Université Paris Ouest Nanterre La Défense

ANALYSE DE LA CONJONCTURE

Support de cours

Document n° 1

Année universitaire 2008 – 2009

UNIVERSITE PARIS X NANTERRE

Année universitaire 2008/2009
U.F.R. SEGMI
Licence 3ème année
Françoise Larbre

ANALYSE DE LA CONJONCTURE

Premiers pas avec le logiciel R

0. Au sujet de R

Le logiciel **R** constitue tout à la fois un langage de programmation généraliste, une énorme collection de fonctions mathématico-statistiques, et un excellent outil de tracé graphique. De plus, son utilisation par de nombreux enseignants et chercheurs a conduit à une documentation très complète de qualité « professionnelle » et directement disponible sur Internet (www.r-project.org/). Bien entendu, il existe dans les trois systèmes d'exploitation principaux actuels (Linux, MacOS et Windows).
R est une réalisation libre du langage statistique **S** (également utilisé dans le logiciel commercial **S-Plus**, avec lequel il est globalement compatible).
C'est un langage de programmation pouvant être utilisé tout aussi bien en mode conversationnel que différé (traitement par lot).
Sa programmation est extrêmement souple : procédurale, fonctionnelle ou objet. Nous nous contenterons le plus souvent de l'utiliser comme une calculette graphique sophistiquée.
Le lecteur intéressé et désireux d'approfondir ses connaissances pourra se reporter avec profit à l'onglet aide du GUI (graphical user interface).

Toutes les variables, les données, les fonctions, ... que vous utiliserez sont des objets qui seront stockés dans la mémoire de l'ordinateur. Ils peuvent alors être manipulés à l'aide d'opérateurs et de fonctions (opérateurs arithmétiques ou logiques, ... fonctions mathématiques usuelles et bien d'autres que vous découvrirez).
Lors de l'exécution d'une fonction le résultat s'affiche à l'écran ou bien est stocké dans un objet. Sauf demande explicite, **R** évite presque toujours d'afficher les résultats.

1. Installation du logiciel R

Pour disposer de la dernière version du logiciel **R**, reportez-vous au site www.r-project.org et aux quelques consignes regroupées dans le fichier Installer_R.pdf (disponible sur ma page).

1.1. Isoler les fichiers d'installation

Il est conseillé de ne pas mélanger les fichiers d'installation et vos fichiers de travail. Sans précaution supplémentaire, **R** travaille par défaut dans le répertoire d'installation. Afin d'éviter cela :

– créer un répertoire dédié à vos travaux avec le logiciel **R**, nommons-le par exemple R-Travail,
– en pointant le raccourci, faire un clic avec le bouton droit de la souris, sélectionner « Propriétés » et effacer le contenu de la case « Démarrer dans »,
– positionner la souris sur le raccourci et faites-le glisser vers le répertoire dédié (R-Travail).

Vous pouvez maintenant lancer le logiciel à partir de votre répertoire de travail. Tous les fichiers engendrés lors de votre session y seront stockés. Avant toute chose, assurez-vous que vous êtes dans le bon répertoire :

```
> getwd()
```

Si vous avez suivi nos conseils, la réponse sera (selon vos disques, ici D) :

```
[1] D:/R-Travail
```

Autre solution : choisir le répertoire de travail après le lancement de **R** (le risque est d'oublier, mais c'est programmable). On pourrait par exemple faire le choix suivant :

```
> setwd("M:/TD/Conjoncture/R-Travail")
```

On peut aussi se contenter de changer ce répertoire, sous Windows, dans le menu du GUI de **R**, via le menu déroulant de l'onglet fichier (en cliquant sur 'Changer le répertoire courant ...'). Vous pouvez vérifier avec la commande `getwd()` que vous êtes bien dans le répertoire souhaité.

1.2. Contrôler le contenu du répertoire sélectionné

```
> dir()
```

1.3. Apprentissage du langage R

Il est temps maintenant de se familiariser avec quelques fonctions.
Nous avons pris le parti de n'en présenter que quelques-unes, dans leur forme la plus simple, parmi la multitude permettant d'engendrer ou de lire des données, de les manipuler, de leur appliquer un traitement statistique ou de les représenter.

R manipule des structures de données, les plus simples sont les vecteurs, cf. le point 2. Le vecteur peut être vu comme une structure de base, au sens où ses composantes sont toutes de même type (numérique, caractère ...). Nous verrons ensuite différents types de tableaux, cf. le point 3.

Afin de traiter l'information économique et plus particulièrement les séries temporelles proposées par divers organismes, nous apprendrons à lire les données produites au format csv (et excel), cf. les points 4 et 5. Le point 6 vous indique comment réaliser quelques représentations graphiques.

Dans le point 7, deux propriétés de **R**, utiles pour la programmation, sont illustrées.

Enfin, et ce sera l'objet d'un autre document, nous verrons comment mettre en œuvre un traitement élémentaire des séries temporelles.

Chaque fois que vous souhaiterez de la documentation sur une quelconque fonction, vous taperez `?nom de la fonction`, *par exemple :*

```
> ?getwd
```

2. Les vecteurs

Vous trouverez ci-après quelques commandes permettant la création et la manipulation de vecteurs (de 1 à n composantes).

= affectation, on peut aussi utiliser <- ou bien ->

```
> x=8.2
> x
[1] 8.2
```

: permet de créer commodément une suite d'entiers de pas constant + ou -1.

```
> x=10:4
> x
[1] 10 9 8 7 6 5 4
> x=2*(10:6)-3
> x
[1] 17 15 13 11 9
> (x=2^(2*(10:6)-10))
[1] 1024 256 64 16 4
```

Observez que mettre entre parenthèse une instruction (c'est le cas du dernier exemple) implique l'affichage automatique du vecteur construit.
Quelle suite obtient-on avec l'instruction x=2^(2*(9:5)-8) ?

`seq()` permet d'engendrer une suite de nombres réels en précisant le premier élément de la suite, le dernier, et le pas souhaité.

```
> seq(1, 10, by = 0.5)
 [1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0
[16] 8.5 9.0 9.5 10.0
```

`mode()` identifie le type des données : numérique ou chaîne de caractères ou logique (vrai ou faux).

```
> mode(x)
[1] "numeric"
> y=c("a","b","c","d")
> y
[1] "a" "b" "c" "d"
> mode(y)
[1] "character"
> z=c(T,T,T,F)
> z
[1] TRUE TRUE TRUE FALSE
> mode(z)
[1] "logical"
```

`c()` permet la concaténation de données de divers modes.

```
> x=c(1,4,9,16,25,36,49)    #     ou x=(1:7)^2   ;-) !!
> x
[1] 1 4 9 16 25 36 49
> mode(x)
[1] "numeric"
> x=c(1,4,"a","b",9, 16,"c")
> x
[1] "1" "4" "a" "b" "9" "16" "c"
> mode(x)
[1] "character"
>x=(c(1,4,"a",NA,9, 16,"c"))
[1] "1" "4" "a" NA "9" "16" "c"
```

Remarque : la présence d'un seul élément introduit en mode chaîne de caractères implique que toutes les composantes de `x` sont considérées comme des chaînes de caractères.

`scan()` permet, entre autres, la saisie au clavier de données numériques

```
> x = scan()
```

et **R** attend la liste (ci-dessous un espace sépare deux données). Un 'Entrée' (↩) permet de passer à la ligne, et deux interrompent la saisie (le rang du premier objet de la ligne est indiqué) :

```
1:  7  23  2  -1   ↩
5:  1  12  -9  0  2   ↩ ↩
10:
Read 9 items
> x
[1] 7  23   2  -1  1  12  -9  0  2
```

Nous verrons d'autres manières d'utiliser `scan`.

`[ ]` permet de sélectionner des composantes du vecteur.

```
> x[3:5]
[1] 2 -1 1
> x[4]
[1] -1
> x[-4]
[1] 7 23 2 1 12 -9 0 2
```

`length()` renvoie le nombre de composantes de son argument

```
> length(x)
> [1] 9
```

3. Les tableaux

Avant de nous lancer dans la lecture de données importées, travaillons sur un exemple de taille limitée :

```
> (x=1:15)
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

puis rangeons ces données dans un fichier sous forme d'un tableau dont nous spécifions le nombre de colonnes et lisons-les.

`write()` écrit un objet dans un fichier. Ci-dessous nous spécifions dans la liste des arguments, le nom de l'objet que l'on souhaite enregistrer, le nom donné au fichier (entre guillemets), le nombre de colonnes et le type de séparateur :

```
> write(x,"fichier_perso.txt",ncolumns=5,sep=" ")
```

fichier_perso.txt se trouve maintenant dans votre répertoire de travail, vérifiez-le !

`read.table()` lit le fichier et permet d'en contrôler le contenu :

```
> read.table("fichier_perso.txt",sep=" ")
   V1 V2 V3 V4 V5
1   1  2  3  4  5
2   6  7  8  9 10
3  11 12 13 14 15
```

Si nous préférons trois colonnes au lieu de cinq :

```
> write(x,"fichier_perso.txt",ncolumns=3,sep=" ")
> read.table("fichier_perso.txt",sep=" ")
  V1 V2 V3
1  1  2  3
2  4  5  6
3  7  8  9
4 10 11 12
5 13 14 15
```

`=` permet l'affectation des données lues :

```
> fichier_test=read.table("fichier_perso.txt",sep=" ")
> fichier_test
  V1 V2 V3
1  1  2  3
2  4  5  6
3  7  8  9
4 10 11 12
5 13 14 15
```

[ ] permet alors de ne faire afficher qu'une partie du tableau, on notera surtout la manière très souple utilisée par **R** pour sélectionner des lignes et des colonnes.

```
> fichier_test[2:4,]
  V1 V2 V3
2  4  5  6
3  7  8  9
4 10 11 12
> fichier_test[,3]
[1] 3 6 9 12 15
> fichier_test[,2:3]
  V2 V3
1  2  3
2  5  6
3  8  9
4 11 12
5 14 15
```

Une structure utile, non traditionnelle : data.frame

Outre les structures traditionnelles de liste, vecteur et matrice, le logiciel **R** introduit le data.frame qui est une matrice dont les colonnes peuvent ne pas être de même type (numériques ...)
Un tableau est d'abord une liste, c'est-à-dire une collection ordonnée d'objets (cf. tableaux A et B).
Un tableau est un **data.frame** s'il contient différents types d'enregistrement : variable numérique, chaîne de caractères, variable logique. Dit autrement, **data.frame** désigne un tableau dont les colonnes peuvent être hétérogènes mais le contenu de chaque colonne doit être homogène et toutes les colonnes doivent avoir la même taille. (cf. tableau B).
Il peut avoir le type **matrix** s'il ne contient que des données homogènes (ou numériques ou caractères ou variables logiques). Ce serait le cas pour des sous-ensembles des tableaux A et B.
Un tableau réduit à une seule ligne ou à une seule colonne peut avoir le type **vector** ; c'est le cas si la ligne ou la colonne sont extraites d'un tableau de type **matrix**. Dans le cas d'une ligne extraite d'un tableau de type **data.frame**, cela n'est pas forcément le cas.
vector désigne un vecteur au sens usuel du terme.

Tableau A

	Nom 1	Nom 2	Nom 3
T1	40	20,0	60,0
T2	39	19,5	58,5
T3	18	19,5	19,5
T4	40	10,5	15,5
T5	39	15,5	39,0
T6	30	25,5	34,5
T7	36	45,5	28,0
T8	34	28,5	16,0

Tableau B

0	A	1	2
1	rouge	10	8
2	bleu	17	18
3	bleu	9	9
4	bleu	3	6
5	bleu	42	24
6	rouge	15	8
7	bleu	12	10
8	rouge	10	4

`str()` permet l'examen d'un objet quelconque de R.
On peut donc l'utiliser pour vérifier le type du tableau crée précédemment :

```
> str(fichier_test)
'data.frame':  5 obs.  of 3 variables:
$ V1:  int 1 4 7 10 13
$ V2:  int 2 5 8 11 14
$ V3:  int 3 6 9 12 15
```

`is.`*type*`()` permet d'interroger différemment

```
> is.data.frame(fichier_test)
[1] TRUE
> is.matrix(fichier_test)
[1] FALSE
> is.vector(fichier_test)
[1] FALSE
```

Remarquons que notre tableau, bien que ne contenant que des valeurs numériques, n'est pas reconnu d'emblée du type **matrix**.

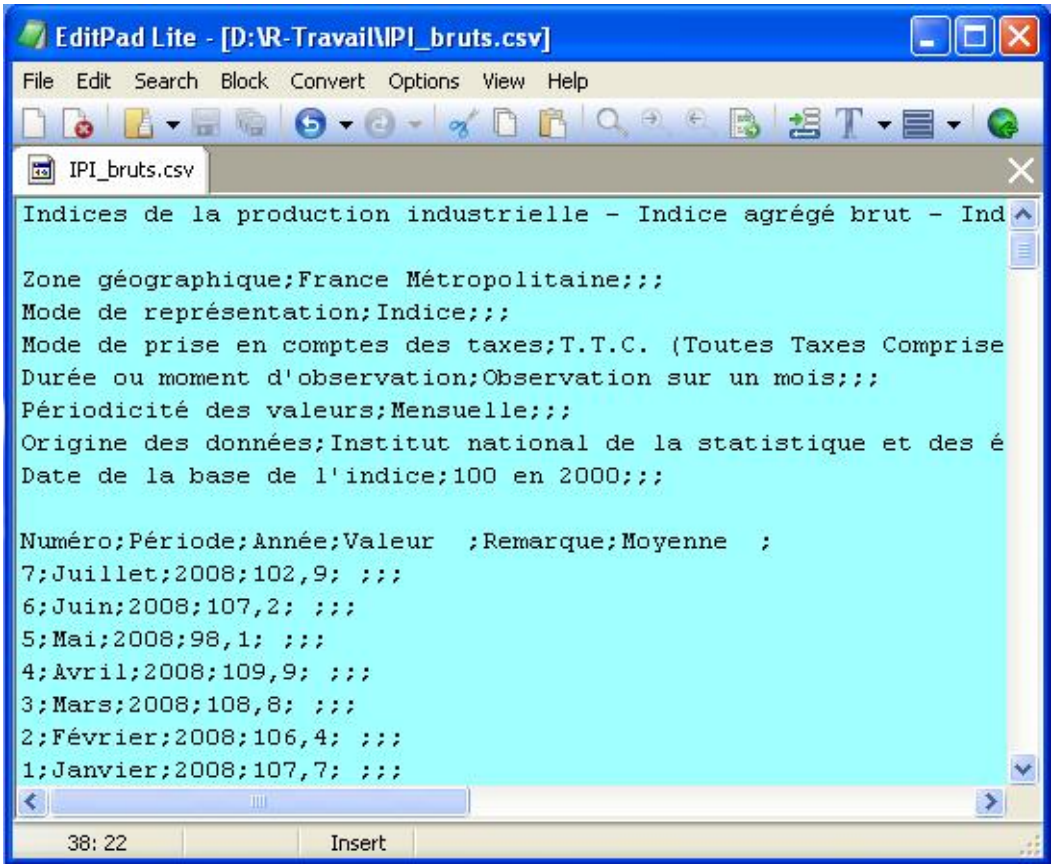
`matrix()` permet l'obtention du statut de matrice.

```
> matrice_test=matrix(fichier_test)
> is.matrix(matrice_test)
[1] TRUE
```

4. Lecture des données rangées en format csv

Vous disposez par exemple des fichier "IPI_bruts.csv" et "IPI_corrigés.csv" au format csv (Comma Separated Value) des indices de la production industrielle.

Voici une image des premières lignes du fichier des indices bruts :



Remarque : le fichier a été ouvert avec l'éditeur **EditPad Lite** (http://www.editpadlite.com/).

Dans notre exemple, les premières lignes contiennent la description des données, la lecture du fichier commencera à la onzième ligne.
Rangeons le contenu du fichier "IPI_bruts.csv" dans un objet que nous nommons IPI_bruts et affichons-le :

```
> IPI_bruts=read.table("IPI_bruts.csv", skip=10, header=TRUE, sep=";", dec=",")
> head(IPI_bruts)
  Numéro Période  Année Valeur   Remarque   Moyenne     X   X.1
1      6    Juin   2008  107.1         NA            NA    NA
2      5     Mai   2008   98.1         NA            NA    NA
3      4   Avril   2008  109.8         NA            NA    NA
4      3    Mars   2008  108.8         NA            NA    NA
5      2 Février   2008  106.4         NA            NA    NA
6      1 Janvier   2008  107.7         NA            NA    NA
```

`skip=10` *permet de sauter les 10 premières lignes du fichier*
`header=TRUE` *indique que la première ligne lue contient les titres des colonnes*
`sep=";"` *indique le type de séparateur utilisé*
`dec=","` *indique le codage de la décimale*

`head()` permet d'afficher les premiers éléments d'un vecteur, d'un tableau ou même d'une fonction.
`tail()` ...vous avez deviné

`attach()` permet d'utiliser automatiquement les noms des colonnes comme noms de variable, il faut « attacher » le fichier.

```
> attach(IPI_bruts)
> val_brutes=Valeur
head(val_brutes)
[1] 107.1 98.1 109.8 108.8 106.4 107.7
```

On a « attaché » le fichier. La variable « Valeur » contient alors les données de la colonne intitulée Valeur qui sont recopiées, dans l'exemple ci-dessus, dans la variable de travail « val_brutes ».

Vous pouvez contrôler à tout moment la nature des objets manipulés à l'aide des commandes `is.`*type*.

```
> is.data.frame(IPI_bruts)
[1] TRUE
> is.data.frame(val_brutes)
[1] FALSE
> is.vector(val_brutes)
[1] FALSE
```

ou bien avec la commande `str()` :

```
> str(IPI_bruts)
'data.frame':  222 obs.  of 8 variables:
 $ Numéro :  int 6 5 4 3 2 1 12 11 10 9 ...
 $ Période :  Factor w/ 12 levels "Août","Avril",..:  7 8 2 9 4 5 3 10 11 12 ...
 $ Année :  int 2008 2008 2008 2008 2008 2008 2007 2007 2007 2007 ...
 $ Valeur :  num 107.1 98.1 109.8 108.8 106.4 ...
 $ Remarque:  logi NA NA NA NA NA NA NA NA NA NA NA NA ...
 $ Moyenne :  num NA NA NA NA NA ...
 $ X : logi NA NA NA NA NA NA ...
 $ X.1 :  logi NA NA NA NA NA NA ...
```

Lisons également le fichier des données corrigées

```
IPI_corrigés=read.table("IPI_corrigés.csv", skip=11, header=TRUE, sep=";", dec=",")
attach(IPI_corrigés); val_corrigés=Valeur
```

5. Lecture des données rangées en format xls

Le plus simple est vraiment de convertir le fichier xls au format csv et ... de se reporter à la section précédente. Toutefois, **R** donne aussi la possibilité de lire les données dans leur format natif.

6. Représentations graphiques rapides

```
> plot(val_brutes, type="l")
> lines(val_corrigées)
```

On notera que **lines** permet de tracer une seconde courbe sans ouvrir une nouvelle fenêtre.

En jetant un coup d'oeil sur le mode d'emploi ...

```
> ?plot
```

... vous apprenez à ajouter un titre ou un peu de couleur pour mieux voir

```
> plot(val_brutes, type="l", col="red")
> lines(val_corrigées, col="black")
> plot(val_brutes, type="l")
> lines(val_corrigées)
```

Les graphiques vous permettent de remarquer, au cas où vous auriez été distrait, que les données fournies sont en ordre chronologique inverse (tendance décroissante de l'indice de la production industrielle !). Pour tracer les données en ordre chronologique il suffisait de demander :

```
> plot(rev(val_brutes), type="l", col="red")
> lines(rev(val_corrigées), col="black")
```

Nous suggérons, pour la suite de nos expériences, la conversion suivante :

```
> val_brutes=rev(val_brutes)
```

Demandons un peu de documentation sur les options graphiques par défaut, et tentons deux essais :

```
> ?plot.default
> plot(val_brutes, type="l", col="red", main="Indice de la production industrielle
01/1990 - 07/2008")
> plot(val_brutes, type="o", col="black",main="Indice de la production industrielle
01/1990 - 07/2008")
```

À vous de découvrir la multitude des possibilités offertes pour vos représentations graphiques.

7. Et si l'on programmait un tout petit peu ?

Nous aurons besoin de très peu de chose. Les deux exemples suivants illustrent deux propriétés de **R** particulièrement commodes.

7.1. La vectorisation

Un grand nombre des opérateurs de **R** utilisent les possibilités de « vectorisation » de calcul.

```
> t=1:20
> y0=100; x=0.05
> (y=y0*(1+x)^t)
[1] 105.0000 110.2500 115.7625 121.5506 127.6282 134.0096 140.7100 147.7455
[9] 155.1328 162.8895 171.0339 179.5856 188.5649 197.9932 207.8928 218.2875
[17] 229.2018 240.6619 252.6950 265.3298
```

7.2. La récursivité

L'exemple ci-dessous est la traduction en **R** du célèbre problème des tours de Hanoï [Edouard Lucas (1842-1891)]. Il s'agit de transférer un à un des disques de diamètres différents empilés sur un support A vers un support B en utilisant éventuellement un support C en respectant la contrainte suivante : un disque quelconque ne se trouve jamais entièrement recouvert. Dans quel ordre effectuer les transferts ?

```
> Hanoi=function(depart, auxiliaire, but, n)
+ { if (n>0)
+  {
+     Hanoi(depart, but, auxiliaire, n-1)
+     print(paste("Transférer", "disque", n, "de ", depart," vers ",but))
+     Hanoi(auxiliaire, depart, but, n-1)
+  }
+ }
> Hanoi("Dep","Aux","But",3)
[1] "Transférer disque 1 de Dep vers But"
[1] "Transférer disque 2 de Dep vers Aux"
[1] "Transférer disque 1 de But vers Aux"
[1] "Transférer disque 3 de Dep vers But"
[1] "Transférer disque 1 de Aux vers Dep"
[1] "Transférer disque 2 de Aux vers But"
[1] "Transférer disque 1 de Dep vers But"
```

On notera :
– la définition de la fonction `Hanoi`
– l'usage de l'instruction de test `if`
`if (expression logique) { instruction si VRAI} else { instruction si FAUX}`,
dans l'exemple ci-dessus, on ne fait rien si la condition `n>0` n'est pas vérifiée.

7.3. Boucle explicite

Voici un exemple « simpliste » :

```
>x=rnorm(100)*100
>(x=rnorm(100)*100)
 [1]  182.4622542  197.9497053   -5.7731901   36.1152678   79.6508228   58.6816463
 [7] -171.0383432   72.4186649   64.1981405  308.3936235  110.7209982  -36.3420826
[13] -176.9118426 -238.1343307  215.9923932  108.8804818   -2.1344407  -42.9693479
[19]  -78.6973269 -190.4444337 -128.1634757 -119.9504734  198.9930562   30.7630557
[25]   47.1324562 -127.6774015  127.2394384   84.0285950   28.1713950 -102.0705007
[31]   81.4934375 -100.7281536  -66.2409979 -122.5758233 -113.4890982   56.8729289
[37]  -57.3158957  -77.5614640  174.4872525   29.2269592  -53.5007166   -5.3242461
[43]  -48.5742266  101.3779272  -64.5671762  -64.7098391 -132.4320382  -99.0871875
[49]   75.2337027 -111.5925877   14.8546335   32.6955308  136.3275626 -101.5496646
[55]  140.4715767  -82.7679074  -92.6668732 -110.7238044   58.9968338  -11.8210580
[61]  -45.3634144   76.6166725  -69.7323626    7.9722472 -101.7843445   92.2555061
[67]  -31.2150421  152.6169375  -46.4755814  -71.0065138  -24.6482370 -143.9030522
[73]  -45.3676040   70.7637960 -150.2762483   87.2668218   45.5371829 -130.2626773
[79] -155.6225511  144.9843667  -65.8433823  -25.3177648   21.5703151  115.7723700
[85]   -0.4462429  164.9947139   65.6091837 -121.0930996  157.2191915  -29.0967283
[91]   60.5377769  161.6106794 -274.2043533   63.1185324   55.9582129  -91.5659278
[97] -110.0504135  107.1393637  107.2042714  132.0750188
>compteur=0
>for (i in 1:100)
>if(x[i]<0) compteur=compteur+1
compteur
[1] 52
```

`rnorm(100)` *engendre 100 réalisations d'une loi normale centrée réduite*

Parmi ces 100 valeurs nous identifions celles qui sont négatives, et nous les comptons.

Il faut remarquer que le même résultat s'obtient de manière plus élégante et efficace en remplaçant la boucle par un calcul « vectorisé » :

```
>which(x<0)
[1]   1 2 4 6 7 8 10 11 16 17 18 22 24 25 28 31 33 35 36 40 43 45 46
[24] 50 51 52 54 57 60 62 64 67 68 69 71 72 74 75 77 78 83 88 89 91 92 95
[47] 99 100
>length(which(x<0))
[1] 48
```

`which(x<0)` *sélectionne les valeurs négatives*

`length(which(x<0))` *indique la longueur de la liste obtenue donc le nombre d'éléments satisfaisant la condition.*