

# **Compte rendu**

## **TD2 - R**



## Question 1 :

### 1/ La fonction des k-arrangements :

Il est demandé de coder une fonction calculant les k-arrangements dont la formule est donnée par le théorème 33 du cours.

**Définition 32.** Soient  $n$  et  $k$  deux naturels non nuls avec  $k \leq n$ . Lorsqu'on effectue  $k$  tirages **sans remise** dans une urne à  $n$  éléments, chaque résultat final (complet) possible est appelé un **k-arrangement** des  $n$  objets.

**Théorème 33.** Le nombre de  $k$ -arrangements possibles de  $n$  objets est :

$$A_n^k = \underbrace{n \times (n-1) \times \dots \times (n-k+1)}_{k \text{ facteurs}}.$$

(lire : A n k)

On observe que pour calculer les k-arrangements, il suffit de faire un produit de tous les entiers situés dans l'intervalle  $[[n-k+1;n]]$ .

```
karrangements <- function(k, n){  
  condition = k <= n && k != 0 && n!=0  
  if(condition){  
    y <- prod(seq(n-k+1, n))  
    return(y)  
  }else{  
    return(FALSE)  
  }  
}
```

La fonction prend en paramètres le nombre  $k$  de répétitions et  $n$  le nombre d'objets interchangeables de l'urne.

Conformément à la définition 32, on vérifie que  $k$  est inférieur ou égal à  $n$  et que  $n$  et  $k$  sont différents de 0. Le résultat de cette opération booléenne est stocké dans la variable `condition`. Si la variable `condition` vaut **FALSE** alors on retourne **FALSE**.

Ensuite on calcule l'output à l'aide de deux fonctions de R. `seq()`, paramétrée de cette façon, retourne l'intervalle  $[[n-k+1;n]]$  (par défaut le pas entre chaque valeur est de 1) sous forme de vecteur.

Puis on donne à la fonction `prod()` l'intervalle  $[[n-k+1;n]]$  (vecteur) pour qu'elle puisse calculer le produit de tous les  $x \in [[n-k+1;n]]$ .

On retourne ensuite le résultat de cette opération.

IN :

```
nbkarrangements <- karrangements(2, 365)
```

OUT :

```
[1] 132860
```

2/ La fonction des  $k$ -combinaisons :

Il est demandé de coder une fonction calculant les  $k$ -combinaisons dont la formule est donnée par le théorème 39 du cours.

**Définition 38.** Soient  $n$  et  $k$  deux naturels non nuls avec  $k \leq n$ . Lorsqu'on effectue  $k$  tirages **sans remise** dans une urne à  $n$  éléments, et qu'on **ignore à la fin l'ordre** dans lequel les objets ont été obtenus, chaque résultat final (complet) possible est appelé une  **$k$ -combinaison** des  $n$  objets.

**Théorème 39.** Le nombre de  $k$ -combinaisons possibles de  $n$  objets est :

$$C_n^k = \frac{A_n^k}{A_k^k} = \frac{n!}{k! (n-k)!} = \underbrace{\frac{n}{k} \times \frac{n-1}{k-1} \times \dots \times \frac{n-k+1}{1}}_{k \text{ facteurs}}.$$

(lire : C n k)

Dans un premier temps, on peut se dire qu'il suffit d'utiliser la fonction **karrangements()** définie dans la première partie de la question et qu'il suffit de retourner: karrangements(k, n)/karrangements(k, k) cependant voici ce qu'il se produit lorsque  $k$  est "grand"

IN :

```
nbkcombinaisons <- karrangements(997, 1000) / karrangements(997, 997)
```

OUT :

```
[1] Inf
```

Le script retourne "Inf" qui est le mot clé en R pour désigner  $+\infty$ , ce qui est problématique pour effectuer des calculs.

Avec l'exemple 37 du cours, on comprend pourquoi le script renvoie Inf. Il y a en fait un nombre trop conséquent de multiplications pour R.

Ainsi pour minimiser le nombre d'opérations, on utilise l'exemple 44 du cours à savoir :

## Exemple 44.

$$C_{100}^{97} = C_{100}^3 = \frac{100 \times 99 \times 98}{3 \times 2 \times 1}$$

```
kcombinaisons <- function(k, n){
  condition = k <= n && k != 0 && n!=0
  if(condition){
    if(k>n/2){
      k = n-k
    }
    numérateur = seq(n-k+1, n)
    dénominateur = seq(1, k)
    resultat_division = numérateur/dénominateur
    return(prod(resultat_division))
  }else{
    return(FALSE)
  }
}
```

Tout comme la fonction **karrangements()**, **kcombinaisons()** prend en paramètres le nombre k de répétitions et n le nombre d'objets interchangeables de l'urne.

Conformément à la définition 38, on vérifie que k est inférieur ou égal à n et que n et k sont différents de 0. Le résultat de cette opération booléenne est stocké dans la variable condition. Si la variable condition vaut **FALSE** alors on retourne **FALSE**.

Comme  $kcombinaisons(k, n) = kcombinaisons(n-k, n)$  (cf. deuxième point de la propriété 41), si  $k > n / 2$  alors k prend comme nouvelle valeur n - k, ce qui permet de réduire **DRASTIQUEMENT** le nombre de multiplications dans le cas où n est grand et k est très proche de n.

Le numérateur prend comme valeur le vecteur qui représente l'intervalle  $[[n-k+1;n]]$

Le dénominateur prend comme valeur k!.

On divise ensuite les deux vecteurs entre-eux avant d'effectuer le produit dans le but d'alléger les multiplications. On stocke le résultat dans la variable *resultat\_division* qui contient un vecteur.

La division de deux vecteurs divise chaque élément du vecteur au numérateur avec l'élément correspondant du vecteur au dénominateur et retourne un vecteur.

Soient  $v1$  et  $v2$  deux vecteurs de taille  $n$ , on a :

$$v1 / v2 = c(v1[1] / v2[1], v1[2] / v2[2], \dots, v1[n] / v2[n])$$

On retourne ensuite le produit des éléments du vecteur *resultat\_division*.

**IN :**

```
nbkcombinaisons <- kcombinaisons(997, 1000)
```

**OUT :**

```
[1] 166167000
```

## Question 2 :

### 1/ Première situation :

Il est ici demandé de reproduire sous forme de fonction la situation de la question 1 de l'exercice 2.7, la fonction prend en paramètres  $n \geq 2$  qui est la taille de l'assemblée et  $N \geq 1$  qui est le nombre d'itérations de "l'expérience".

On a donc une assemblée de  $n$  personnes et on veut répondre à la question suivante : "à quelle fréquence observe-t-on au moins une coïncidence de deux anniversaires dans l'assemblée ?".

Pour des raisons de simplicité, on suppose que tous les membres de l'assemblée ne sont pas nés dans une année bissextile. On associe à chaque jour de l'année un entier  $x \in [[1;365]]$ .

- 1  $\mapsto$  1 Janvier
- 2  $\mapsto$  2 Janvier
- ...
- 31  $\mapsto$  31 Janvier
- ...
- 365  $\mapsto$  31 Décembre

On sait que  $p(n)$  est la probabilité d'observer au moins une coïncidence de deux anniversaires et que  $q(n)$  est la probabilité que les  $n$  dates d'anniversaires soient deux à deux différentes.

On sait aussi que  $q(n)$  est donnée par la formule :

$$q(n) = \text{karrangements}(n, 365) / 365^n$$

et que  $p(n)$  est donnée par la formule:

$$p(n) = 1 - q(n)$$

On a le code suivant :

```
fonction_exo1 <- function(n, N){  
  q <- karrangements(n, 365) / pow(365, n)  
  p <- 1 - q  
  nb_succes = 0  
  for(i in 1:N){  
    dateAssemblee <- sample(x=1:365, size=n, replace=TRUE)  
    dateAssembleeUnique <- unique(dateAssemblee)  
    if(length(dateAssembleeUnique) != length(dateAssemblee)){  
      nb_succes <- nb_succes + 1  
    }  
  }  
  frequence = nb_succes/N  
  return(list("p(n) :", p, "f(n) :", frequence))  
}
```

Dans un premier temps, on commence par calculer  $q(n)$  et  $p(n)$ . Puis on initialise une variable `nb_succes` à 0 qui, comme son nom l'indique, contiendra le nombre de succès, donc le nombre de fois où  $p$  s'est produit.

On démarre une boucle `for()` que l'on répète  $N$  fois. A l'intérieur, on récupère une assemblée de  $n$  personnes générée de façon aléatoire à l'aide de la fonction `sample()`. Le paramètre `x` permet de d'indiquer à la fonction dans quelles données elle doit piocher (ici le vecteur `1:365` donc l'intervalle `[[1;365]]` qui représente les dates d'anniversaires) et le paramètre `size` est la taille de l'échantillon (ici  $n$  car le nombre de personne dans l'assemblée est  $n$ ), `sample()` retourne un vecteur généré aléatoirement en fonction des paramètres précisés.

Il faut maintenant savoir si  $p$  s'est produit donc si dans l'échantillon on observe au moins deux entiers identiques (au moins une coïncidence de deux anniversaires). Pour cela, il existe la fonction `unique()` qui permet de supprimer les doublons dans un vecteur et qui retourne un nouveau vecteur sans doublons. Exemple : `monVecteur = c(1, 1, 2, 2, 3, 3)`, Si on passe ce vecteur dans la fonction `unique()` on obtient le vecteur `c(1, 2, 3)`.

Ainsi, on peut dire que si la taille de `dataAssemblee` est différente de la taille de `dataAssembleeUnique` alors  $p$  s'est produit. Pour avoir la taille d'un vecteur, il suffit d'utiliser la fonction `length()` qui permet en partie de connaître la taille d'un vecteur. `length()` retourne un entier  $\geq 1$  dans notre cas.

Si  $p$  s'est produit, alors il y a succès, c'est pourquoi on incrémente la variable `nb_succes`.

Pour obtenir la fréquence, il suffit de de diviser le nombre de succès par le nombre d'essais.

Enfin on retourne sous forme de liste  $p(n)$  et  $f(n)$  pour pouvoir comparer la probabilité de succès avec la fréquence du succès.

**IN :**

```
exo1 <- fonction_exo1(30, 1000)
```

**OUT :**

```
[[1]]  
[1] "p(n) :"
```

```
[[2]]  
[1] 0.7063162
```

```
[[3]]  
[1] "f(n) :"
```

```
[[4]]  
[1] 0.732
```

**IN :**

```
exo1 <- fonction_exo1(30, 100000)
```

**OUT :**

```
[[1]]  
[1] "p(n) :"
```

```
[[2]]  
[1] 0.7063162
```

```
[[3]]  
[1] "f(n) :"
```

```
[[4]]  
[1] 0.7046
```



Plus N est grand et plus la fréquence de succès se rapproche de la probabilité de succès (Loi des grands nombres).

## *2/ Deuxième situation :*

La deuxième situation est presque similaire à la première, c'est pourquoi je ne détaillerai que les nouveautés/différences de cette dernière par rapport à la première.

Il est ici demandé de reproduire sous forme de fonction la situation de la question 2 de l'exercice 2.7, la fonction prend en paramètres  $n \geq 2$  qui est la taille de l'assemblée et  $N \geq 1$  qui est le nombre d'itérations de "l'expérience".

On a donc une assemblée de  $n$  personnes et on veut répondre à la question suivante : "à quelle fréquence observe-t-on au moins une coïncidence d'anniversaire entre une personne de l'assemblée et moi-même ?".

On sait que  $U(n)$  est la probabilité d'observer au moins une coïncidence d'anniversaire avec moi-même et que  $V(n)$  est la probabilité de ne pas observer de coïncidence.

On sait aussi que  $V(n)$  est donnée par la formule :

$$V(n) = (364/365)^n$$

et que  $U(n)$  est donnée par la formule:

$$U(n) = 1 - V(n)$$

On a le code suivant :

```
fonction_exo2 <- function(n, N){
  Vn <- pow(364/365, n)
  Un <- 1 - Vn
  nb_succes = 0
  maDate = sample(x=1:365, size=1, replace=TRUE)
  for(i in 1:N){
    dateAssemblee <- sample(x=1:365, size=n, replace=TRUE)
    if(maDate %in% dateAssemblee){
      nb_succes <- nb_succes + 1
    }
  }
  frequence = nb_succes/N
  return(list("Un :", Un, "f(n) :", frequence))
}
```

On commence par calculer  $V(n)$  et  $U(n)$  grâce aux formules définies ci-dessus

Ensuite, on prélève de façon aléatoire une date de naissance qui représentera la date de naissance de l'individu extérieur à l'assemblée.

On change également la condition de succès, **unique()** et **length()** sont désormais inutiles, on se concentrera ici sur le l'opérateur **%in%**, **if(maDate %in% dateAssemblee)** se traduit par "Si il y a une coïncidence d'anniversaire entre une personne de l'assemblée et moi-même" donc si **maDate** est dans **dateAssemblee**. Si oui, alors **Un** s'est produit.

Le reste de la fonction est identique à celle de la première situation.

**IN :**

```
exo2 <- fonction_exo2(365, 1000)
```

**OUT :**

```
[[1]]
[1] "Un :"
```

```
[[2]]
[1] 0.6326251
```

```
[[3]]
[1] "f(n) :"
```

```
[[4]]  
[1] 0.623
```

**IN :**

```
exo2 <- fonction_exo2(365, 100000)
```

**OUT :**

```
[[1]]  
[1] "Un :"
```

```
[[2]]  
[1] 0.6326251
```

```
[[3]]  
[1] "f(n) :"
```

```
[[4]]  
[1] 0.63197
```

### Question 3 :

Il est demandé ici de détailler les étapes de la fonction  $f()$  et d'expliquer son utilité.

La fonction  $f$  prend comme seul paramètre  $x$  un entier  $> 0$ , elle initialise une variable  $y$  à 1 puis elle démarre une boucle qui itère  $x$  fois. On observe ensuite dans cette boucle 3 opérations sur 3 vecteurs.

Première itération :

$$\begin{aligned}j &= c(0, 1) \\k &= c(1, 0) \\y &= c(1, 1)\end{aligned}$$

Deuxième itération :

$$\begin{aligned}j &= c(0, 1, 1) \\k &= c(1, 1, 0) \\y &= c(1, 2, 1)\end{aligned}$$

Troisième itération :

$$\begin{aligned}j &= c(0, 1, 2, 1) \\k &= c(1, 2, 1, 0) \\y &= c(1, 3, 3, 1)\end{aligned}$$

....

Jusqu'à l'itération  $x$ .

Puis la fonction retourne  $y$ . Pour  $y$  voir plus clair affichons  $y$  à chaque itération de la boucle pour  $x = 5$ .

**OUT :**

```
[1] 1 1
[1] 1 2 1
[1] 1 3 3 1
[1] 1 4 6 4 1
[1] 1 5 10 10 5 1
k=0 k=1 k=2 k=3 k=4 k=5
1 5 10 10 5 1
```

On reconnaît le triangle de Pascal, on en déduit donc que le programme retourne sous forme de vecteur la ligne  $n=x$  du triangle de Pascal, dans le cas où  $x$  vaut 5 la fonction retourne donc  $c(1, 5, 10, 10, 5, 1)$ .

Dans la boucle **for()** on exploite en fait la relation de pascal donnée par la formule :

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

Pour des raisons de clarté on réécrit la fonction **f()** de la façon suivante :

```
triangle_pascal <- function(n) {  
  # x doit etre un entier > 0  
  ligne_n <- 1  
  for (i in 1:n) {  
    ligne_n_1 <- c(0, ligne_n)  
    ligne_n_2 <- c(ligne_n, 0)  
    ligne_n <- ligne_n_1 + ligne_n_2  
  }  
  names(ligne_n) <- paste( "k=", 0:n, sep="" )  
  return(ligne_n)  
}
```

Le troisième point de la propriété 41 du cours nous dit que les k-combinaisons suivent une relation de Pascal. On peut donc exploiter la fonction **triangle\_pascal()** pour calculer les k-combinaisons.

On a le code suivant :

```
kcombinaisons_pascal <- function(k, n){  
  condition = k <= n && k != 0 && n!=0  
  if(condition){  
    ligne_n_pascal = triangle_pascal(n)  
    nb_kcombinaisons = ligne_n_pascal[k+1]  
    return(nb_kcombinaisons)  
  }  
  else{  
    return(FALSE)  
  }  
}
```

`nb_kcombinaisons` contient l'élément  $k+1$  ( $k+1$  car la première valeur de  $k$  dans le triangle est de 0) de la colonne  $n$  du triangle de Pascal. Donc la variable contient  $C(k, n)$ .

IN:

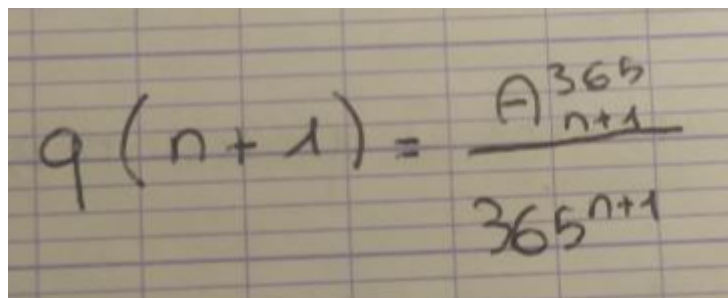
`kcombinaisons_pascal(997, 1000)`

OUT:

$k=997$   
166167000

*Question bonus :*

Il est demandé de programmer un script R qui exploite la relation de récurrence suivante pour trouver tel que  $q(n)$  franchit 50% (0,5):


$$q(n+1) = \frac{A 365^{n+1}}{365^{n+1}}$$

On a donc le script suivant :

```
n = 1
q = 1
while(q >= 0.5){
  n = n+1
  q <- karrangements(n, 365) / pow(365, n)
}
print(n)
```

On initialise  $n$  à 1 car il est incrémenté au début de la boucle **while()** et on initialise  $q$  à sa valeur maximale pour forcer l'entrée dans la boucle.

On ouvre une boucle **while()**, on reste dedans tant que  $q \geq 0,5$  (et non pas  $>$  car on veut savoir quand  $q(n)$  FRANCHIT  $0,5$ ), à l'intérieur on incrémente  $n$  de 1 (relation de récurrence) puis on calcule  $q(n)$ . Enfin, on affiche à l'écran  $n$ .

**OUT:**

*[1] 23*

On obtient bien le résultat attendu.