# Projet R 2

Cours de Programmation

*Vittorio Perduca, Master 1 Mathématiques et Applications*

*UFR Math-Info, Université Paris Descartes, automne 2017*

## Table des matières



Dans ce projet on étudie les propriétés statistiques de deux jeux de société par simulations.

## 0. Instructions

- Vous rendrez un rapport écrit sous format `.Rmd` **et** `.pdf` (ou `.html`).
- Vous enverrez vos fichiers `NOM1_NOM2.Rmd` et `NOM1_NOM2.pdf` dans un email avec le sujet **Projet Programmation R 2017** à mon adresse vittorio.perduca@parisdescartes.fr
- Vous écrirez les noms des auteurs dans le format **NOM1 Prénom1, NOM2 Prénom2.**
- Vous écrirez vos programmes `R` complets dans des blocs de code, de façon à ce que le lecteur puisse les faire tourner facilement, et vos commentaires et explications dans des paragraphes de texte.
- La notation prendra en compte le soin et la clarté des réponses.

## 1. Exercice d'échauffement

### 1.1 Prise en main

On commence par un jeu très simple, avec une urne contenant huit boules noires, une urne vide, six boules rouges et une pièce de monnaie. Chaque tour du jeu consiste en deux opérations :

- on tire la pièce
- si on obtient pile, on enlève une boule noire de la première urne. Si on obtient face, on met une boule rouge dans la deuxième urne.

On répète ces deux opérations jusqu'à ce qu'il y n'ait plus de boules noires dans la première urne ou jusqu'à ce que la deuxième urne contienne toutes les six boules rouges.

On se pose la question d'estimer le nombre moyen de tours dans une partie : pour cela on simulera un très grand nombre de parties pour ensuite calculer la moyenne de leur durées.

Chaque tour de la partie peut être représenté par un vecteur `tour` avec quatre composantes : le nombre de boules noires dans la première urne, le nombre de boules rouges dans la deuxième urne, le résultat du lancement de la pièce, et le numéro du tour. Une partie sera donc représentée par le data frame `partie` dont les lignes sont les tours.

On peut simuler une partie avec la fonction suivante :

```r
une_partie1=function(){
  #initialisation
  tour=c(urne1=8,urne2=0,piece=NA,ntour=0)
  partie=rbind(tour)

  #tours
  while(tour[1]!=0 & tour[2]!=6){
    tour[4]=tour[4]+1

    piece=sample(1:2,size=1) #codes: 1=pile, 2=face
    tour[3]=piece

    tour[piece]= -(piece==1) + (piece==2) + tour[piece]

    partie=rbind(partie,tour)
  }
  return(partie)
}

une_partie1()
```

```
##      urne1 urne2 piece ntour
## tour     8     0    NA     0
## tour     8     1     2     1
## tour     8     2     2     2
## tour     8     3     2     3
## tour     7     3     1     4
## tour     7     4     2     5
## tour     7     5     2     6
## tour     6     5     1     7
## tour     6     6     2     8
```

On simule $n = 300$ parties, on récupère la durée de chaque partie simulée $l_i$ et on estime la durée moyenne par

$$\hat{l} = \frac{\sum_{i=1}^{n} l_i}{n}$$

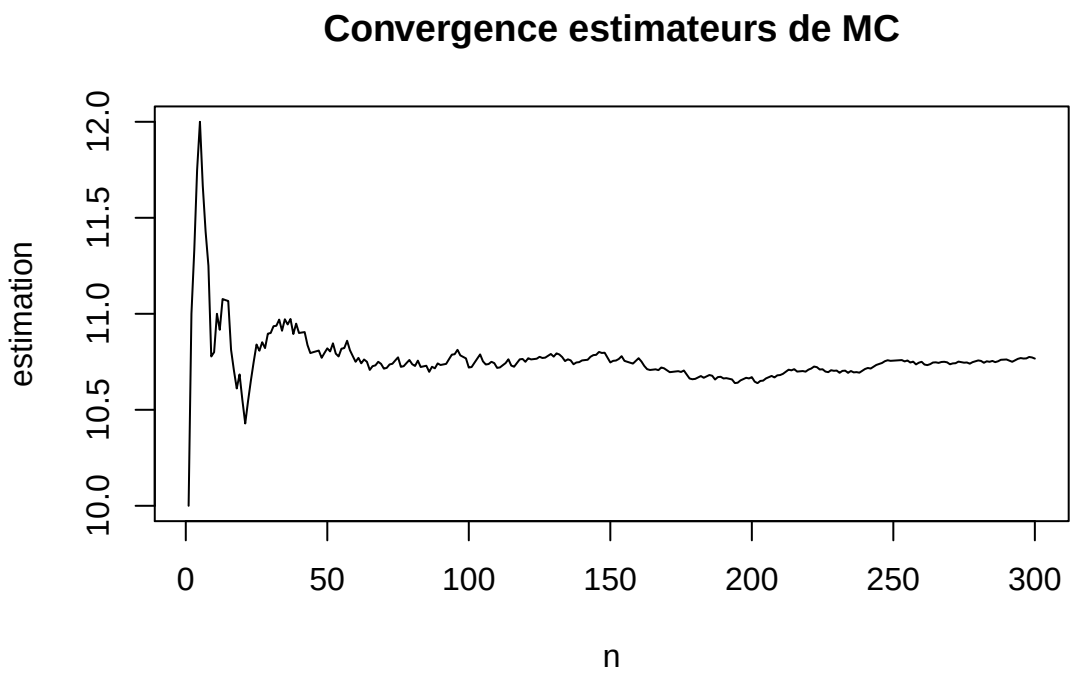
Cette procédure d'estimation est dite estimation par **méthode de Monte-Carlo.**

```r
n=300
res=replicate(n,une_partie1())
duree=sapply(res,FUN=function(partie) nrow(partie)-1)
mean(duree)
```

```
## [1] 10.76667
```

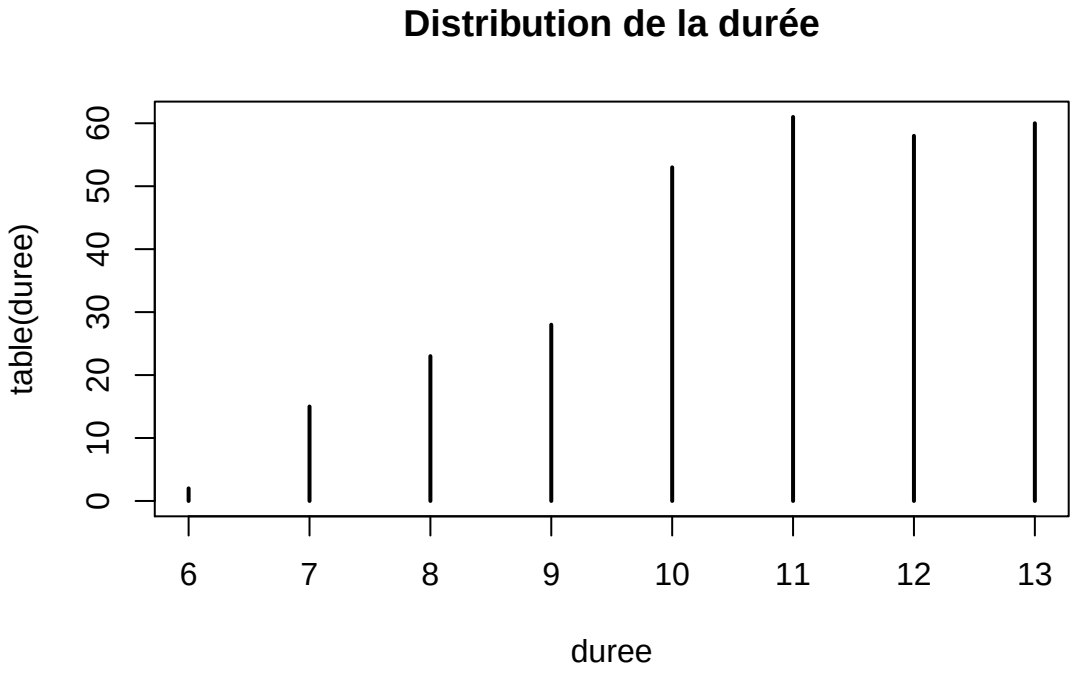
On peut monitorer la converge de l'estimation pour savoir si le $n$ choisi est suffisant :

```r
plot(cumsum(duree)/(1:n),type='l',
     main='Convergence estimateurs de MC',
     xlab='n',
     ylab='estimation')
```




On peut aussi regarder la distribution empirique de la durée :

```
plot(table(duree),main='Distribution de la durée')
```




### 1.2 Estimation de la probabilité de gagner

Dans le jeu précédent on gagne la partie si on vide la première urne avant de remplir complètement la deuxième urne de boules rouges. Dans ce dernier cas, on perd la partie.

1. Ecrire une fonction qui prend en argument la sortie de `une_partie1()` et rend `1` si la partie se termine avec une victoire et `0` si la partie se termine avec une défaite.


FIGURE 1 – Premier Verger est un jeu pour enfants à partir de 2 ans.

2. Estimer la probabilité de gagner et l'écart type de l'estimateur par la méthode de Monte-Carlo.

## 2. Le jeu Premier Verger

On considère maintenant le jeu Premier Verger de Haba.

3. Lire attentivement les instructions qu'on peut trouver à l'adresse http://w3.mi.parisdescartes.fr/~vperduca/programmation/instructions_premier_verger.pdf.

On remarque que dans ce jeu, on perd la partie si le corbeau dépasse la dernière carte du chemin et se retrouve dans le verger avant que tous les fruits ne soient cueillis. Par ailleurs, le joueur a la possibilité de choisir un arbre si le dé tombe sur la face *Panier à fruits*. Il est donc possible d'implémenter une stratégie pour essayer d'augmenter les chances de gagner.

4. D'un point de vue intuitif, quelle stratégie adopteriez-vous pour maximiser la chance de gagner une partie ?
5. Ecrire une fonction `une_partie2()` qui simule une partie en implémentant la stratégie ci-dessus. De chaque tour on retiendra le nombre de fruits sur chaque arbre, la position du corbeau (sur la 1e, 2e, 3e, 4e ou 5e carte du chemin), le nombre de fruits dans le panier et le résultat du dé. Attention : un arbre ne peut pas contenir un nombre de fruits négatif ! Par rapport au jeu précédent, vous devrez gérer la situation où le dé tombe sur la face d'un arbre qui ne contient pas de fruits.
6. Estimer par la méthode de Monte-Carlo, la probabilité de gagner une partie, la durée moyenne des parties et l'écart type de l'estimateur.
7. Comparer les résultats avec ceux obtenus en utilisant une stratégie alternative.