Useful Functions in R: apply, lapply, and sapply

Maria van Schaijik

November 9, 2015
Introduction

Get to know any function in R
Get to know any function in R
Get to know any function in R

Simple Examples

Simple Examples

Simple Examples

Simple Simulation

When have I used them?
When have I used them?
How I’ve used them
How I’ve used them

There’s a lot more!
Aproach

For any new function the first thing I do is check the arguments that it takes:
Two easy ways to do this:
Aproach

For any new function the first thing I do is check the arguments that it takes:
Two easy ways to do this:

▶ help(new function)
▶ or just type the name of the function into your console.
Aproach

For any new function the first thing I do is check the arguments that it takes:
Two easy ways to do this:
- help(new function)
- or just type the name of the function into your console.
There are advantages to both
The apply function

Here’s the start of the apply function:

```r
> apply
function (X, MARGIN, FUN, ...)
{
  FUN <- match.fun(FUN)
  dl <- length(dim(X))
  if (!dl)
    stop("dim(X) must have a positive length")
  if (is.object(X))
    X <- if (dl == 2L)
      as.matrix(X)
    else as.array(X)
  d <- dim(X)
  ...
```
The apply function

Here's the start of the apply function:

```r
> apply
function (X, MARGIN, FUN, ...)
{
    FUN <- match.fun(FUN)
    dl <- length(dim(X))
    if (!dl)
        stop("dim(X) must have a positive length")
    if (is.object(X))
        X <- if (dl == 2L)
            as.matrix(X)
        else as.array(X)
    d <- dim(X)
    ...
```

some things I notice:
- if you're new to R this is a good way to learn how to code
- the arguments
- if statements
Useful Functions in R: apply, lapply, and sapply

Introduction

help(sapply)

lapply {base}

Apply a Function over a List or Vector

Description

lapply returns a list of the same length as x, each element of which is the result of applying fun to the corresponding element of x.

sapply is a user-friendly version and wrapper of lapply by default returning a vector, matrix or, if simplify = "array", an array if appropriate, by applying simplify2array(). sapply(x, f, simplify = FALSE, USE.NAMES = FALSE) is the same as lapply(x, f).

vapply is similar to sapply, but has a pre-specified type of return value, so it can be safer (and sometimes faster) to use.

replicate is a wrapper for the common use of sapply for repeated evaluation of an expression (which will usually involve random number generation).

simplify2array() is the utility called from sapply() when simplify is not false and is similarly called from mapply().

Usage

lapply(X, FUN, ...)

sapply(X, FUN, ..., simplify = TRUE, USE.NAMES = TRUE)

vapply(X, FUN, FUN.VALUE, ..., USE.NAMES = TRUE)

replicate(n, expr, simplify = "array")

simplify2array(x, higher = TRUE)

Arguments

x

a vector (atomic or list) or an expression object. Other objects (including classed objects) will be coerced by base::as.list.

FUN

the function to be applied to each element of x: see ‘Details’. In the case of functions like +, *, the function name must be backquoted or quoted.

...

optional arguments to fun.
What they have in common?

They’re all part of the apply family!
What they have in common?

They’re all part of the apply family!

In other words they all take the argument FUN
How do they differ?

- apply is the simplest case
How do they differ?

- apply is the simplest case
- Use sapply when you want a vector
How do they differ?

- apply is the simplest case
- Use sapply when you want a vector
- Use lapply when you want a list
How do they differ?

- apply is the simplest case
- Use sapply when you want a vector
- Use lapply when you want a list

Actually you can get identical results with sapply and lapply, especially in simple cases, but it’s a good idea to stick to that rule.
An Example

```r
> mymat = matrix(ceiling(runif(12, -2, 3)), 3, 4)
> mymat
[1,]   3   1   0   1
[2,]   1   3   2   0
[3,]  -1   2   3   0
> apply(mymat, 1, sum)
[1]  5  6  4
> apply(mymat, 2, sum)
[1]  3  6  5  1
```
Useful Functions in R: apply, lapply, and sapply

Simple Examples

```r
> sapply(1:4, FUN=function(x) sum(mymat[,x]))
[1]  3  6  5  1
> lapply(1:4, FUN=function(x) sum(mymat[,x]))
[[1]]
[1] 3

[[2]]
[1] 6

[[3]]
[1] 5

[[4]]
[1] 1
```
Useful Functions in R: apply, lapply, and sapply

Simple Examples

Apply with Higher Dim

```r
> load("custarray.Rdata")
> dim(custarray)
  [1] 10 365 96
>
> apply(custarray[,,], 1, mean, na.rm=T)
>
> apply(custarray[,,], 1, FUN=function(x) sum(x,na.rm=T)/4)
  [1] 2967520 2908825 3026295 2882453 2926160 2917057 2862053 2913955 2874793 2820947
>
> myindex=apply(custarray[,,],1, FUN=function(x){x=as.matrix(x)
            + which(x==max(x,na.rm=T),arr.ind=T)[1]})
>
> customer$DATE[myindex]
```
A silly example

I default to sapply often:
Useful Functions in R: apply, lapply, and sapply

A silly example

I default to sapply often:

```r
f = function(x, mu = 0, sigma = 1) { 63 / (sigma * 4) * (((x - mu) / sigma)^6 - ((x - mu) / sigma)^8) }

y1 = sapply(seq(-1, 1, .001), FUN = f)
y2 = sapply(seq(2, 4, .001), FUN = function(x) f(x, mu = 3, sigma = 1))
y3 = sapply(seq(1, 5, .001), FUN = function(x) f(x, mu = 3, sigma = 2))

plot(seq(-1, 1, .001), y1, xlim = c(0, 5), pch = 19, cex = 1, type = "l", xlab = "", ylab = "")
lines(cbind(seq(2, 4, .001), y2), col = "blue")
lines(cbind(seq(1, 5, .001), y3), col = "red")
legend("top", c("mu = 0 sigma = 1", "mu = 3 sigma = 1", "mu = 3 sigma = 2"),
  cex = .70, lty = c(1, 1, 1), lwd = c(2.5, 2.5, 2.5),
  col = c("black", "blue", "red"), bty = "n")
```
Useful Functions in R: apply, lapply, and sapply

Simple Examples
library(smoothmest)
library(quantreg)
gencordata=function(n,rho=.8, beta=c(1,1,1,rep(0,5)), noise=1, dist=1){
    m=length(beta)
    if(m==1){v=1}else{
        v=c(1,rep(rho,seq(1:(m-1))))
        v=toeplitz(v)
    }
    x=mvrnorm(n,rep(0,m),v)
    stu=.5+x%*%beta+rt(n,3)
    norm=.5+x%*%beta+rnorm(n,0,noise)
    laplace=.5+x%*%beta+rdoublex(n,0,noise)
    if(dist==1) return(cbind(norm,x))
    if(dist==2) return(cbind(stu,x))
    if(dist==3) return(cbind(laplace,x))
}

mysims=array(NA, dim=c(1000, 3, 2, 9))

for(i in 1:1000) {

    sim=as.array(lapply(c(1,2,3), function(dist){

        data=data.frame(gencordata(200,dist=dist))
        rbind(coef(rq(X1~., data=data)),
              coef(lm(X1~., data=data)))
    })))

    mysims[i,1,,]=sim[[1]]
    mysims[i,2,,]=sim[[2]]
    mysims[i,3,,]=sim[[3]]
}

layout(matrix(1:6,3,2))
for(i in 1:3) boxplot(mysims[,i,2,,], col="blue")
for(i in 1:3) boxplot(mysims[,i,1,,], col="red")
Useful Functions in R: apply, lapply, and sapply

Simple Simulation
SRP biller

SRP is utility company here in the valley.
A customer’s bill is determined by (among other things):

▶ The rate a customer is on.
SRP biller

SRP is utility company here in the valley. A customer’s bill is determined by (among other things):

- The rate a customer is on.
- For many rates consumption is charged differently based on time of use.
SRP biller

SRP is utility company here in the valley. A customer’s bill is determined by (among other things):

- The rate a customer is on.
- For many rates consumption is charged differently based on time of use.
- There is also frequently a demand charge, which varies based on TOU.
In Load Research, we have access to 15 minute demand data from each customer. Different rates are meant to incentivize customers to shift their consumption to “off-peak” hours. As an intern, my first task was to create a biller that would compare bills for different rates.
First I had to create a few pretty ugly functions. But once, they were created I could use the lapply and sapply functions to ‘apply’ each function:
First I had to create a few pretty ugly functions. But once, they were created I could use the `lapply` and `sapply` functions to ‘apply’ each function:

```r
> largeplans=c(61,63,65)
> kwh.by.rate=lapply(largeplans, FUN=function(rate){get.kwh.tou(rate,customer,month)})
> names(kwh.by.rate)=paste("E",largeplans, sep="")
```
First I had to create a few pretty ugly functions. But once, they were created I could use the `lapply` and `sapply` functions to ‘apply’ each function:

```r
> largeplans=c(61,63,65)
> kwh.by.rate=lapply(largeplans, FUN=function(rate){get.kwh.tou(rate,customer,month)})
> names(kwh.by.rate)=paste("E",largeplans, sep="")

> kwh.by.rate
$E61
$kwh.on.peak
 January February March April May June July August September
15215.62 14067.17 16015.39 15883.44 39731.81 43694.06 49698.05 41860.32
$kwh.shldr.peak
 January February March April May June July August September
18819.89 18669.74 23566.85 25250.69 50981.38 56799.84 66349.06 54919.34
$kwh.off.peak
 January February March April May June July August September
79247.28 77343.94 105876.29 108381.46 114132.43 124217.38 140277.70 135060.58 115802.45
```
Useful Functions in R: `apply`, `lapply`, and `sapply`

---

How I've used them

```r
> df=data.frame(kwh.by.rate)
> rownames(df)=month
> df
```

<table>
<thead>
<tr>
<th>Month</th>
<th>E61.kwh.on.peak</th>
<th>E61.kwh.shldr.peak</th>
<th>E61.kwh.off.peak</th>
<th>E63.kwh.on.peak</th>
<th>E63.kwh.shldr.peak</th>
<th>E63.kwh.off.peak</th>
</tr>
</thead>
<tbody>
<tr>
<td>January</td>
<td>15215.62</td>
<td>18819.89</td>
<td>79247.28</td>
<td>15215.62</td>
<td>18819.89</td>
<td>79247.28</td>
</tr>
<tr>
<td>February</td>
<td>14067.17</td>
<td>18669.74</td>
<td>77343.94</td>
<td>14067.17</td>
<td>18669.74</td>
<td>77343.94</td>
</tr>
<tr>
<td>March</td>
<td>16015.39</td>
<td>23566.85</td>
<td>105876.29</td>
<td>16015.39</td>
<td>23566.85</td>
<td>105876.29</td>
</tr>
<tr>
<td>April</td>
<td>15883.44</td>
<td>25250.69</td>
<td>108381.46</td>
<td>15883.44</td>
<td>25250.69</td>
<td>108381.46</td>
</tr>
<tr>
<td>May</td>
<td>39731.81</td>
<td>50981.38</td>
<td>114132.43</td>
<td>39731.81</td>
<td>50981.38</td>
<td>114132.43</td>
</tr>
<tr>
<td>June</td>
<td>43694.06</td>
<td>56799.84</td>
<td>124217.38</td>
<td>43694.06</td>
<td>56799.84</td>
<td>124217.38</td>
</tr>
<tr>
<td>July</td>
<td>49698.05</td>
<td>66349.06</td>
<td>140277.70</td>
<td>49698.05</td>
<td>66349.06</td>
<td>140277.70</td>
</tr>
<tr>
<td>August</td>
<td>41860.32</td>
<td>54919.34</td>
<td>135060.58</td>
<td>41860.32</td>
<td>54919.34</td>
<td>135060.58</td>
</tr>
<tr>
<td>September</td>
<td>42725.90</td>
<td>55195.06</td>
<td>115802.45</td>
<td>42725.90</td>
<td>55195.06</td>
<td>115802.45</td>
</tr>
<tr>
<td>October</td>
<td>36196.61</td>
<td>43567.20</td>
<td>92553.22</td>
<td>36196.61</td>
<td>43567.20</td>
<td>92553.22</td>
</tr>
<tr>
<td>November</td>
<td>14416.22</td>
<td>20033.90</td>
<td>92423.18</td>
<td>14416.22</td>
<td>20033.90</td>
<td>92423.18</td>
</tr>
<tr>
<td>December</td>
<td>14775.84</td>
<td>18713.09</td>
<td>74948.30</td>
<td>14775.84</td>
<td>18713.09</td>
<td>74948.30</td>
</tr>
</tbody>
</table>
```
> kwh.tou=lapply(32, FUN=function(rate){get.kwh.tou(rate,customer,month)})
> dfgen=data.frame(kwh.tou)
> rownames(dfgen)=month
> kwh=sapply(month,FUN=function(x){kwh=get.tot.consumption(customer,x)/4)
> dfgen$MaxDemand30=sapply(month,FUN=function(x){get.dcharge.gen(32,customer,x)[2]})
> dfgen$Billkwh=sapply(month,FUN=function(x){unlist(dfgen[[which(rownames(dfgen)==x),2:4]])%*%prices$E32(x)
> dfgen$Billdemand30=sapply(month,FUN=function(x){get.dcharge.gen(32,customer,x)[1]})
> dfgen$Onpeakflag=sapply(month,FUN=function(x){get.dcharge.gen(32,customer,x)[3]})
> dfgen$MaxDemand15=sapply(month,FUN=function(x){get.dcharge.gen(36,customer,x)[2]})
> dfgen$Billdemand15=sapply(month,FUN=function(x){get.dcharge.gen(36,customer,x)[1]})
> dfgen$Monthlycharge=22.08
> dfgen
Useful Functions in R: apply, lapply, and sapply

How I've used them

<table>
<thead>
<tr>
<th></th>
<th>kwh.on.peak</th>
<th>kwh.shldr.peak</th>
<th>kwh.off.peak</th>
<th>MaxDemand30</th>
<th>Billkwh</th>
<th>Billdemand30</th>
<th>Onpeakflag</th>
<th>MaxDemand15</th>
<th>Billdemand15</th>
</tr>
</thead>
<tbody>
<tr>
<td>January</td>
<td>15215.62</td>
<td>18819.89</td>
<td>79247.28</td>
<td>244.224</td>
<td>10487.00</td>
<td>239.224</td>
<td>0</td>
<td>245.952</td>
<td>1077.055</td>
</tr>
<tr>
<td>February</td>
<td>14067.17</td>
<td>18669.74</td>
<td>77343.94</td>
<td>284.544</td>
<td>10272.59</td>
<td>279.544</td>
<td>0</td>
<td>291.456</td>
<td>1280.458</td>
</tr>
<tr>
<td>March</td>
<td>16015.39</td>
<td>23566.85</td>
<td>105876.29</td>
<td>345.024</td>
<td>13828.78</td>
<td>340.024</td>
<td>0</td>
<td>354.240</td>
<td>1561.103</td>
</tr>
<tr>
<td>April</td>
<td>15883.44</td>
<td>25250.69</td>
<td>108381.46</td>
<td>360.384</td>
<td>14287.12</td>
<td>355.384</td>
<td>0</td>
<td>366.336</td>
<td>1615.172</td>
</tr>
<tr>
<td>May</td>
<td>39731.81</td>
<td>50981.38</td>
<td>114132.43</td>
<td>484.896</td>
<td>19866.71</td>
<td>479.896</td>
<td>0</td>
<td>492.672</td>
<td>2155.510</td>
</tr>
<tr>
<td>June</td>
<td>43694.06</td>
<td>56799.84</td>
<td>124217.38</td>
<td>483.168</td>
<td>21822.16</td>
<td>478.168</td>
<td>0</td>
<td>484.416</td>
<td>2119.019</td>
</tr>
<tr>
<td>July</td>
<td>49698.05</td>
<td>66349.06</td>
<td>140277.70</td>
<td>516.328</td>
<td>26861.29</td>
<td>510.328</td>
<td>0</td>
<td>518.400</td>
<td>3470.584</td>
</tr>
<tr>
<td>August</td>
<td>41860.32</td>
<td>54919.34</td>
<td>135060.58</td>
<td>471.552</td>
<td>24345.42</td>
<td>466.552</td>
<td>0</td>
<td>472.704</td>
<td>3161.679</td>
</tr>
<tr>
<td>September</td>
<td>42725.90</td>
<td>55195.06</td>
<td>115802.45</td>
<td>449.952</td>
<td>20689.47</td>
<td>444.952</td>
<td>0</td>
<td>453.504</td>
<td>1982.388</td>
</tr>
<tr>
<td>October</td>
<td>36196.61</td>
<td>43667.20</td>
<td>92653.22</td>
<td>385.440</td>
<td>16452.92</td>
<td>380.440</td>
<td>0</td>
<td>394.944</td>
<td>1723.552</td>
</tr>
<tr>
<td>November</td>
<td>14416.22</td>
<td>20033.90</td>
<td>92423.18</td>
<td>319.584</td>
<td>12010.08</td>
<td>1277.211</td>
<td>1</td>
<td>325.824</td>
<td>1434.083</td>
</tr>
<tr>
<td>December</td>
<td>14775.84</td>
<td>18713.09</td>
<td>74948.30</td>
<td>280.512</td>
<td>10026.82</td>
<td>275.512</td>
<td>0</td>
<td>282.048</td>
<td>1238.405</td>
</tr>
</tbody>
</table>
Parallel Computing

The functions we’ve discussed today don’t necessarily improve your run time. However:

```r
#install.packages("simsalapar")
#install.packages("doParallel")
library(simsalapar)
library(MASS)
varList=varlist(n.sim=list(expr=quote(N[sim]),value=3),
t=list(type="grid", value=c(100,200,400)),
bpt=list(type="grid",value=c(0.25,0.75)),
beta1=list(type="frozen",value=c(.5,1,1.5,2,rep(0,5))),
beta2=list(type="frozen",value=c(.5,-1,-1,1,rep(0,5))),
method=list(type="inner",value=c("OLS","MEDIAN","LASSO","ENET","LAD1","LAD2","LAD3")),
distribution=list(type="grid", value=c("normal","student","leplace")),
rho=list(type="frozen", value=.8),
noise=list(type="frozen",value=.5))
doOne1=function(t,bpt,rho,noise,beta1,beta2,distribution,method){
gencordata=function(t,bpt,rho,noise, beta1,beta2, noise2=0, distrib=c("normal","student","leplace")){...}
stepDoall=function(x,y,q,trim=.15,method=c("OLS","MEDIAN","LASSO","ENET","LAD1","LAD2","LAD3")){...}
data=gencordata(t=t,bpt=bpt,rho=rho,noise=noise,beta1=beta1,beta2=beta2,distrib=distribution)
stepDoall(data$x,data$y,data$q,method=method)
}
simu1=doForeach(varList,doOne=doOne1,check=F,extraPkgs=c("mvtnorm","stats","glmnet","lsr","smoothmest","flare","quantreg"))
```

Parallel Computing

The functions we’ve discussed today don’t necessarily improve your run time. However:

```r
#install.packages("simsalapar")
#install.packages("doParallel")
library(simsalapar)
library(MASS)

varList=varlist(n.sim=list(expr=quote(N[sim]),value=3),
    t=list(type="grid", value=c(100,200,400)),
    bpt=list(type="grid", value=c(0.25,0.75)),
    beta1=list(type="frozen",value=c(.5,1,1.5,2,rep(0,5))),
    beta2=list(type="frozen",value=c(.5,-1,-1,1,rep(0,5))),
    method=list(type="inner",value=c("OLS","MEDIAN","LASSO","ENET","LAD1","LAD2","LAD3")),
    distribution=list(type="grid", value=c("normal","student","leplace")),
    rho=list(type="frozen", value=.8),
    noise=list(type="frozen",value=.5)
)

doOne1=function(t,bpt,rho,noise,beta1,beta2,distribution,method){
    gencordata=function(t,bpt,rho,noise, beta1,beta2, distrib=c("normal","student","leplace")){...}
    stepDoall=function(x,y,q,trim=.15,method=c("OLS","MEDIAN","LASSO","ENET","LAD1","LAD2","LAD3")){...}
    data=gencordata(t=t,bpt=bpt,rho=rho,noise=noise,beta1=beta1,beta2=beta2,distrib=distribution)
    stepDoall(data$x,data$y,data$q,method=method)
}

system.time(
  { sim1=doForeach(varList,doOne=doOne1,check=F,extraPkgs=c("mvtnorm","stats","glmnet","lsr","smoothmest","flare","quantreg")) })
```
res=getArray(sim1)

layout=layout(matrix(1:6, 3,2))
for(k in 1:3){
  for(i in 1:3){
    for(j in 1:2){

      boxplot(res[,i,j,k,], names=c("ols","lasso","alasso","alad1","alad2", "alad3"),las=2)
      bpt=c(.25,.75)
      abline(h=bpt[j], col="red")
    }
  }
}

res=getArray(sim1)

layout=layout(matrix(1:6, 3,2))
for(k in 1:3){
    for(i in 1:3){
        for(j in 1:2){

            boxplot(res[,i,j,k], names=c("ols","lasso","alasso","alad1","alad2", "alad3"),las=2)
            bpt=c(.25,.75)
            abline(h=bpt[j], col="red")
        }
    }
}

Thankyou!