Université de Kairouan **Année Universitaire 2012/2013**
ISMAI **M2 : IF**
Enseignant : Med Essaied Hamrita

Série temporelle-Simulation
TP0 : Introduction au logiciel R

L'objectif de ce TP est de présenter les différents types d'objets (vecteurs, matrices, listes,...) et quelques fonctions utiles.

1 Objets

1.1 Objets simples :

Comme tout langage de programmation, R permet de faire des opération arithemétiques tout en affectant le résultat dans un objet.

```
> x=2 # mettre le valeur 2 dans l'objet x.
```

On peut utiliser cet objet dans d'autres calculs :

```
> x+3
```

```
[1] 5
```

On peut utiliser autant d'objets qu'on veut. Ceux-ci peuvent contenir des nombres, des chaînes de caractères et bien d'autres choses :

```
> x=27;x
```

```
[1] 27
```

```
> y=10;y
```

```
[1] 10
```

```
> s=x+y;s
```

```
[1] 37
```

```
> h="Bonjour";h
```

```
[1] "Bonjour"
```

1.2 Vecteurs :

R possède des fonctions prédéfinies pour créer des vecteurs et des séquences simples :

1.2.1 La fonction c()"

Pour créer un vecteur simple, on peut utiliser la fonction c().

```
>   c(1,1,3,5) # création d'un vecteur contenant 1 1 3 5 respectivement.
```

```
[1] 1 1 3 5
```

1.2.2 La fonction " :"

peut être utiliser pour créer des séquences entières de bases.

```
>   2:5 # création d'une séquence de 2 à 5 d'un pas égale à  1.

[1] 2 3 4 5

>   6:3 # création d'une séquence dans l'ordre décroissant.

[1] 6 5 4 3
```

1.2.3 La fonction seq()

La fonction `seq` génère des séquences régulières qui ne sont pas nécessairement des entiers. La syntaxe de base est `seq(from,to,by)` où `by` est l'incrément.

```
> seq(1,2,0.1) # génère une séquence de 1 à 2 d'un pas égale à  0.1.

 [1] 1.0 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2.0
```

Mais, on peut utiliser `length` ou `len` plutôt que `by`

```
> seq(1,2,len=4) # génère une séquence de 1 à 2 de longueur égale à 4.

[1] 1.000000 1.333333 1.666667 2.000000
```

1.2.4 La fonction rep()

La fonction `rep` génère des vecteurs avec répitition d'un échantillon donné.

```
> rep(1,5)           # renvoi un vecteur dans lequel on repète 1 cinq fois.

[1] 1 1 1 1 1

> rep(c(1,2),3)      # on repète le vecteur 1 2 trois fois.

[1] 1 2 1 2 1 2

> rep(1:3,times=1:3) # on repète chaque éléments du vecteur times fois.

[1] 1 2 2 3 3 3
```

```
> rep(0:2, each=2)   # on repète chaque élément du vecteur each fois

[1] 0 0 1 1 2 2
```

Extraction et remplacement d'un élément d'un vecteur : Si `x` est un vecteur, `x[i]` est le ieme élément de `x`.

```
> x=letters[1:8]; x # renvoi les huit premiers lettres.

[1] "a" "b" "c" "d" "e" "f" "g" "h"

> x[4]              # extraction du 4ieme lettre.

[1] "d"

> x[4:5]            # extraction des 4ieme et 5ieme lettres.

[1] "d" "e"

> i=c(1,3,7)
> x[i]              # extraction des 1er, 3ieme et 7ieme lettre.

[1] "a" "c" "g"

> x=seq(0,10,2); x  # création d'une séquence de 0 à 10 avec un pas de 2.

[1]  0  2  4  6  8 10

> x[3]=NA ; x       # affectation d'une valeur manquante.

[1]  0  2 NA  6  8 10

> x[-4]             # enlever le 4ieme élément.

[1]  0  2 NA  8 10
```

Exercice 1

1) Générer les vecteurs suivants de deux manières différentes.
1 2 3 4 ; 1 2 3 4 8 9 10 11 ; 2 4 6 8 et 1.0 1.2 1.4 1.6 1.8 2.0

2) Générer les vecteurs suivants :
1 4 1 4 1 4 ; 1 1 1 4 4 4 et 1 1 4 4 4 4 4.

3) Avec R, calculer (ou approximer) les sommes suivantes :

$$\sum_{i=1}^{10} i, \quad \sum_{i=1}^{100} i^2, \quad \sum_{i=1}^{+\infty} \frac{1}{2^i}$$

1.3 Matrices

Les matrices sont crées par la fonction matrix() à partir d'un vecteur. On doit fixer le nombre de colonnes ncol et/ou le nombre de lignes nrow.

```
> X=matrix(c(2,3,5,7), ncol=2); X # création d'une matrice
```

```
     [,1] [,2]
[1,]    2    5
[2,]    3    7
```

Par défaut, la matrice est remplie colonne par colonne. Pour la remplir ligne par ligne, on doit ajouter l'argument byrow=T à la fonction matrix.

```
> Y=matrix(c(2,3,5,7),ncol=2,byrow=T); Y
```

```
     [,1] [,2]
[1,]    2    3
[2,]    5    7
```

```
> nrow(Y) ;ncol(Y)  # renvoie le nombre de lignes et le nombre de colonnes respectivement.
```

```
[1] 2
```

```
[1] 2
```

```
> dim(Y)          # renvoie la dimension d'un matrice.
```

```
[1] 2 2
```

```
> Y[2,]           # extraire la deuxième ligne.
```

```
[1] 5 7
```

```
> Y[-1,]          # enlever la première ligne.
```

```
[1] 5 7
```

```
> Y[2,1]          # extraire l'élément y21 de la matrice.
```

```
[1] 5
```

Il y a aussi une autre façon pour créer un matrice en utilisant les fonctions cbind() et rbind(). cbind() permet de concaténer deux colonnes ou deux matrices ayant le même nombre de lignes et rbind() permet de concaténer deux lignes ou deux matrices ayant le même nombre de colonnes.

```
> x1=1:3; x2=c(-1,0,5)
> A=rbind(x1,x2); A      # concaténation de deux lignes.
```

```
   [,1] [,2] [,3]
x1    1    2    3
x2   -1    0    5
```

```
> B=cbind(x1,x2); B      # concaténation de deux colonnes.
```

```
     x1 x2
[1,]  1 -1
[2,]  2  0
[3,]  3  5
```

```
> C=rbind(X,Y); C        # concaténation de deux matrices.
```

```
     [,1] [,2]
[1,]    2    5
[2,]    3    7
[3,]    2    3
[4,]    5    7
```

Opérations mathématiques sur les matrices : $+$, $/$, $*$, sont les opérateurs usuels terme à terme. Le produit matriciel se fait à l'aide de la fonction $\%*\%$.

```
> X+Y           # somme terme à terme de deux matrices.
```

```
     [,1] [,2]
[1,]    4    8
[2,]    8   14
```

```
> X*Y           # produit terme à terme de deux matrices.

     [,1] [,2]
[1,]    4   15
[2,]   15   49

> X^2           # renvoie le carré des éléments de la matrice.

     [,1] [,2]
[1,]    4   25
[2,]    9   49

> X%*%Y; det(X) # le produit matriciel de deux matrices et le déterminent de X.

     [,1] [,2]
[1,]   29   41
[2,]   41   58

[1] -1

> solve(X)      # calcul de l'inverse d'une matrice carré.

     [,1] [,2]
[1,]   -7    5
[2,]    3   -2

> b=c(2,-1); solve(X,b)  # résout l'équation X*x=b.

[1] -19   8

> eigen(X)      # donne les valeurs propres et les vecteurs propres de X.

$values
[1]  9.1097722 -0.1097722

$vectors
           [,1]       [,2]
[1,] -0.5752492 -0.9213379
[2,] -0.8179782  0.3887626
```

Exercice 2

On veut résoudre le système d'équations suivant :

$$(S) : \begin{cases} x + z = 2 \\ x + 2y - z = 2 \\ x - y + z = 3 \end{cases}$$

Créer une matrice A et un vecteur b telles que le système S s'écrit $AX = b$, puis résoudre ce système.

1.4 Les tableaux data.frame

Les tableaux de données ressemblent beaucoup aux matrices par leur structure. Ils sont en effet composés de lignes et de colonnes et sont donc eux aussi des objets à deux dimensions. Les tableaux de données différent des matrices en un principal point : **les éléments d'une ligne ou d'une colonne peuvent être de types différents**.

```
> # Création d'un data.frame
> H <- data.frame(age = c(15,20), nom = c("Mohamed", "Sarah"),
+                 sexe=c("M","F"), row.names = c("I1", "I2"))
> H
```

```
   age     nom sexe
I1  15 Mohamed    M
I2  20   Sarah    F
```

```
> H$nom               # donne un vecteur contenant les noms.
```

```
[1] Mohamed Sarah
Levels: Mohamed Sarah
```

```
> H$age               # donne un vecteur contenant les ages.
```

```
[1] 15 20
```

1.5 Listes

Une liste est un tableau ordonné d'éléments qui peuvent être hétérogènes.

```
> l1=list(X=X,Nom="Matrice", n=2)  # création d'une liste.
> names(l1)                        # les noms des objets dans l1.
```

```
[1] "X"   "Nom" "n"
```

```
> l1["X"]                          # extraire l'objet X de la liste l1.
```

```
$X
     [,1] [,2]
[1,]    2    5
[2,]    3    7


> l1$X; l1[[1]]                    # autres façons pour extraire l'objet X.


     [,1] [,2]
[1,]    2    5
[2,]    3    7


     [,1] [,2]
[1,]    2    5
[2,]    3    7


> attach(l1)                       #


The following object(s) are masked _by_ '.GlobalEnv':

    X
```

2 Quelques fonctions

Dans cette section, nous présentons quelques fonctions utiles.

2.1 Opérateurs mathématiques

Il existe trois types d'opérateurs : opérateurs arithmétiques, opérateurs de comparaison et opérateurs logiques. En voici la liste :

Opérateurs					
Arithmétique		Comparaison		Logique	
`+`	Addition	$<$	Inférieur à	`!x`	NON logique
`-`	Soustraction	$>$	Supérieur à	`x&y`	ET logique
$*$	multiplication	$<=$	Inférieur ou égale à	`x&&y`	idem
^	Puissance	$>=$	Supérieur ou égale à	`x|y`	OU logique
`/`	Division	`==`	Égale à	`x||y`	idem
`%%`	Modulo	`!=`	Différent à	`xor(x,y)`	OU exclusif
`%/%`	Division entière				

2.2 Fonctions utiles

Ici, nous donnons une liste des fonctions utiles pour manupiler une base de données.

`sum(x)`	somme des éléments de `x`.
`prod(x)`	produit des éléments de `x`.
`max(x)`	maximum des éléments de `x`.
`min(x)`	minimum des éléments de `x`.
`which.max(x)`	retourne l'indice du maximum des éléments de `x`.
`which.min`	retourne l'indice du minimum des éléments de `x`.
`range(x)`	donne le vecteur `c(min(x),max(x))`.
`length(x)`	donne le nombre d'éléments dans `x`.
`mean(x)`, `median(x)`	moyenne et médiane de `x` respectivement.
`var(x)` ou `cov(x)`	variance de `x` (calculée sur $n-1$) ; si `x` est une matrice ou un tableau de données, la matrice de variance-covariance est calculée.
`cor(x)`	matrice de corrélation si `x` est une matrice ou un tableau de données.
`cumsum(x)`	la somme cumulé de `x`.
`cumprod(x)`	le produit cumulé de `x`.
`fft(x)`	transformée de fourier de `x`.

2.3 Fonctions `apply`

La fonction `apply()` permet d'appliquer une fonction (par exemple une moyenne, une somme) à chaque ligne ou chaque colonne d'un tableau de données. Cette fonction prend 3 arguments dans l'ordre suivant : nom du tableau de données, un nombre pour dire si la fonction doit s'appliquer aux lignes (1), aux colonnes (2) ou aux deux (c(1,2)) et le nom de la fonction à appliquer.

```
> data<-matrix(c(1,2,3,4,5,6), nrow=2)  # On crée une matrice 2x3.
> apply(data, 1, sum) # On utilise la fonction apply() pour faire la somme de chaque ligne
```

```
[1]  9 12
```

Remarque : Pour appliquer la somme et la moyenne sur les colonnes (les lignes) d'une matrice, on peut utiliser aussi les fonctions `colSums` et `colMeans` (`rowSums` et `rowMeans`) respectivement.

```
> colSums(data)
```

```
[1]  3  7 11
```

```
> colMeans(data)
```

```
[1] 1.5 3.5 5.5
```

```
> rowMeans(data)
```

```
[1] 3 4
```

Les fonctions `sapply` et `lapply` appliquent la même fonction sur tous les élément d'un vecteur ou une liste. Le résulta de la fonction `sapply` est retourné sous forme de vecteur, si possible.

```
> ll1=lapply(1:4, runif); ll1  # donne une liste contenant des simulations des réalisations uni
```

```
[[1]]
[1] 0.5178887

[[2]]
[1] 0.9005588 0.7332670

[[3]]
[1] 0.1320209 0.3801821 0.5234797

[[4]]
[1] 0.0513474 0.3719633 0.7269017 0.3396311
```

```
> ll2=lapply(1:4, rnorm); ll2
```

```
[[1]]
[1] 0.1330747

[[2]]
[1] -0.6889214  0.0697643

[[3]]
[1] 0.08601576 1.85041396 1.90774763

[[4]]
[1] -0.69663339 -0.08448775 -0.87106169  0.37266470
```