x) > length(y) e argumento 'rodadas' especificado
pareamento(x=c("a","b","c","d","e","f","g","h","i","j"),y=c("k","l","m","n",
"o","p","q","r"), rodadas=1:4)

## length(x) < length(y) e argumentos 'linhas' e 'nomes.linhas'
especificados
x=c(1,2,3,4,5,6,7)
y=c(11,12,13,14,15,16,17,18,19,20)
pareamento(x,y, linhas=1:7, , "femeas")  #ou
pareamento(x,y, linhas=1:length(x), , "femeas")

## pareamento dos elementos de duas matrizes
pareamento(matrix(1:12,nrow=3,ncol=4),matrix(1:12,nrow=4,ncol=3))
```

**Trabalho Final - Código da Função**

```
 ############ # Função de Pareamento
pareamento= function(x,y,linhas,rodadas,nomes.linhas)
{
if(class(x)=="data.frame"|class(y)=="data.frame")
{
return("Erro: x e y podem ser uma lista de determinado dataframe, mas não
todo dataframe")
}
if(length(x)==length(y)&missing(nomes.linhas)&length(x)>1)
{
x=as.character(x)
y=as.character(y)
sorteio=sample(y,replace=FALSE)
matrizparcial=matrix(sorteio,nrow=(length(x)-1),ncol=(length(y)))
inversa=rev(sorteio)
matriz=rbind(matrizparcial,inversa)
colnames(matriz)=paste("rodada",1:length(y))
rownames(matriz)=paste(x)
return(matriz[linhas,rodadas])
}
if(length(x)==length(y)&!missing(nomes.linhas)&length(x)>1)
{
x=as.character(x)
y=as.character(y)
sorteio=sample(y,replace=FALSE)
matrizparcial=matrix(sorteio,nrow=(length(x)-1),ncol=(length(y)))
inversa=rev(sorteio)
matriz=rbind(matrizparcial,inversa)
colnames(matriz)=paste("rodada",1:length(y))
rownames(matriz)=paste(nomes.linhas,1:length(x))
return(matriz[linhas,rodadas])
}
```

```
if(length(x)>length(y)&missing(nomes.linhas)&length(y)>1)
{
x=as.character(x)
y=as.character(y)
y2=paste(rep("Ø", abs(length(x)-length(y))),1:abs(length(x)-length(y)))
y3=union(y,y2)
sorteio=sample(y3,replace=FALSE)
matrizparcial=matrix(sorteio,nrow=(length(x)-1),ncol=(length(y3)))
inversa=rev(sorteio)
matriz=rbind(matrizparcial,inversa)
colnames(matriz)=paste("rodada",1:length(y3))
rownames(matriz)=paste(x)
return(matriz[linhas,rodadas])
}
if(length(x)>length(y)&!missing(nomes.linhas)&length(y)>1)
{
x=as.character(x)
y=as.character(y)
y2=paste(rep("Ø", abs(length(x)-length(y))),1:abs(length(x)-length(y)))
y3=union(y,y2)
sorteio=sample(y3,replace=FALSE)
matrizparcial=matrix(sorteio,nrow=(length(x)-1),ncol=(length(y3)))
inversa=rev(sorteio)
matriz=rbind(matrizparcial,inversa)
colnames(matriz)=paste("rodada",1:length(y3))
rownames(matriz)=paste(nomes.linhas,1:length(x))
return(matriz[linhas,rodadas])
}
if(length(x)<length(y)&missing(nomes.linhas)&length(x)>1)
{
x=as.character(x)
y=as.character(y)
x2=paste(rep("NULL", abs(length(x)-length(y))),1:abs(length(x)-length(y)))
x3=union(x,x2)
sorteio=sample(y,replace=FALSE)
matrizparcial=matrix(sorteio,nrow=(length(x3)-1),ncol=(length(y)))
inversa=rev(sorteio)
matriz=rbind(matrizparcial,inversa)
colnames(matriz)=paste("rodada",1:length(y))
rownames(matriz)=paste(x3)
return(matriz[linhas,rodadas])
}
if(length(x)<length(y)&!missing(nomes.linhas)&length(x)>1)
{
x=as.character(x)
y=as.character(y)
x2=paste(rep("NULL", abs(length(x)-length(y))),1:abs(length(x)-length(y)))
x3=union(x,x2)
sorteio=sample(y,replace=FALSE)
matrizparcial=matrix(sorteio,nrow=(length(x3)-1),ncol=(length(y)))
inversa=rev(sorteio)
```

```
matriz=rbind(matrizparcial,inversa)
colnames(matriz)=paste("rodada",1:length(y))
rownames(matriz)=paste(nomes.linhas,x3)
return(matriz[linhas,rodadas])
}
if(length(x)<2|length(y)<2)
{
return("Erro: x e y devem ser vetores com length>1")
}
}
```

### Arquivo da Função

funcao_pareamento.r

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

Permanent link:
**http://ecor.ib.usp.br/doku.php?id=05_curso_antigo:r2013:alunos:trabalho_final:renato.rego:start**

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