Funções scacorum

```
#####Código das três funções scacorum#####

#A continuação vamos a apresentar o código das três funções scacorum
(scacorum.sim, scacorum.calc e scacorum.graf)

#Começamos pela primeira função, scacorum.sim para fazer as simulações
scacorum.sim=function(t,m,source)
{
  #Temos colocado um limite de 100 simulações (t) e 1000 movimentos, para
que os cálculos sejam relativamente rápidos, e também para criar uma saída
gráfica bonita com scacorum.graf
  #porem, sempre podem se trocar esses limites modificando o seguinte objeto
  t.limite=100
  #O número t deve ser um número inteiro entre 1 e t.limite
  if(is.element(t,c(1:t.limite))==FALSE )
   #se não for assim para a função e envia uma mensagem de erro
  {
    stop(paste("t deve ser um numero inteiro entre 1 e ", t.limite))
  }
  #A mesma coisa para o número de movimentos (m)
  #criamos o limite em 1000
  m.limite=1000
  #...m deve ser um numero inteiro entre 1 e o limite
  if(is.element(m,c(1:m.limite))==FALSE )
   #Se não para a função e envia uma mensagem de erro
  {
    stop(paste("m deve ser um numero inteiro de 1 até ", m.limite))
  }
  #Se falta o argumento source procede a simulará tanto as trajetórias do
rei quanto as do cavalo, que é o seguinte que vamos programar
  if(missing(source))
  {
 #Agora vamos carregar os movimentos primários:
    #Primeiro os 8 movimentos primários do rei,
    #o vetor da x
    mov.prim.x.rei=c( 0, 1, 1, 1, 0, -1,-1,-1)
    #e o vetor da y
    mov.prim.y.rei=c(-1,-1, 0, 1, 1, 1, 0,-1)
    #E juntados numa matriz, lembremos que os movimentos primários são
direções, cada uma das 8 filas nos indica se o rei vai subir, permanecer ou
descer (x),
    #e se vai para a esquerda, fica no centro, ou vai para a direita (Y)
    movimentos.prim.rei=as.matrix(cbind(mov.prim.x.rei,mov.prim.y.rei))
    #Agora o cavalo, primeiro o vetor da x
    mov.prim.x.cav=c(1, 2, 2,   1, -1, -2, -2, -1)
    #depois o vetor da y
    mov.prim.y.cav=c(-2, -1, 1, 2, 2,    1, -1, -2)
```

```
    #... juntados numa matriz
    movimentos.prim.cav=as.matrix(cbind(mov.prim.x.cav,mov.prim.y.cav))
 #Agora criamos uma matriz cheia de NA, onde colocaremos t vetores de
tamanho m
    ordem.mov.sim=matrix(NA, ncol=t, nrow=m)
   for(i in 1:t)
    {
       #A cada ciclo criamos um vetor com o comprimento do numero de
movimentos (m), enchido aleatoriamente com números do 1 ao 8, cada número
representa um movimento primário.
      #Esse será o vetor que direcione tanto a trajetória do rei, quanto a
do cavalo. O número de movimentos e m-1 porque vamos considerar o primeiro
movimento o estado inicial (0,0), que será colocado mais na frente
      ordem.mov=sample(c(1:length(mov.prim.x.rei)),m-1, replace=T)
     #Colocamos o primeiro vetor na segunda posição da matriz (deixamos
reservado espaço para o movimento inicial)
       ordem.mov.sim[2:m,i]=ordem.mov
    }
    #Agora vamos a criar as trajetórias do rei e do cavalo, traduzindo os
números (1 ao 8) nos movimentos primários de cada peça
    #Começamos pela trajetória do rei
    #O seguinte vetor traduz cada número de movimento primário ao
correspondente movimento de x
    rei.mov.x=matrix(movimentos.prim.rei[ordem.mov.sim,1], nrow=m)
    #A mesma coisa com a y
    rei.mov.y=matrix(movimentos.prim.rei[ordem.mov.sim,2], nrow=m)
    #Agora vamos colocar o ponto inicial (0,0) em todas as trajetórias
    #Um 0 na primeira posição da x
    rei.mov.x[1,]=rep(0,t)
    #Outro 0 na primeira posição da y
    rei.mov.y[1,]=rep(0,t)
     #Para obter as coordenadas temos que somar os movimentos da x e da y.
    #somamos acumulativamente as x
    rei.x=apply(rei.mov.x,2,FUN = cumsum)
    #somamos acumulativamente as y
    rei.y=apply(rei.mov.y,2,FUN = cumsum)
     #Agora faremos o mesmo processo com o cavalo
    #Traduzimos cada número de movimento primário ao correspondente
movimento de x e y
    #Tradução da x
    cav.mov.x=matrix(movimentos.prim.cav[ordem.mov.sim,1], nrow=m)
    #Tradução da y
    cav.mov.y=matrix(movimentos.prim.cav[ordem.mov.sim,2], nrow=m)
    #Colocamos o ponto inicial da x
    cav.mov.x[1,]=rep(0,t)
    #Colocamos o ponto inicial da y
    cav.mov.y[1,]=rep(0,t)
    #E obtemos as coordenadas da x e da y.
    #somando acumuladamente as x
    cav.x=apply(cav.mov.x,2,FUN = cumsum)
```

```
    #e somando acumuladamente as y
    cav.y=apply(cav.mov.y,2,FUN = cumsum)
    #Agora temos que calcular as medidas espaciais, é importante ressaltar
que as medidas devem poder ser extraídas com qualquer quantidade de
movimentos
   ###Vamos começar a programar todas as medidas do rei e depois
procederemos igual com o cavalo
    #Para calcular o número de escaques repetidos, o primeiro vamos fazer um
vetor para reconhecer cada escaque, juntando x e y num único elemento
    ponto.nick.rei=matrix(paste0(rei.x,"/",rei.y),nrow=m)
    #Agora, faremos um vetor que indique, a cada movimento, quantas vezes o
escaque presente foi ocupado até esse momento da trajetória.
    #Utilizamos a função ave, que divide um conjunto de dados segundo as
categorias de outros conjuntos de dados, e calcula alguma operação
matemática no conjunto de dados resultado dessa divisão amostral
    #primeiro criamos uma matriz do tamanho das nossas simulações (t*m)
cheia de "1", esse é o conjunto de dados que a função ave vai dividir e
aplicar a operação
    #E vamos dividir em subconjuntos com respeito a duas condições:
    #1)"ponto.nick.rei", ou seja fará subconjuntos com os escaques com mesmo
"nick"
    #2)matrix(rep(1:t,each=m), essa matriz ajuda a diferenciar de uma
trajetória simulada para outra, tem um 1 em todos os movimentos da primeira
trajetória, um 2 em todos os movimentos da segunda, etc...
    #Assim, a primeira matriz (cheia de "1") é dividida por nick, e por
trajetória, e realizamos um cumsum. Ou seja obtemos o acumulado de "1"
(primeira matriz), que cumpra as duas condições de agrupação indicadas
    #O resultado é interessante, como falado, nos indica  em cada simulação
(coluna), e em cada movimento (linha) quantas vezes foi ocupada o escaque
presente até esse movimento da trajetória.
    rep.eachmov.rei=ave(matrix(1,nrow = m,
ncol=t),ponto.nick.rei,matrix(rep(1:t,each=m),ncol=t),FUN=cumsum)
    #Porem, o nosso interesse é saber, em cada trajetória e a cada
movimento, o acumulado de escaques que foram repetidos até esse momento da
trajetória
    #Por lógica, os escaques repetidos devem ter um número maior de 1 em
"rep.eachmov.rei",
    #assim, utilizamos a mesma fórmula, ave, sobre "rep.eachmov.rei", mas
somente naqueles escaques nos quais 1> es verdadeiro (true),
    #dividimos essa matriz a partir da mesma matriz anterior criada para
reconhecer as trajetórias simuladas diferentes,
    #e aplicamos cumsum aos subconjuntos:
    r.rei=ave(rep.eachmov.rei>1, matrix(rep(1:t,each=m),ncol=t),FUN=cumsum)
    #Continuamos com o projeto
    ## "3)Área líquida ocupada (l): ...é o resultado desta operação: l=m+1-
r."
    #Temos modificado com respeito ao projeto original, e vamos considerar o
primeiro movimento a posição 0, e assim não devemos somar 1 ao numero de
movimentos (m)
    #Criamos um vetor com números do 1 até m
    vetor.m=seq(1,m)
```

```
    #E restamos o número de escaques repetidos, para obter a área líquida a
cada movimento
    l.rei=vetor.m-r.rei
    ## "4)N° de escaques do mínimo tabuleiro retangular que compreende a
trajetória (tab): O mínimo tabuleiro é o resultado da fórmula: tab=(max(x)-
min(x)+1)*(max(y)-min(y)+1)"
    #No projeto já explicamos porque era preciso somar 1 a cada lado do
retângulo: não estamos trabalhando com portos, se não com escaques, e temos
que considerar o tamanho do primeiro e o último escaque.
    #O primeiro vamos a procurar o mínimo valor de x em cada movimento,
durante a trajetória do rei
    #vamos a utilizar de novo a função ave:
    #Cogemos "rei.x", que contem as coordenadas x do rei, e dividimos a
amostra entre a matriz indicadora de trajetória simulada,
    #e aplicamos cummin a cada subgrupo, cummin é o acumulado de valores
mínimos (o valor mínimo até esse momento da trajetória)
     min.x.rei=ave(rei.x,matrix(rep(1:t,each=m),ncol=t),FUN=cummin)
    #Agora precisamos o valor máximo de x na trajetória do rei, fazemos a
mesma coisa, mas com a função cummax
    max.x.rei=ave(rei.x,matrix(rep(1:t,each=m),ncol=t),FUN=cummax)
    #Agora fazemos a mesma coisa com a coordenada y da trajetória do rei
    #primeiro a y mínima
    min.y.rei=ave(rei.y,matrix(rep(1:t,each=m),ncol=t),FUN=cummin)
    #depois a y máxima
    max.y.rei=ave(rei.y,matrix(rep(1:t,each=m),ncol=t),FUN=cummax)
    #Agora já temos os argumentos para calcula o mínimo tabuleiro
rectangular (tab), segundo a fórmula descrita
    tab.rei=(max.x.rei-min.x.rei+1)*(max.y.rei-min.y.rei+1)
    #Seguimos com o projeto
    ## "5)Medidas proporcionais da área: Com o objetivo de comparar as
distintas medidas de área propostas consideramos interessantes os seguintes
índices:
    ##-% Área líquida/máx. teórico de área líquida:(l*100)/m
    rei.prop.1=matrix((l.rei*100)/vetor.m,ncol=t)
    ##%Área líquida/mín. tabuleiro: (l*100)/tab
    rei.prop.2=matrix((l.rei*100)/tab.rei,ncol=t)
    ##% Máx. teórico de área líquida/mín. tabuleiro: (tab*100) /m Aqui temos
invertido a equação, considerando melhor após observar as saidas do boxplot
de scacorum.calc
    rei.prop.3=matrix((tab.rei*100)/vetor.m,ncol=t)

    ###Procedemos igual com o cavalo
    #Vetor para reconhecer cada escaque
    ponto.nick.cav=matrix(paste0(cav.x,"/",cav.y),nrow=m)
    #Agora, faremos o vetor que indica em cada simulação (coluna), e em cada
movimento (linha) quantas vezes foi ocupada o escaque presente até esse
movimento da trajetória.
    rep.eachmov.cav=ave(matrix(1,nrow = m,
ncol=t),ponto.nick.cav,matrix(rep(1:t,each=m),ncol=t),FUN=cumsum)
    #E, o numero de escaque repetidos a cada movimento
```

```
    r.cav=ave(rep.eachmov.cav>1, matrix(rep(1:t,each=m),ncol=t),FUN=cumsum)
    #Área líquida ocupada
    l.cav=vetor.m-r.cav
    #Coordenada x
    #O acumulado de valores mínimo (o valor mínimo até esse momento)
    min.x.cav=ave(cav.x,matrix(rep(1:t,each=m),ncol=t),FUN=cummin)
    #O valor máximo
    max.x.cav=ave(cav.x,matrix(rep(1:t,each=m),ncol=t),FUN=cummax)
    #Coordenada y
    #mínima
    min.y.cav=ave(cav.y,matrix(rep(1:t,each=m),ncol=t),FUN=cummin)
    #máxima
    max.y.cav=ave(cav.y,matrix(rep(1:t,each=m),ncol=t),FUN=cummax)
    #Mínimo tabuleiro rectangular (tab)
    tab.cav=(max.x.cav-min.x.cav+1)*(max.y.cav-min.y.cav+1)
    ##-% Área líquida/máx. teórico de área líquida:(l*100)/m
    cav.prop.1=matrix((l.cav*100)/vetor.m,ncol=t)
    ##%Área líquida/mín. tabuleiro: (l*100)/tab
    cav.prop.2=matrix((l.cav*100)/tab.cav,ncol=t)
    ##% Máx. teórico de área líquida/mín. tabuleiro: (tab*100) /m
    cav.prop.3=matrix((tab.cav*100)/vetor.m,ncol=t)
  #Agora vamos criar uma lista com 16 matrices com todos os resultados, 8
matrices do rei e 8 do cavalo, com as trajetórias (x e y) e com as medidas
espaciais
simulações=list(rei.x,rei.y,r.rei,l.rei,tab.rei,rei.prop.1,rei.prop.2,rei.pr
op.3, cav.x,cav.y,r.cav,l.cav,tab.cav,cav.prop.1,cav.prop.2,cav.prop.3)
  #Colocamos os nomes nas matrizes
names(simulações)=c("rei.x","rei.y","r.rei","l.rei","tab.rei","rei.%l/m","re
i.%l/tab","rei.%tab/m",
"cav.x","cav.y","r.cav","l.cav","tab.cav","cav.%l/m","cav.%l/tab","cav.%tab/
m")
    #E pedimos que nos devolva a lista
      return(simulações)

    }
  else
  #Esse else quer dizer: se o argumento source não é nulo, então...
    #No caso, a gente vai permitir que o usuário crie uma tabela com os
movimentos primários de uma peça inventada,
    #O usuário deve colocar um objeto com a tabela e lhe atribuir um nome, o
argumento source seria esse nome
    {
      #criamos uma matriz igual à das peças anteriores, vamos chamar
"invent" aos objetos criados.
      ordem.mov.sim.invent=matrix(NA, ncol=t, nrow=m)
      for(i in 1:t)
      {
        #vetor com o comprimento do numero de movimentos (m), enchido
aleatoriamente com um número de movimentos da matriz inventada.
        ordem.mov.invent=sample(c(1:length(source[1,])),m-1, replace=T)
        #Colocamos os vetores na segunda posição da matriz (deixamos
```

```
reservado espaço para o movimento inicial)
        ordem.mov.sim.invent[2:m,i]=ordem.mov.invent
         }
      #Agora vamos a traduzir os movimentos primários em trajetórias
      #O seguinte vetor traduz cada número de movimento primário ao
correspondente movimento de x. Ao criar a matriz o usuario devera colocar a
x na primeira coluna e a y na segunda.
      invent.mov.x=matrix(source[ordem.mov.sim.invent,1], nrow=m)
      #A mesma coisa com a y
     invent.mov.y=matrix(source[ordem.mov.sim.invent,2], nrow=m)
      #Agora vamos colocar o ponto inicial na sequencia de movimentos
     #O 0 inicial da x
     invent.mov.x[1,]=rep(0,t)
     #E o 0 inicial da y
      invent.mov.y[1,]=rep(0,t)
      #Para obter as coordenadas, agora temos que somar os movimentos da x e
da y.
      #da x
      invent.x=apply(invent.mov.x,2,FUN = cumsum)
      #e da y
     invent.y=apply(invent.mov.y,2,FUN = cumsum)
     #Agora as medidas espaciais
     #Vetor para reconhecer cada escaque
     ponto.nick.invent=matrix(paste0(invent.x,"/",invent.y),nrow=m)
     #Agora, faremos o vetor que indica em cada simulação (coluna), e em
cada movimento (linha) quantas vezes foi ocupado o escaque presente até esse
movimento da trajetória.
     rep.eachmov.invent=ave(matrix(1,nrow = m,
ncol=t),ponto.nick.invent,matrix(rep(1:t,each=m),ncol=t),FUN=cumsum)
     #E, o numero de escaque repetidos a cada movimento
     r.invent=ave(rep.eachmov.invent>1,
matrix(rep(1:t,each=m),ncol=t),FUN=cumsum)
     #Área líquida ocupada
     #Criamos um vetor do primeiro até o último movimento
     vetor.m=seq(1,m)
     #E calculamos a cada movimento a área líquida, restando a m os escaques
repetidos
     l.invent=vetor.m-r.invent
     #Continuamos com o mínimo tabuleiro, achamos os mínimos e máximos
valores da coordenada x
     #O acumulado de valores mínimos
     min.x.invent=ave(invent.x,matrix(rep(1:t,each=m),ncol=t),FUN=cummin)
     #O acumulado de valores máximos
     max.x.invent=ave(invent.x,matrix(rep(1:t,each=m),ncol=t),FUN=cummax)
     #E a mesma coisa com a Coordenada y
     #mínima
     min.y.invent=ave(invent.y,matrix(rep(1:t,each=m),ncol=t),FUN=cummin)
     #máxima
     max.y.invent=ave(invent.y,matrix(rep(1:t,each=m),ncol=t),FUN=cummax)
     #Mínimo tabuleiro rectangular (tab)
```

```
     tab.invent=(max.x.invent-min.x.invent+1)*(max.y.invent-min.y.invent+1)
     ##-% Área líquida/máx. teórico de área líquida:(l*100)/m
     invent.prop.1=matrix((l.invent*100)/vetor.m,ncol=t)
     ##%Área líquida/mín. tabuleiro: (l*100)/tab
     invent.prop.2=matrix((l.invent*100)/tab.invent,ncol=t)
     ##% Máx. teórico de área líquida/mín. tabuleiro: (m *100) /tab
     invent.prop.3=matrix((tab.invent*100)/vetor.m,ncol=t)
     #Criamos uma lista com 8 tabelas com as coordenadas e com as medidas
espaciais da peça inventada
simulações.invent=list(invent.x,invent.y,r.invent,l.invent,tab.invent,invent
.prop.1, invent.prop.2,invent.prop.3)
      #Colocamos os nomes nas matrices
names(simulações.invent)=c("invent.x","invent.y","r.invent","l.invent","tab.
invent","invent.%l/m","invent.%l/tab","invent.%tab/m")
      #E pedimos que retorne a lista
       return(simulações.invent)
  #Fechamos a parte "invent" da função
  }
#E fechamos a toda a função scacorum.sim
}


#########################################################################



#Agora vamos a programar scacorum.calc, para utilizar essa função o usuário
previamente deve ter desenvolvido as simulações com scacorum.sim e criado um
objeto com elas,
#O nome desse objeto deverá ser introduzido no primeiro argumento (dados) de
scacorum.calc
#Com o resto de argumentos selecionamos uma medida espacial, escolhemos que
estatístico operar, e configuramos a amostra que vamos a analisar
scacorum.calc=function(dados,nm,surfmeas,estimator,intra.inter,peça)
#Abrimos a função
  {
 #Criamos um objeto indicando o límite no número de movimentos (nm), que
será o número máximo de movimentos simulados no objeto "dados" (m)
  nm.limite=length(dados[[1]][,1])
   #Colocamos um condicional com todas as possibilidades de nm estar errado
  if(is.element(nm,c(1:nm.limite))==FALSE )
  {
    #E indicamos que, caso estiver errado interrompa a função e envie uma
mensagem de erro
    stop(paste("m deve ser um inteiro maior de 1 e menor do número máximo de
movimentos simulados (", nm.limite,")"))
  }
  #Fazemos a mesma coisa com as medidas de superfície, que tem 6 opções
(r,l,tab,l/m,l/tab,tab/m)
  if(is.element(surfmeas,c(1:6))==FALSE )
  {
```

```
    #...interrupção e mensagem de erro
    stop("surfmeas deve ser um número inteiro do 1 ao 6")
  }
  #Temos 4 opções de estimador, média, sd, min, max
  if(is.element(estimator,c(1:4))==FALSE )
  {
    #...interrupção e mensagem de erro
    stop("estimator deve ser um número inteiro do 1 ao 4")
  }
  #intra (1) significa que o estatístico é calculado entre as trajetórias a
cada movimento,inter (2) significa que o estatístico é calculado sobre o
total de movimentos de cada trajetória (em principio parece contra-
intuitivo, mas depois quando vejam o boxplot vão ver porque chamei intra e
inter)
  if(is.element(intra.inter,c(1:2))==FALSE )
  {
   #E a função para se não for o 1 ou 2
     stop("intra.inter deve ser ou o 1 ou o 2")
  }
  #Temos três opções de peças, 1) rei, 2) cavalo, 3)peça inventada
  if(is.element(peça,c(1:3))==FALSE )
  {
    #...interrupção e mensagem de erro
    stop("peça deve ser um número inteiro do 1 ao 3")
  }
  #Tanto se a peça é 1 (rei) quanto se for 3 (peça inventada), os dados vão
estar nas primeiras 8 matrices da lista do objeto "dados"
  if(peça==1|peça==3)
  {
    #Como as duas primeiras matrices contem a trajetória da peça (x e y),
temos que somar 2 ao valor de surfmeas
  dados=dados[[surfmeas+2]][1:nm,]
  }
  #Se a peça for 2 (cavalo), a informação figura nas 8 segundas tabelas,
assim a tabela 9 vai ter a coordenada x da trajetória do cavalo, a 10 a
coordenada y, e da 11 à 16 as medidas espaciais do cavalo
  if(peça==2)
  {
    #Como falado, as medias espaciais do cavalo acham-se depois da matriz 10
da lista de dados
    dados=dados[[surfmeas+10]][1:nm,]
  }
  #Agora o usuario deve ter colocado corretamente os argumentos, e
scacorum.calc já sabe com que amostra trabalhar
 #Criamos um data.frame muito particular, tem quatro componentes, cada um
deles é um apply pre-programado para aplicar cada um dos quatro estatísticos
  #O argumento intra.inter nos indica que tipo de cálculo marginal vai
fazer, por filas (1), ou por colunas (2), isso é precisamente o que queremos
result=data.frame(apply(dados,intra.inter,FUN=mean),apply(dados,intra.inter,
FUN=sd),apply(dados,intra.inter,FUN=min),apply(dados,intra.inter,FUN=max))
```

```
  #O apply que vai calcular está na posição do data.frame que indica o
argumento "estimator"
  result.sim=result[estimator]
 #Agora, temos um vetor com todos os possíveis cálculos
  #Agora vamos criar objetos com caracteres com a informação dos cálculos
estatísticos relevante para o usuário
  #O sumário
  result.summ=summary(result.sim)
  #O desvio padrão
  desvpad=sd(result.sim[,1])
  #Também queremos saber as posições que ocupam os valores mais baixos.
Vamos colocar primeiro os valores NA, na verdade, até agora nunca apareceram
em nenhuma prova, mas sempre é bom que sejam visíveis caso existissem
  prim.val=head(order(result.sim[,1], decreasing = F,na.last =T))
  #os valores mais altos
  ult.val=head(order(result.sim[,1], decreasing = T,na.last =F))
  #e o/s valor/es da mediana, com a seguinte equação:
  #Qual ou quais (which) valor/es tem a menor diferença com a mediana
  mediana=which (abs(result.sim[,1]-
median(result.sim[,1]))==min(abs(result.sim[,1]-median(result.sim[,1]))))
  #Agora vamos criar vetores com caracteres referentes aos argumentos,
colocados na ordem certa para ser chamados e formar frases coerentes, que
ajudem a entender os resultados
  #Precisamos criar frases no título, e no boxplot
  #As medidas espaciais
  medidas=c("escaques repetidos","area líquida","min
tabuleiro","l/m","l/tab","tab/m")
 #Os estatísticos
   estim=c("média","sd","min","max")
  #As peças
   peças=c("rei", "cavalo","peça inventada")
   #O seguinte indicará se o boxplot deve considerar as filas ou as colunas,
para que coincida o gráfico com o argumento intra-inter
  intr.entr=c(FALSE,TRUE)
  #O seguinte vetor é útil para o título do eixo x
  xclavedim=c("n movimento","n simulação")
  #O seguinte vetor é para construir o título, que fique claro que significa
ter escolhido entre intra ou entre
  intra.entre.clave=c("a cada movimento, com ", "entre trajetórias
simuladas, com ")
  #O seguinte é o numero total de simulações (lembrem que o numero total de
movimentos simulados já foi calculado, nm.limite)
  num.simulaçoes=length(dados[1,])
  #Agora criamos o título colando os diferentes caracteres dos vetores
segundo os argumentos selecionados
  titulo=paste("Peça:",peças[peça],",", estim[estimator],"de",
medidas[surfmeas],intra.entre.clave[intra.inter], nm," de ",nm.limite, "
movimentos, e com ",num.simulaçoes, "trajetórias simuladas")
 #Abrimos a tela de gráficos
  x11()
  #E pedimos um boxplot, que permite ter uma ideia de como se distribuem os
```

```
valores da medida espacial calculada
  #É importante assinalar, que sendo iguais o resto de argumentos, o gráfico
não vai mudar trocando o estatítisco
  #Observando a programação do boxplot vemos que os títulos dos eixos mudam
com respeito a escolha de argumentos do usuário
  graf.estim=boxplot(dados,use.cols = intr.entr[intra.inter],
xlab=paste(xclavedim[intra.inter],"(",peças[peça],")"),ylab=
medidas[surfmeas],las=1)
  #pedimos que nos devolva o título, o sumário, o desvio padrão, as posições
dos valores mínimos, máximos e mediana
  return (list( título=titulo, sumário=result.summ,
desvio_padrão=desvpad,posição_min_val=prim.val,posição_max_val=ult.val,posiç
ão_median_val=mediana))
  #E fechamos a função scacorum.calc
}

##########################################################################



#Por último vamos programar scacorum.graf, para ver as trajetórias simuladas
#O argumento dados é um objeto criado com scacorum.sim, que contem
trajetórias simuladas
#Com o resto de argumentos escolhemos qual das trajetórias simuladas
queremos ver (nt), com quantos movimentos(nm) do total de moviementos
simulados,de qual peça(peça) e se queremos observar o caminho que une os
movimentos (traj)
scacorum.graf=function(dados,nt,nm,traj,peça)
{
 #de novo, o número limite de trajetórias simuladas
  nt.limite=length(dados[[1]][1,])
  #nt deve ser um numero inteiro entre 1 e o número máximo de trajetórias
simuladas em "dados"
    if(is.element(nt,c(1:nt.limite))==FALSE )
  {
    #Se não for assim, interrompe a função, e envia mensagem de erro
      stop(paste("nt deve ser um numero inteiro, maior ou igual a 1 e menor
do que o número máximo de simulações criadas (", nt.limite,")"))
  }

  # A mesma coisa com o número de movimentos, achamos o limite
  nm.limite=length(dados[[1]][,1])
  #E criamos um condicional para os casos em que esteja mal escrito o
argumento
  if(is.element(nm,c(1:nm.limite))==FALSE )
  {
    #Se não for assim interrompe e manda mensagem de erro
    stop(paste("nm deve ser um numero inteiro, maior ou igual a 1 e menor do
que o número máximo de simulações criadas(", nm.limite,")"))
  }
```

```
  #A visualização da trajetória, sim (1), não (2)
  if(is.element(traj,c(1:2))==FALSE )
  {
    #Se não for 1 ou 2 interrompe e envia uma mensagem de erro
    stop("traj deve ser ou 1 (quero ver a trajetória), ou 2 (não quero)")
  }
  #A peça deve ser 1 (rei), 2 (cavalo), ou 3 (peça inventada)
  if(is.element(peça,c(1:3))==FALSE )
  {
    #Se não for assim interrompe e envia uma mensagem de erro
    stop("peça deve ser 1 (rei, 2 (cavalo), ou 3 (inventada)")
  }
  #Caso a peça seja o rei o peça inventada...
  if(peça==1|peça==3)
  {
    #as coordenadas estarão na primeira matriz de "dados" (x)
    xgraf=dados[[1]][1:nm,nt]
    #e na segunda matriz de "dados" (y)
    ygraf=dados[[2]][1:nm,nt]
  }
  #Se a peça é o cavalo
  if(peça==2)
  {
    #Lembremos, no cavalo a coordenada x está na matriz 9
    xgraf=dados[[9]][1:nm,nt]
    #A y na matriz 10
    ygraf=dados[[10]][1:nm,nt]
  }

  #Agora criamos dois conjuntos com os limites do gráfico
  #O primeiro conjunto, para o eixo da x, o valor mínimo -0,5, e máximo +
0.5, o motivo de somar meio ponto é, como sempre, porque trabalhamos com
escaques, não com pontos. As coordenadas são estimadas no centro do escaque,
devemos somar uma metade do primeiro escaque e outra metade do último
  xlimite= c(min(xgraf)-0.5,max(xgraf)+0.5)
  #o segundo cojunto e para o eixo da y
  ylimite= c(min(ygraf)-0.5,max(ygraf)+0.5)
  #Agora aplicamos um jitter, para evitar a sobreposição dos escaques
utilizados mais de uma vez
  xjitter=jitter(xgraf)
  #Aplicamos jitter tanto na x quanto na y, pois o resultado é mais
estético, na minha opinião
  yjitter=jitter(ygraf)

   #Abrimos a função TeachingDemos, com ela podemos criar um desenho, e
colocá-lo nos pontos de um plot
  library(TeachingDemos)

  #Para criar o símbolo do rei criei uma simples silhueta dentro de uma
quadrícula de 10X10
  #primeiro um polígono com a forma da peça do rei (desculpem o desenho, eu
```

```
não sou nenhum artista),
  #primeiro a x
  x.dsnh.rei=c(0,1,2,3,3,2,2,3,3,4,4,5,5,4,4,5,6,7,6,1,0)
  #e depois a y
  y.dsnh.rei=c(6,4,6,4,7,7,8,8,9,9,8,8,7,7,4,6,4,6,0,0,6)

  #A função que vamos utilizar"mysymbols", dentro do pacote "TeachingDemos"
exige que o polígono a plotar tenha as coordenadas entre -1 e 1,
  #assim, primeiro dividimos entre 10 as coordenadas
 #A coordenada x
   x.pad.rei=x.dsnh.rei/10
   #e a coordenada y
  y.pad.rei=y.dsnh.rei/10
#E agora vamos a centrar o desenho,cujos valores são todos positivos entre 0
e 1
  #Para as coordenadas x, primeiro restamos o valor mínimo da x, para que o
valor mínimo seja 0
  #depois restamos a metade do comprimento da figura (valor máximo menos
mínimo), para que fique bem centrado
  x.pad.centr.rei=x.pad.rei-min(x.pad.rei)-(max(x.pad.rei)-min(x.pad.rei))/2
  #E fazemos a mesma coisa com as coordenadas y
  y.pad.centr.rei=y.pad.rei-min(y.pad.rei)-(max(y.pad.rei)-min(x.pad.rei))/2
  #E criamos uma matriz com as coordenadas do poligono com a forma do cavalo
  simbolo.rei=as.matrix(cbind(x.pad.centr.rei,y.pad.centr.rei))
  #agora criamos o cavalo (dei o meu melhor...)
  #coordenada x
  x.dsnh.cav=c(1.8,2.3  ,2.3,2.7,2.7,3.2, 3.2, 3.1 ,2.9,2.7, 2.2  ,2, 1.95 ,
1.8,1.5,1.3, 1.5,2.1    ,2.5   ,3,2.8,5,6,      6.2  ,6  , 5.8,5.5,5.8
,5.8,6.3,6.3,1.8,1.8)
  #coordenada y
  y.dsnh.cav=c(  1,1  ,1.5,1.5 ,2  ,3   , 4 ,4.4  ,4.3,4.1, 3.6  ,3,  3.2 ,
3.1,3.4,4  , 5  ,6.3    ,7.1   ,8,9  ,7,5,     3.4  , 2.3 ,1.8 ,1.5  ,
1.5,1,  1,  0,0  ,1)
  #Colocamos as coordenadas entre 0 e 1
  #a x
  x.pad.cav= x.dsnh.cav/10
  #a y
  y.pad.cav=y.dsnh.cav/10
  #...centramos a x
  x.pad.centr.cav=x.pad.cav-min(x.pad.cav)-(max(x.pad.cav)-min(x.pad.cav))/2
  #a y
  y.pad.centr.cav=y.pad.cav-min(y.pad.cav)-(max(y.pad.cav)-min(x.pad.cav))/2
  #e colocamos numa matriz
  simbolo.cav=as.matrix(cbind(x.pad.centr.cav,y.pad.centr.cav))
  #Para a peça inventada vamos a fazer um simples quadrado
  #Vamos fazer o desenho diretamente centrado no 0
  #A x
   x.pad.centr.inv=c(-0.4,-0.4,0.4,0.4,-0.4)
   #E a y
  y.pad.centr.inv=c(-0.4,0.4, 0.4, -0.4,-0.4)
```

```
#E criamos uma matriz com as coordenadas (esse foi o símbolo que ficou
melhor)
  simbolo.inv=as.matrix(cbind(x.pad.centr.inv,y.pad.centr.inv))
 #Agora criamos uma lista com os três símbolos, que serão escolhidos
conforme indique o argumento "peça"
  simbolos=list(simbolo.rei,simbolo.cav,simbolo.inv)
  #O seguinte vetor será útil para a etiqueta da y, onde colocaremos o nome
da peça que está sendo representada
  peças=c("rei", "cavalo","peça inventada")
 #abrimos o visor gráfico
x11()
 #E pedimos o seguinte gráfico:
  #primeiro indicamos as coordenadas onde queremos colocar os símbolos
  #Depois indicamos a matriz onde ficam as coordenadas dos símbolos de cada
peça
  #Depois especificamos o tamanho que queremos os símbolos, medido no mesmo
formato do que as unidades do gráfico (assim o simbolo mudará de tamanho
conforme o tamanho total do gráfico)
  #add=F significa que não vamos a juntar essa figura a outra figura
previamente plotada
  #Marcamos os limites do gráfico (xlim ylim)
  #pty="s" para a visual do eixo x seja do mesmo tamanho que o eixo y
  #type="n" significa, sem mostrar nada na visual,
      #por que fazer isso? Esse plot vai ser a referencia de uma rede que
vamos plotar a continuação, depois voltaremos a plotar o gráfico acima da
rede, assim evitaremos que os símbolos fiquem tampados pela rede
  #Depois colocamos a explicação da peça no eixo da y e da trajetória
escolhida no eixo da x
  grafico=my.symbols(xjitter,yjitter,simbolos[[peça]][,1:2], xsize=2,
ysize=2,add=F,ylim=ylimite,xlim=xlimite,pty="s",type="n",xlab=paste("Trajeto
ria nº",nt,"de um total de",nt.limite, ", com",nm,"de",nm.limite,"movimentos
simulados"),ylab=peças[peça])
  #Agora faremos a rede que divide os escaques, primeiro varias abline na
vertical
  abline(v =seq (from=min(xlimite),to=max(xlimite),by=1),col="lightgray")
  #E depois na horizontal
 abline(h =seq (from=min(ylimite),to=max(ylimite),by=1),col="lightgray")
  #E plotamos de novo o gráfico, mas agora colocamos add=T, pois queremos
que adjunte no plot (invisível) acima, e não colocamos o modo invisível
(type="n")
  grafico=my.symbols(xjitter,yjitter,simbolos[[peça]][,1:2], xsize=2,
ysize=2,add=T,ylim=ylimite,xlim=xlimite,pty="s",xlab=paste("trajetoria",nt,"
de",nt.limite, "com",nm,"de",nm.limite,"movimentos
simulados"),ylab=peças[peça])

  #Por último, caso o usuário deseje ver o caminho que une os pontos
  if(traj==1)
  {
    #Adjuntamos umas líneas de movimento a movimento, desde o menor ao
máximo valor
    lines(xjitter, yjitter, xlim=range(xjitter), ylim=range(yjitter),
```

```
pch=16)
    #E reproduz o gráfico
  print(grafico)
  }
  #Se o usuário não colocou 1 no argumento traj, e porque colocou 2 (se
tivesse colocado cualquer outra coisa nesse argumento já teria recebido
previamente uma mensagem de erro)
  else
  {
    #Assim, se o usuário colocou 2 o gráfico já está pronto
    print(grafico)
}
}
```

Help das três funções

```
scacorum.sim                package:nenhum             R Documentation

~~ Simulação de trajetórias aleatórias do rei, do cavalo ou de uma peça
inventada de xadrez se movimentando num tabuleiro sem limites. ~~

Description:

     ~~scacorum.sim t cria trajetórias aleatórias de m movimentos do rei do
cavalo (versão default) ou de outra peça de xadrez que o usuário cujos
movimentos devem ser programados pelo usuário. Nessas simulações as peças se
movimentam num tabuleiro sem limites, e são tomadas 6 medidas espaciais a
cada movimento. A função simplesmente retorna tabelas matrizes com a
informação simulada~~

Usage:

     ~~ scacorum.sim (t, m, source)~~

Arguments:

 ~~ t: Número inteiro do 1 ao 100 indicando o número de trajetórias a
simular
m: Número inteiro do 1 ao 1000 indicando o número de movimentos de cada
trajetória. O primeiro movimento é considerado a posição inicial, sobre essa
posição são calculados m-1 movimentos.
source: Nome da matriz com os movimentos primários da peça inventada. Na
versão default (simulações do rei e do cavalo) não devemos colocar nada
nesse argumento. A matriz deve ter duas colunas de igual tamanho a primeira
com a coordenada x e a segunda com a coordenadas y.
Value:
```

```
     ~

Na versão default a função devolve 16 tabelas no console, as 8 primeiras com
a informação do rei, e as 8 segundas com a do cavalo, na versão “peça
inventada” somente devolve 8 tabelas.

  comp1 : Coordenada x da trajetória do rei ou da peça inventada

  comp2 : Coordenada y da trajetória do rei ou da peça inventada

    -Medidas espaciais calculadas a cada movimento do rei ou da peça
inventada.
  comp3 : Número de escaques repetidos (r)

comp4 : Área líquida (l), número escaques utilizados descontando os
repetidos.
comp5: Mínimo tabuleiro retangular que compreende a trajetória(tab).
Comp 6: Porcentagem de área líquida sobre o total de movimentos. A área
líquida são os escaques utilizados sem contar os repetidos
Comp7: Porcentagem de área líquida sobre o Mínimo tabuleiro retangular que
compreende a trajetoria.
Comp8: Porcentagem do mínimo tabuleiro retangular sobre o total de
movimentos (os valores podem ser maiores de 100)

Na versão default os componentes 9 a 16 contem idêntica informação, mas da
trajetória do cavalo.

Warning:
    A função devolve uma mensagem de erro se o usuário introduz um numero
inadequado de trajetórias(t): valor menor ou igual a cero, ou número com
decimais, ou superior ao máximo previsto (100). A função faz a mesma coisa
com o número de inadequado de movimentos (m), o número máximo foi fixado em
1000. Para a opção da peça inventada, o próprio r avisa se não encontra o
objeto que contem os movimentos primários (source).



Author(s):

     ~~Marcelo Fernández-Bolaños~~

References:

     ~put references to the literature/web site here ~

See Also:

     ~~objects to See Also as 'help', ~~~

Examples:
```

```
     ##scacorum.sim(t,m,source). Nesse exemplo calcularemos 10 trajetorias
do rei e 10 trajetorias do cavalo com 100 movimentos, na verdade, 99
movimentos a partir do escaque inicial (0,0))
sim.1=scacorum.sim(10,100)

#Para a opção "invent", precisamos criar uma matriz com os movimentos
primários de uma peça inventada, vamos a fazer um peão de exemplo
mov.prim.x.peao=c(1,0,-1)
mov.prim.y.peao=c(1,1,1)
movimentos.prim.peao=as.matrix(cbind(mov.prim.x.peao,mov.prim.y.peao))
sim.peao=scacorum.sim(10,80,movimentos.prim.peao )
```

```
scacorum.calc                 package:nenhum            R Documentation

~~ Calcula estatísticos descritivos sobre um objeto criado com a função
scacorum.sim , que contem simulações de trajetórias de peças de xadrez ~~

Description:

     ~~ Scacorum.calc permite calcular quatro estatísticos descritivos sobre
seis medidas de superfície de trajetórias aleatórias do rei, do cavalo, ou
de outra peça inventada de xadrez. O usuário pode escolher o número de
movimentos a analisar e como calcular o estatístico (intra ou entre
trajetórias).
Usage:

     ~~ scacorum.calc (dados,nm,surfmeas,estimator,intra.inter,peça)

Arguments:

 ~~
dados: Nome de objeto criado com a função com a função scacorum.sim que
contem um lista de matrizes com informação sobre trajetórias aleatória de
peças de xadrez.
-nm: Numero de movimentos sobre o total de movimentos simulados (m) do
objeto dados. Número inteiro do 1 ao m.
-surfmeas: Medida de espacial sobre a qual queremos estimar o estadístico.
Número inteiro do 1 ao 6: (1) escaques repetidos, 2) área líquida, 3)mínimo
tabuleiro retangular, 4) porcentagem de área líquida sobre o total de
movimentos, 5) porcentagem de área líquida sobre o Mínimo tabuleiro
retangular,  6) porcentagem do mínimo tabuleiro retangular sobre o total de
movimentos

-estimator: Estatístico exploratório que queremos calcular. Número inteiro
do 1 ao 6: 1)média, 2) desvio padrão , 3)valor mínimo, 4)valor máximo.
-intra.inter: Número inteiro: 1) a cada movimento o estatístico é calculado
entre trajetórias (para comparar entre movimentos), 2) O estático é
calculado sobre o total de todas as trajetórias (para comprar entre
```

```
trajetórias)
-peça: Número inteiro: 1)rei, 2)cavalo, 3) peça inventada.



~~

Value:

     ~

Scacorum.calc devolve uma lista com vários resultados
  comp1 : Título explicativo dos argumentos que foram escolhidos pelo
usuário (ex: "cavalo , média de escaques repetidos a cada movimento, com
100  de  100  movimentos, e com  10 trajetórias simuladas"



  comp2 : Sumario com o valor mínimo, o primeiro quantil, a mediana, a
média, o terceiro quantil e o valor máximo
comp3:Desvio padrão
comp4:Posição dos primeiros valores máximos. Quando no argumento intra-inter
escolhemos 1,nos indica com quantos movimentos obtemos os valores máximos do
estatístico, escolhendo 2 na opção intra-inter nos inca as trajetórias que
apresentaram os valores máximos do estatístico. A última indicação serve
para os seguintes dois componentes (5 e 6), estas saídas foram pensadas para
selecionar quais trajetórias visualizar com a função scacorum.graf.

comp5: Posição dos primeiros valores mínimos

comp6:Posição do/s valor/es da mediana

Também aparece um gráfico boxplot que permite visualizar como se distribuem
os valores da medida de superfície selecionada, com a amostra selecionada.
Warning:
    Se o usuário introduz um valor fora do intervalo de um argumento (os
intervalos foram indicados acima) a função devolve uma mensagem de erro.
Para o correto funcionamento de scacorum.calc o usuário deve entender a
posição de cada argumento, porem, sempre recomendamos observar o título para
evitar confusões.



Author(s):

     ~~Marcelo Fernández-Bolaños~~

Examples:

#Primeiro precisamos das simulaçoes criadas com scacorum.sim
sim.1=scacorum.sim(10,100)
##scacorum.calc(dados,nm,surfmeas,estimator,intra.inter,peça). Nesse exemplo
calculamos a média de escaques repetidos calculado a cada moviemento entre
as trajetórias do rei, incluindo todos os 100 movimentos simulados (em
```

```
sim.1)
scacorum.calc(sim.1,100,1,1,1,1)

sim.peao=scacorum.sim(10,80,movimentos.prim.peao )
##Nesse outro exemplo, calculamos o desvio padrão de tab/m calculado sobre o
total de trajetórias simuladas, com  50  de  100  movimentos simulados
(movimentos.prim.peao) da peça inventada (o peão)
scacorum.calc(sim.1,50,6,2,2,3)
```

```
scacorum.graf                package:nenhum            R Documentation

~~ Mostra a trajetória de uma peça de xadrez, previamente criada com a
função scacorum.sim ~~

Description:

     ~~ Apresenta uma imagem da trajetória de uma peça de xadrez criada com
a função scacorum.sim, pudendo escolher a quantidade de movimentos, e se
queremos ou não visualizar a trajetória que une os movimentos ~~

Usage:

     ~~ scacorum.graf (dados,nt,nm,traj, peça)
Arguments:

 ~~
dados: Nome de objeto criado com a função com a função scacorum.sim
-nt: Numero de trajetória que queremos visualizar, numero inteiro de 1 até o
máximo de trajetórias que contem o objeto dados.
-nm: numero de movimentos que queremos visualizar, numero inteiro de 1 até o
máximo de movimentos simulados no objeto  dados.
-traj: Indica 1)quero visualizar a trajetória que conecta ordinalmente os
escaques, 2) não quero visualizá-la.
-peça: 1)rei, 2)cavalo, 3) peça inventada.


~~

Details:

     ~~ É necessário ter o pacote "TeachingdDemos" para visualizar o gráfico
~~

Value:

     ~

Scacorum.graf  devolve um gráfico mostrando trajetória dados os argumentos
```

```
selecionados
Warning:
    Se o usuário introduz um valor fora do intervalo de um argumento (os
intervalos foram indicados acima) a função devolve uma mensagem de erro.


     ~ ~

Author(s):

     ~~Marcelo Fernández-Bolaños~~


Examples:
#Precisamos instalar o pacote TeachingdDemos
install.packages("TeachingdDemos")
#Depois precisamos  ter um objeto criado com scacorum.sim
sim.1=scacorum.sim(10,100)
# scacorum.graf=function(dados,nt,nm,traj,peça). Nesse exemplo veremos a
primeira trajetoria do rei, com todos os 100 movimentos simulados (no objeto
scacorum.sim), observando a trajetoria que une os movimentos
scacorum.graf(sim.1,1,100,1,1)
#Nesse outro exemplo vemos a quinta trajetória do cavalo, com 50 de 100
movimentos simulados (no objeto scacorum.sim), sem observar a trajetoria que
une os movimentos
scacorum.graf(sim.1,5,10,2,2)
```