Vectorisation et tests

Timothée Poisot

October 16, 2013

Dans l'épisode précédent

1. Comment écrire une fonction
2. L'importance de bien penser à son algorithme

Solution de l'exercice

On génére N points positionnés *au hasard* dans le rectangle ou les deux cercles sont inscrits

On compte combien de points sont dans au moins un cercle (U) et combien sont dans les deux (D)

L'aire relative de la surface ou les cercles se recouvrent est U/D

Version R: `advanced/seance1_cercles.r`

Programme de la séance

1. Les tests
2. La vectorisation
3. Dynamiques écologiques neutres

Les tests

Principe général: **si** une condition, **alors** une instruction (**sinon**, autre chose)

Par exemple

```
pour tous les nombres i entre et 10
    si i est pair
        afficher i
    sinon
        afficher i + 1
```

Les tests

```
pair = function(x) (x%%2) == 0

for (i in c(1:10)) {
    if (pair(i)) {
        print(i)
    } else {
        print(i + 1)
    }
}
```

```
## [1] 2
## [1] 2
## [1] 4
## [1] 4
## [1] 6
## [1] 6
## [1] 8
```

Pour faire court en R

```
a = 2
b = ifelse(a < 3, 0, 1)
b
```

```
## [1] 0
```

Le type booléen

Prend deux valeurs: `vrai` et `faux`

Dans R: `TRUE`, `FALSE`, `T`, `F`, *mais aussi* `1`, `0`

Par exemple:

```
a = 2
a == 2
```

```
## [1] TRUE
```

```
a + 3 > 3
```

```
## [1] TRUE
```

```
(a + 1 > 3) + 1
```

```
## [1] 1
```

Comparaisons: *et* logique

```
TRUE & TRUE
```

```
## [1] TRUE
```

```
TRUE & FALSE
```

```
## [1] FALSE
```

```
FALSE & FALSE
```

```
## [1] FALSE
```

Comparaisons: *ou* logique

```
TRUE | TRUE
```

```
## [1] TRUE
```

```
TRUE | FALSE
```

```
## [1] TRUE
```

```
FALSE | FALSE
```

```
## [1] FALSE
```

Comparaisons: précédence

```
TRUE | FALSE & TRUE
```

```
## [1] TRUE
```

```
TRUE | (FALSE & TRUE)
```

```
## [1] TRUE
```

```
(TRUE | FALSE) & TRUE
```

```
## [1] TRUE
```

Comparaisons

```
TRUE + TRUE
```

```
## [1] 2
```

```
TRUE * FALSE
```

```
## [1] 0
```

```
TRUE + FALSE
```

```
## [1] 1
```

Comparaisons: *ou exclusif*

```
xor(TRUE, FALSE)
```

```
## [1] TRUE
```

```
xor(FALSE, FALSE)
```

```
## [1] FALSE
```

```
xor(TRUE, TRUE)
```

```
## [1] FALSE
```

Comparaisons: *non*

```
TRUE
```

```
## [1] TRUE
```

```
!TRUE
```

```
## [1] FALSE
```

```
!FALSE
```

```
## [1] TRUE
```

Exercice - programmer le *ou exclusif*

Rappel: (prédicat 1 *ou* prédicat 2) *mais pas* (prédicat 1 *et* prédicat 2)

Exercice - en R

```
ouExcl = function(pr1, pr2) (!(pr1 & pr2)) & (pr1 | pr2)
xor(T, F)
```

```
## [1] TRUE
```

```
ouExcl(T, F)
```

```
## [1] TRUE
```

```
xor(T, T)
```

```
## [1] FALSE
```

```
ouExcl(T, T)
```

```
## [1] FALSE
```

Rappel - vecteur

Dans R, un vecteur est un object avec plusieurs éléments, numérotés de `1` à `length(objet)`

On accède à l'élément à la position `i` avec `objet[i]`

```
a = seq(from = 0, to = 3, length = 9)
print(a[1])
```

```
## [1] 0
```

```
print(a[3])
```

```
## [1] 0.75
```

Rappel - matrices

Une matrice a deux dimensions, allant de `1` à `nrow(matrice)` et `ncol(matrice)`

On accède à la ligne `i` par `matrice[i,]`, à la colonne `j` par `matrice[j]`, et à l'élément `i,j` par `matrice[i,j]`

Rappel - matrices

```
b = matrix(c(1:4), nrow = 2)
print(b)
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

```
print(b[1, 2])
```

```
## [1] 3
```

```
print(b[2, 1])
```

```
## [1] 2
```

La vectorisation

1. Permet d'accéder rapidement a des éléments de vecteurs
2. Automatise le traitement des vecteurs
3. **Central** pour écrire du code `R` efficace

La vectorisation - accès

```
ve = c(1, 2, 3, 5, 8, 13)
ve[1]
```

```
## [1] 1
```

```
ve[c(1, 3, 4)]
```

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

```
ve <= 5
```

```
## [1]  TRUE  TRUE  TRUE  TRUE FALSE FALSE
```

```
ve[ve <= 5]
```

```
## [1] 1 2 3 5
```

La vectorisation - opérations

```
ve = c(1:3)
vpl2 = c()
for (i in c(1:length(ve))) vpl2[i] = ve[i] + 2
vpl2
```

```
## [1] 3 4 5
```

```
ve + 2
```

```
## [1] 3 4 5
```

La vectorisation - répétitions

```
replicate(4, 10)
```

```
## [1] 10 10 10 10
```

Mouvement brownien

(*en résumé*) une mouvement dans lequel on se déplace aléatoirement à partir de son état actuale (*random walk*)

$$x_{t+1} = x_t + \mathcal{N}(0, \sigma)$$

Mouvement brownien

```
brownian = function(x0 = 0, steps = 10) {
    x = c(x0)
    for (i in c(2:steps)) {
        x[i] = x[(i - 1)] + rnorm(1, 0, 0.05)
    }
    return(x)
}

brownian(steps = 4)
```

```
## [1]  0.0000000 -0.0006817  0.0709960  0.1881901
```

Mouvement brownien

```
for (i in c(1:5)) {
    print(brownian(steps = 3))
}
```

```
## [1]  0.000000 -0.062936 -0.009519
## [1]  0.00000 -0.11086 -0.07223
## [1] 0.00000 0.06029 0.05691
## [1]  0.00000 -0.01774 -0.01945
## [1]  0.00000 -0.01925 -0.05899
```

Mouvement brownien

```
replicate(5, brownian(steps = 3))
```

```
##             [,1]    [,2]    [,3]      [,4]     [,5]
## [1,] 0.000000 0.00000 0.00000  0.000000  0.00000
## [2,] 0.008055 0.01034 0.06162 -0.011972 -0.04312
## [3,] 0.019478 0.03119 0.05261 -0.006249 -0.02353
```

Mouvement brownien

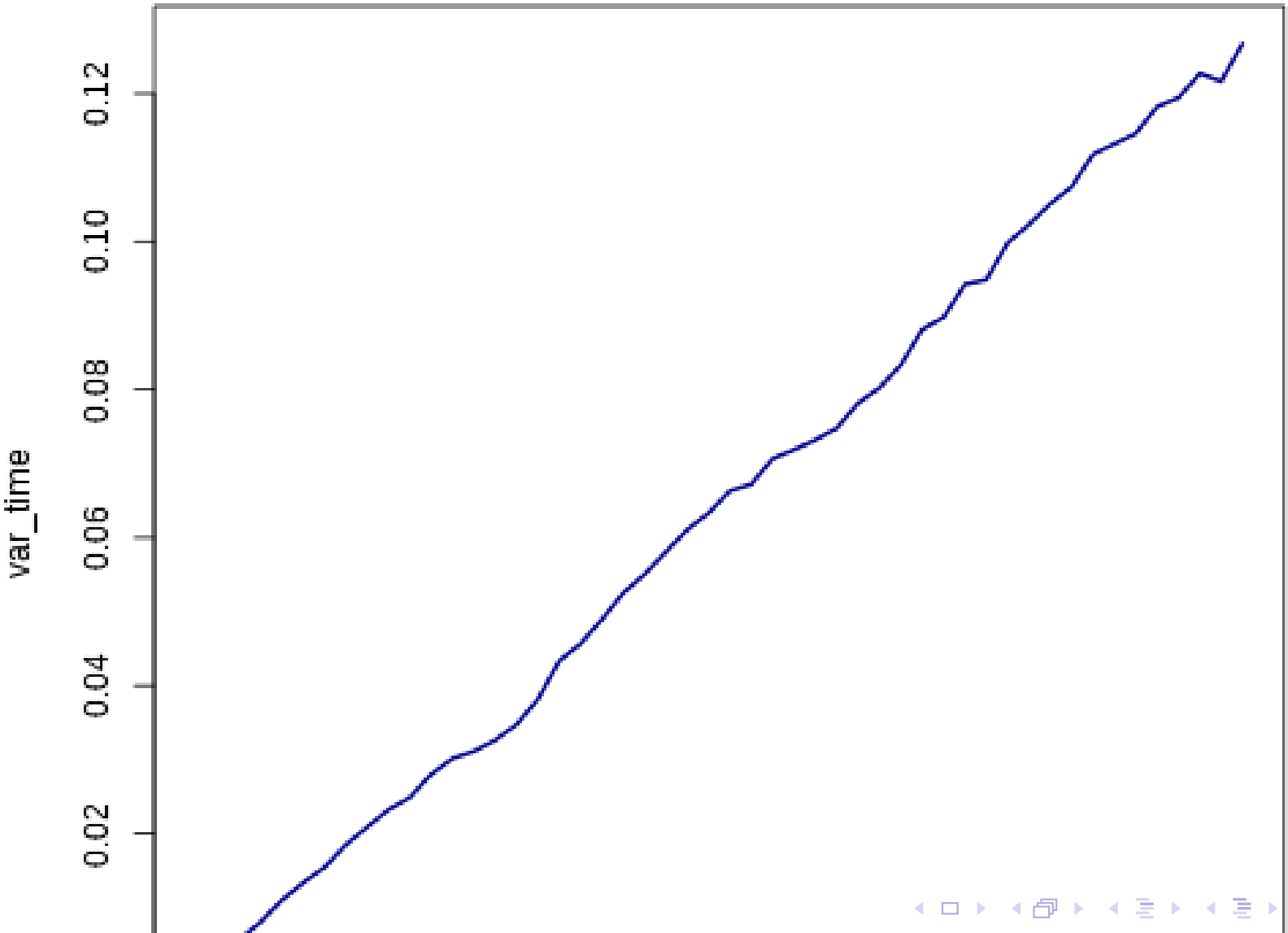
```
walk = replicate(300, brownian(steps = 50))

var_time = apply(walk, 1, var)

print(head(var_time))

## [1] 0.000000 0.002381 0.005618 0.007971 0.010988 0.0133
```

Mouvement brownien



apply

```
apply(matrice, DIM, FUN)
```

- DIM: 1 pour les lignes, 2 pour les colonnes
- FUN: toute fonction prennant un vecteur comme argument

apply - définition en-ligne

```
coef_var = apply(walk, 1, function(x) var(x)/mean(abs(x)))
plot(coef_var, type = "l", lwd = 2, col = "darkgreen")
```