MUSÉUM NATIONAL D'HISTOIRE NATURELLE

SEMIN-R

Récurrence en R

Anton CAMACHO

ENS Paris, UMR 7625
"Fonctionnement et Evolution des Systèmes Ecologiques"

camacho@biologie.ens.fr

INTE  ÊT

- Exécuter plusieurs fois une tâche identique.
- Nécessite l'écriture de petits programmes.
-  rend la programmation accessible à des non-spécialistes.

Pour cela, 2 types d'outils:

- Les boucles ou structures de contrôle (type langage C). ex: for, while ,...

- Les fonctions déjà implémentées dans  . ex: apply(), sapply(),lapply(),...

Les boucles for.

- Syntaxe:

for(var in X)

{

instructions

}

- var est la variable de contrôle.
- X est un vecteur dont chaque élément correspond à une itération des {instructions}.

Exemple, for.

```
> for(i in 1:5) print(1:i)
[1] 1
[1] 1 2
[1] 1 2 3
[1] 1 2 3 4
[1] 1 2 3 4 5
```

```
> for(i in c(2,5,10,20,50))
{
    x <- rnorm(i)
    cat(i,":", sum(x^2),"\n")
}
2 : 0.1007362
5 : 6.928372
10 : 11.55081
20 : 16.46345
50 : 37.68763
```

Les boucles while.

- Syntaxe:

while(condition)

{

instructions

}

- condition est une expression logique.
- Tant que la condition est vraie, les {instructions} sont exécutées successivement.

Exemple, while.

```
> minimum<-5
> x<-10
> while(x>minimum)
{
print(x)
x<-x-1
}
[1] 10
[1] 9
[1] 8
[1] 7
[1] 6
```

- Mais attention aux boucles infinies:
avec `x<-x+1` la condition `x>5` reste toujours vraie!

Un exemple concret.

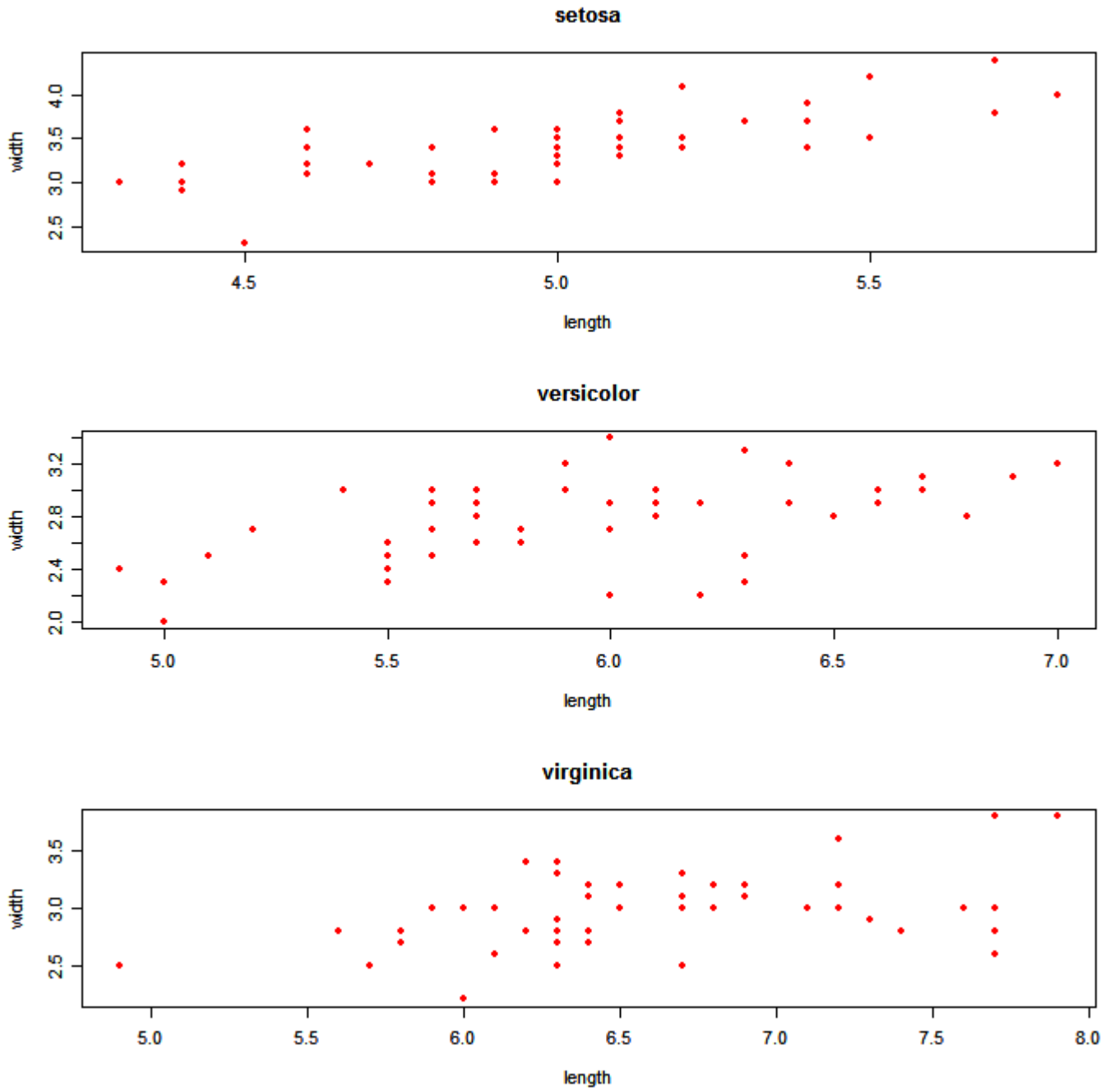
- On veut tracer le même graphique pour 3 espèces d'iris différentes, les données se trouvant dans 3 fichiers distincts: *setosa.dat*, *versicolor.dat* et *virginica.dat*

- Plutôt que de répéter 3 fois les mêmes instructions on va utiliser la boucle for.

```
> layout(matrix(1:3,3,1)) #partitionne le graph
> species<-c("setosa","versicolor","virginica")
> for(i in species)
{
data<-read.table(paste(i,".dat",sep=""),h=T)
plot(data[,1],data[,2],type="p",pch=19,
col="red",xlab="length",ylab="width")
title(i) #ajoute le titre
}
```

- `Paste()` est une fonction qui concaténe des chaînes de caratères en les séparant selon l'argument sep choisi:

```
> paste("setosa",".dat",sep="")
[1] "setosa.dat"
```


setosa
width
length
versicolor
width
length
virginica
width
length

Les fonctions récurrentes.

- Elles évitent d'écrire des boucles.
- Elles dépendent des objets que l'on manipule

Objet en entrée	Objet en sortie	Fonction
Matrix, array	Vector, list, array	Apply()
List	List	Lapply()
List	Matrix, vector	Sapply()

- Ils en existent d'autres: tapply(), mapply()...

Apply()

- Syntaxe:
apply(X, MARGIN, FUN, ...)
- X est un tableau ou une matrice.
- MARGIN indique si l'action doit être appliquée sur les lignes (1), sur les colonnes (2) ou les deux (c(1,2)).
- FUN est la fonction qui sera utilisée.
- ... sont d'éventuels arguments supplémentaires pour FUN.

Exemple, apply().

•Matrice de nombres aléatoires de distribution gaussienne:

```
> M<-matrix(rnorm(20,5,2),nrow=5)
> M
          [,1]      [,2]     [,3]      [,4]
[1,] 7.747164 2.794391 6.026218 5.6616948
[2,] 4.948285 5.653785 5.890097 5.3554275
[3,] 6.267819 5.405549 1.568598 7.7097979
[4,] 6.922841 6.359697 3.213906 4.8524603
[5,] 5.280376 5.057567 8.542031 0.8350141
```

•On cherche la moyenne par colonne:

```
> apply(M,2,mean)
[1] 6.233297 5.054198 5.048170 4.882879
```

Sapply() et Lapply().

- Fonctions similaires mais sapply() retourne un vecteur, plus facile à afficher en pratique.
- Syntaxe:
sapply(X, FUN, ...) et lapply(X, FUN, ...)
- Plus d'argument MARGIN car on a affaire à des listes.

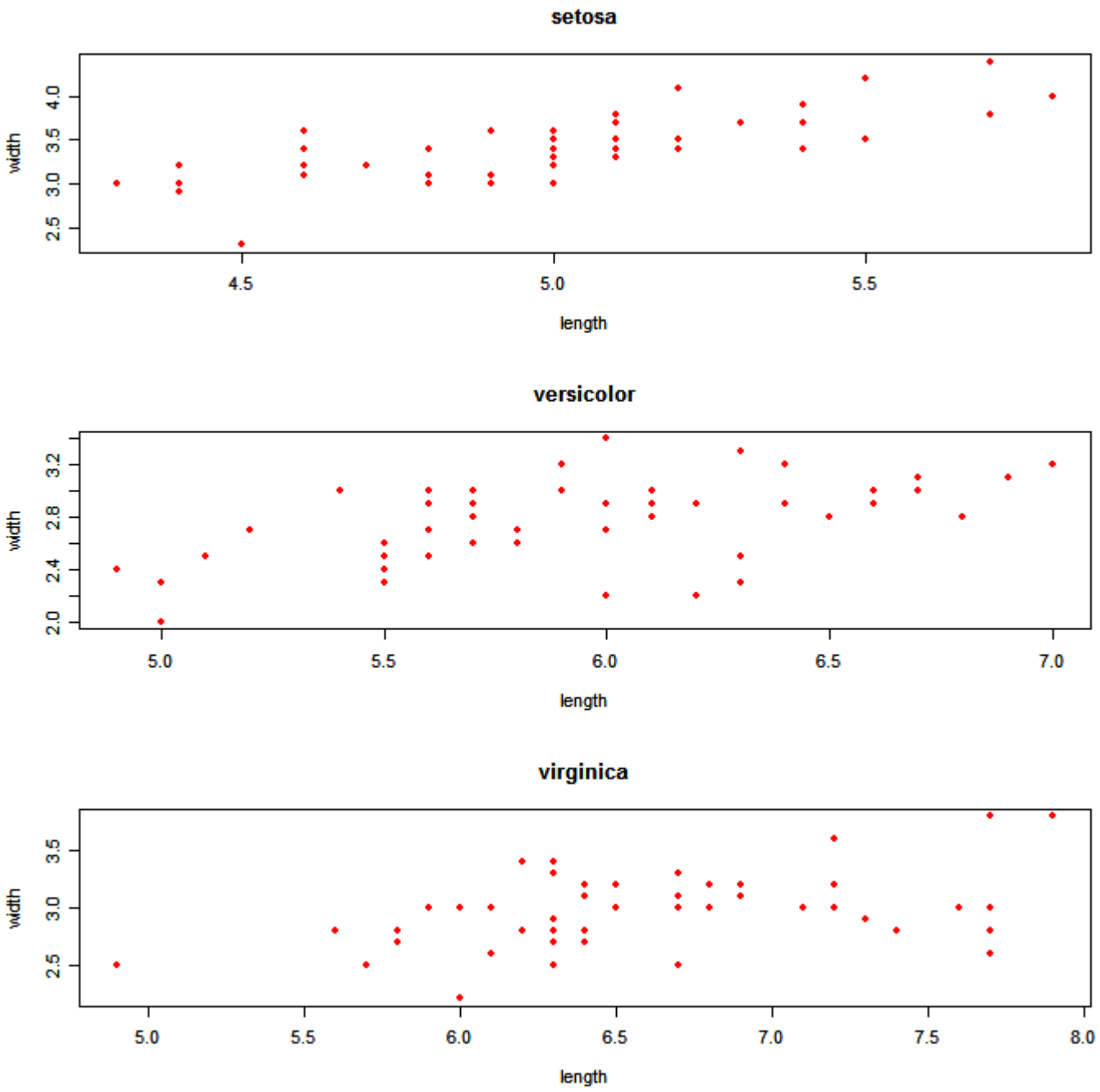
Exemple concret.

- Autre façon de tracer nos 3 graphiques: on définit la fonction à appliquer à chaque espèce:

```
> maFunc<-function(x,h,...)
{
data<-read.table(paste(x,".dat",sep=""),h)
plot(data[,1],data[,2],...)     #prend les arguments
                                #supplementaires (...)
                                #passés dans maFunc().
title(x)
}
```

- On utilise sapply() sur le vecteur species:

```
>sapply(species,maFunc,h=T,type="p",pch=19,col="red",
xlab="length",ylab="width")
```


setosa
width
length
versicolor
width
length
virginica
width
length

Un exemple avec tapply()

- Utile quand on s'intéresse aux différents niveaux d'un facteur: dans le jeu de données « iris », on a, pour chaque espèce (le facteur), plusieurs mesures de la longueur des sépales. On aimerait connaître la longueur moyenne des sépales pour chacune des 3 espèces.
- Le jeu de données:

```
> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
...
```

- Syntaxe: tapply(X, INDEX, FUN, ...)
- INDEX est le vecteur de même taille que X, indiquant le niveau du facteur.

```
> tapply(iris$Sepal.Length,iris$Species,mean)
    setosa versicolor  virginica
     5.006      5.936      6.588
```

Boucles ou Fonctions récurrentes?

- Dernière version de  : temps d'exécution similaires.

- Les fonctions récurrentes sont souvent plus commodes à utiliser.

- Attention: si trop de boucles imbriquées, le temps d'exécution peut devenir très long.