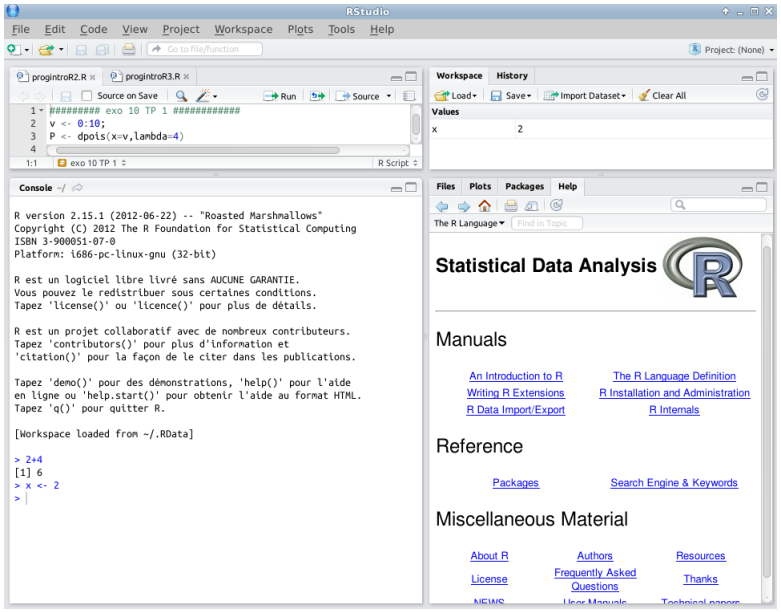
Introduction au logiciel R

R est une implémentation libre du langage S dédiée à l'analyse des données. Il peut être téléchargé gratuitement sur le site du CRAN (Comprehensive R Archive Network, http ://cran.r-project.org/).

– R est un langage orienté-objet : les variables, les données, les fonctions, les résultats de calculs ... sont stockés dans la mémoire de l'ordinateur sous forme d'objets qui ont chacun un nom.
– R est un langage interprété : les lignes de commande sont exécutées au fur et à mesure.

Démarrage de R sous Linux

On dispose d'une interface graphique RStudio[1] pour travailler avec R. Pour lancer cette interface, cliquer le menu des applications (en haut à gauche de l'écran) pour aller sur l'onglet *Développement* et cliquer sur RStudio. R est lancé en même temps que RStudio.
L'interface de RStudio se présente sous la forme d'une unique fenêtre découpée en quatre zones que l'on peut redimensionner :



– En haut à gauche, fenêtre pour l'édition de fichiers. Utiliser le menu *File* pour ouvrir un fichier ;
– En bas à gauche se trouve la console R où on tape les commandes R.
– En haut à droite, on peut basculer entre la fenêtre Workspace qui affiche la liste des objets de l'espace de travail actuel et la fenêtre History qui affiche l'historique des commandes exécutées dans la console R ;
– En bas à droite, on peut faire afficher :
  – un navigateur de fichiers ;
  – la fenêtre d'affichage des graphiques ;
  – une liste des extensions installées ;
  – la fenêtre d'aide.

NB : On peut aussi lancer R directement dans un terminal, en tapant R. On quitte le logiciel en tapant q()
NB : Pour sauvegarder l'espace de travail et l'historique avant de quitter RStudio, utiliser le bouton *Save* dans les onglets Workspace et History.

Le prompt de R Lorsque le caractère > s'affiche en début de ligne, cela signifie que R attend une instruction. Si le caractère + apparaît au lieu de >, cela signifie que la commande tapée n'est pas complète, R attend qu'elle soit complétée pour l'exécuter.

Exemple :
```
 > 2 + 4
```
R affiche le résultat et attend la prochaine commande :

```
[1] 6
>
```

Par contre, si on voulait taper 3*4 et que l'on a écrit 3* puis tapé sur la touche Entrée, on obtient :

```
 > 3*
 +
```

Il faut compléter en écrivant 4, puis en tapant sur la touche Entrée :

```
 > 3*
 + 4
[1] 12
 >
```

NB : on verra plus loin la signification du [1] dans l'affichage de la réponse.

[1] RStudio fonctionne aussi bien sur Windows, Mac OS X que sur Linux, il est téléchargeable sur le site http://www.rstudio.org/

Les objets

Un objet est désigné par son nom. Son contenu est caractérisé par :
- **le mode**, c'est-à-dire le type de données de l'objet ; il y en quatre principaux : numérique, caractère, complexe et logique.
- **la longueur**, c'est-à-dire le nombre d'éléments de l'objet.

Exercice 1. Exécuter les lignes suivantes (<- s'obtient avec la touche '<' suivi de la touche '-' (signe moins)) :

```
a <- 1+3
a
b <- 2*a
```

Explications :
- La 1ère ligne définit un objet appelé `a` de type numérique de longueur 1.
- Lorsqu'on tape le nom d'un objet, le contenu de cet objet s'affiche.
- La « flêche » `<-` permet d'affecter un contenu à une variable.

Question : Faire afficher le contenu de la variable `b`. Quelle est sa valeur ?

Exercice 2. Exécuter les lignes suivantes :

```
v <- c(2,-1,0,3) ; v
v[2]
w <- 2:40 ; w
m <- c('longueur','hauteur','largeur') ; m
ls.str()
```

Explications :
- le point-virgule permet de séparer des instructions écrites sur la même ligne ;
- `v` est une liste de 4 nombres ; on dit que c'est un vecteur numérique de longueur 4. Pour $i \in \{1, 2, 3, 4\}$, `v[i]` désigne le i-ème élément du vecteur `v`.
- `w` est un vecteur numérique de taille 39. Si l'affichage des éléments de w prend plusieurs lignes, au début de chaque ligne l'indice du premier élément de la ligne est indiqué entre `[]`. C'est pourquoi `[1]` est affiché au-début de la 1ère ligne.
- `m` est un vecteur de longueur 3 contenant des chaînes de caractères (une chaîne de caractères est entrée entre des quotes ′ ′ ou des doubles quotes ″ ″),
- `ls.str()` est une fonction qui donne la liste des variables créées.
  Attention : le nom des variables peut contenir des lettres et des chiffres mais ne peut pas commencer par un chiffre et ne doit pas contenir d'espaces, ni d'accents.

Exercice 3. Quel est le 20-ième élément de `w` ?
Taper les commandes `w[1 :5]` et `w[c(1,4,6)]`. Que font-elles ?

Exercice 4. On a vu que la commande `c` permet de créer un vecteur en entrant une liste d'éléments. Les instructions suivantes donnent d'autres méthodes pour créer des vecteurs, les exécuter pour voir ce qu'elles font :

```
3:10
rep(x=1, times=10)
rep(x=1:2, times=4)
rep(x=1:2, each=4)
seq(from=-0.2, to=0.5, by=0.1)
seq(from=0, to=1, length=10)
z <- numeric(5)
z
```

Exercice 5. Créer un vecteur `x` contenant 100 valeurs régulièrement espacées entre -3 et 3.

Les opérateurs

L'utilisateur agit sur les objets avec des opérateurs
- arithmétiques ($+, -, *, /, \hat{}$)
- logique (&, |, `xor` correspondant respectivement aux opérateurs logiques “et”, “ou” et “ou exclusif”)
- de comparaison ($==, !=, >, <, <=, >=$)

et avec des fonctions (qui sont elles-même des objets).

Les opérations sur les vecteurs

Rappelons qu'un vecteur est une liste de nombres numérotés de 1 à la longueur du vecteur.

Ajout d'un nombre `a` à tous les éléments d'un vecteur `v`	`a+v`
Multiplication de tous les éléments d'un vecteur v par un nombre `a`	`a*v`
Addition de 2 vecteurs `v` et `w` de même taille	`v+w`
Multiplication coefficient par coefficient de 2 vecteurs `v` et `w` de même taille	`v*w` (son i-ème coefficient est `v[i]w[i]` pour tout `i`).

NB : La taille d'un vecteur `v` s'obtient en tapant `length(v)`.

Exercice 6. Exécuter les instructions suivantes

```
v <- c(2,-1,0,3)
v-4
2*v
v == 3
L <- v==3
1 - L
2*L
v[v < 1]
```

Explications : on voit, sur les commandes précédentes, qu'avec `R` on peut appliquer une opération à tous les éléments d'un vecteur. Par exemple, `v == 3` applique l'opérateur `==` à tous les élements du vecteur et retourne un vecteur « logique ». On peut effectuer des opérations sur les vecteurs logiques : TRUE a alors la valeur 1 et FALSE a la valeur 0. Par exemple, comme `L` contient le résultat de l'opération `v == 3`, `1-L` sera le même vecteur que `v != 3`.

Les fonctions

Une fonction dont le nom est *nomf*, est exécutée en tapant :
- `nomf()`, si elle n'a pas d'arguments
  Exemple : On a déjà rencontré les commandes `q()` pour quitter `R` et `ls.tr()` pour obtenir la liste des variables en mémoire.
- `nomf`(`narg1`=v1, `narg2`=v2, ...), si elle a besoin des arguments dont les noms sont `narg1, narg2`,...
  Exemple : On a déjà utilisé la fonction `seq` par exemple avec la commande `seq(from=-0.2, to=0.5, by=0.1)` : ici les noms des arguments utilisés sont `from`, `to` et `by` ; cette commande crée le vecteur (-0.2 -0.1 0.0 0.1 0.2 0.3 0.4 0.5) ;

Avec cette syntaxe, l'ordre dans lequel on écrit les arguments de la fonction n'a pas d'importance, puisqu'un nom est attaché à chaque argument. On pourrait aussi simplement écrire `nomf(v1, v2, ...)`. Mais là il faut respecter l'ordre des arguments.
Exemple : On peut écrire `seq(-0.2,0.5,0.1)` au lieu de `seq(from=-0.2, to=0.5, by=0.1)`, mais c'est beaucoup moins parlant.

`R` contient beaucoup de fonctions qui peuvent s'appliquer à chaque élément d'un vecteur. C'est le cas de toutes les fonctions usuelles `sin`, `cos`, `exp` ...

Exercice 7.
Exécuter les trois instructions ci-contre :
1. Que font les fonctions `sum` et `mean` ?
   indication : penser à la signification de ces mots en anglais.
2. On considère une variable aléatoire X de loi uniforme sur les entiers de -10 à 20. Utiliser les commandes de `R` déjà rencontrées pour faire calculer à `R` l'espérance de X et la variance de X.
   Indication : commencer par définir deux variables `val` et `p` qui contiendront respectivement les valeurs possibles pour X et les coefficients de la loi de X de sorte que : `val[1] = -10, val[2] = -9` etc. et `p[1] = P(X=-10), p[2] = P(X=-9)` etc. (x^2 s'obtient en tapant `x^2`. Le caractère `^` s'obtient en tapant en même temps sur les touches AltGr et 9).

```
sum(v)
mean(v)
sum( v < 1 )
```

Exercice 8. Exécuter les instructions suivantes :

```
t <- seq(from = 0, to = 2*pi, by = 0.1)
f <- sin(t)
plot(y = f, x = t)
plot(t,f,main = 'fonction sinus')
plot(f,t)
g <- f/cos(t)
plot(y = g, x = t)
```

Quelles sont les fonctions utilisées ?
Quel tracé a-t-on obtenu avec la commande `plot(f,t)` ?
Comment fonctionne / entre 2 vecteurs ?

NB : `sum`, `mean`, `seq`, `sin` et `plot` sont des fonctions.

Exercice 9. Tracer 100 points de la fonction $x \mapsto \exp(-x^2)$ en des valeurs réparties régulièrement entre -3 et 3.

L’aide

Pour obtenir l’aide sur une fonction :
– si on connait le nom de la fonction par exemple `nomf`, on peut taper la commande `?nomf` ou `help(nomf)`. La fenêtre d’aide s’ouvre dans la fenêtre *Help* en bas à droite. On peut aussi taper directement `nomf` dans le champ de recherche en haut à droite de la fenêtre *Help* de `RStudio`.
– si on ne connait pas le nom de la fonction, on peut faire une recherche par mot-clé en utilisant le lien “Search Engine & Keywords” dans la page d’accueil de l’aide.

Pensez que l’aide en ligne est en anglais, il faut donc entrer des mots anglais comme mots-clés.

Exercice 10.

1. Taper `help(plot)` afin de lire l’aide de `plot`. Modifier la commande `plot(t,f)` pour que les points soient reliés par des segments de droite.
2. Chercher les commandes permettant de calculer $n! = n(n-1)(n-2)...1$ et $\binom{n}{k}$.

NB : La liste des variables qui ont été créée apparaît dans la fenêtre *Workspace* en haut à droite. Il faut penser à effacer régulièrement les variables inutiles pour ne pas encombrer la mémoire disponible :
– la commande `rm(a)` supprime la variable `a`
– `rm(list = ls())` supprime toutes les variables créées.

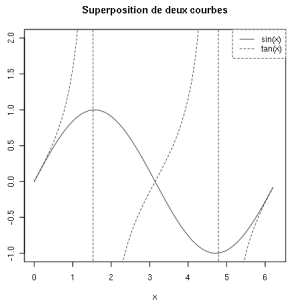
La gestion des graphiques

Quelques commandes utiles pour faires des graphiques :

`plot(x=...,y=...,type=...,` `col=..., xlim=...,` `ylim=..., main=...,` `xlab=..., ylab=...)`	trace les points dont les abscisses sont données dans le vecteur x et les ordonnées dans le vecteur y, `type` permet de définir le type de tracé, `col` la couleur, `xlim` et `ylim` permettent de préciser les limites des axes, `main` le titre, `xlab` et `ylab` les légendes pour les axes.
`points(x=...,y=...,type=...)`	pour ajouter des points sur un tracé
`par(new=TRUE)`	pour superposer des graphiques (par défaut, new=FALSE)
`par(ask=TRUE)`	l’exécution du programme s’arrête jusqu’à ce que l’utilisateur tape sur la touche Enter
`par(mfrow=c(lig,col))`	divise la fenêtre graphique permettant d’afficher `col` graphiques par ligne sur `lig` lignes
`par(lty=n)`	controle le type de lignes tracées ; par exemple : n=1 pour une ligne continue, n=2 pour des tirets, n=3 pour des pointillés
`par(pch=''*'')`	pour que le symbole utilisé pour tracer des points soit *
`legend(x=..., y=...,` `legend=...)`	pour écrire des légendes lorsqu’on superpose des graphiques ; `x` et `y` permettent de préciser la position des légendes.
`title(main=..., sub=...,` `xlab=..., ylab=...)`	pour ajouter des titres à une figure

Exemple : Pour tracer la figure ci-contre, on peut utiliser les commandes suivantes

```
par(lty=1)
plot(x=t,y=f,type='l',xlim=c(0,2*pi),ylim=c(-1,2),xlab='x',ylab=' ')
par(new=TRUE,lty=2)
plot(x=t,y=g,type='l',xlim=c(0,2*pi),ylim=c(-1,2),xlab=' ',ylab=' ')
legend(x='topright',legend=c('sin(x)','tan(x)'),lty=c(1,2))
title(main='Superposition de deux courbes')
```



ou de façon équivalente :

```
par(lty=1)
plot(x=t,y=f,type='l',xlim=c(0,2*pi),ylim=c(-1,2),xlab='x',ylab=' ')
par(lty=2)
points(x=t,y=g,type='l')
legend(x='topright',legend=c('sin(x)','tan(x)'),lty=c(1,2))
title(main='Superposition de deux courbes')
```

Exercice 11.

1. Créer un sous-répertoire *progR* dans votre répertoire *Documents* en utilisant votre gestionnaire de fichiers (le gestionnaire de fichiers s’obtient par exemple en cliquant sur l’icône représentant un répertoire).
2. Ouvrir le fichier `ex_trace.R` qui se trouve dans le répertoire /home/doc/lemaire/L3CM12/ en utilisant le menu *File* de `RStudio`. Sauvegarder ce fichier dans votre répertoire `progR` toujours en utilisant le menu *File*.

3. Cliquer sur le bouton *Source* pour exécuter les commandes qui se trouvent dans ce fichier. Regarder la commande qui a été exécutée dans la console de `R`. Puis, regarder la figure qui a été créée (cliquer sur l'onglet *Plots* dans la fenêtre en bas à gauche si la figure n'est pas visible).
4. Rajouter des commandes pour afficher aussi la courbe du cosinus entre 0 et 2π en pointillés.
5. Exécuter votre fichier en utilisant le bouton *Source on save* (ce bouton sauvegarde le fichier et l'exécute).
6. Utiliser le menu *Export* en haut de la figure pour sauvegarder la figure obtenue (sélectionner le format pdf et choisir le répertoire où le fichier sera enregistré).

Travailler avec les lois classiques

Il existe 4 fonctions permettant de manipuler chaque loi classique. Nous donnons en exemple celles associées à une loi binomiale dont les paramètres sont à préciser dans les objets `size` et `prob`

`dbinom(x,size,prob)`	pour calculer les coefficients de la loi aux valeurs à spécifier dans `x`
`pbinom(q, size, prob)`	pour calculer la fonction de répartition de la loi aux valeurs spécifiées dans `q`
`qbinom(p, size, prob)`	pour calculer les quantiles de la loi dont les ordres sont spécifiés dans `p`
`rbinom(n, size, prob)`	pour simuler un `n` échantillon de v.a. ayant cette loi

Ces quatre types de fonctions existent pour différentes lois classiques, la syntaxe est toujours la même d,p,q ou r suivi du nom abrégé de la loi (`pois` pour les lois de Poisson, `geom` pour les lois géométriques sur $\mathbb{N}$, `nbinom` pour la loi binomiale négative, `hyper` pour les lois hypergéométriques ...). Attention, le nom des variables `size`, `prob` dépend de la loi. Regarder l'aide pour utiliser ces fonctions.

Exemple :

`pbinom(q=3.5,size=10,p=0.5)` donne la valeur de la fonction de répartition de la loi binomiale $\mathcal{B}(10, 0.5)$ au point 3.5.

`dbinom(x=0:3,size=10,p=0.5)` retourne un vecteur de taille 4 contenant dans l'ordre les probabilités $P(X = 0), P(X = 1), P(X = 2), P(X = 3)$ où X désigne une v.a. de loi binomiale $\mathcal{B}(10, 0.5)$.

`rbinom(n=100,size=10,p=0.5)` simule 100 réalisations d'une v.a. de loi binomiale $\mathcal{B}(10, 0.5)$.

Exercice 12. Déterminer la probabilité qu'une variable aléatoire X de loi géométrique sur $\mathbb{N}$ de paramètre 0.1 soit supérieure ou égale à 4 (X peut représenter le nombre de fois où une pièce tombe sur 'face' avant de tomber pour la 1ère fois sur 'pile' si on suppose que la pièce a une probabilité 0.1 de tomber sur 'pile' à chaque lancer).

Exercice 13. Cinq personnes choisissent au hasard un entier entre 1 et 3. Utiliser `R` pour calculer la probabilité que 3 des 5 chiffres soient identiques.

Une société de livraison emploie trois coursiers. En un mois, il y a eu au total cinq incidents, dont trois avec le même coursier.

Quelle est a priori la probabilité d'une telle répartition des incidents ?

Ecrire des programmes

Lorsqu'on veut exécuter une série d'instructions, il est souvent plus pratique d'écrire les lignes de commande dans un fichier en utilisant un éditeur de texte. Il faut donner à ce fichier un nom suivi de l'extension `.R`. Le fichier `ex_trace.R` est un exemple de programme `R`.

Avec l'interface graphique `RStudio`, on peut créer un nouveau fichier ou en ouvrir un déjà créé en utilisant le menu *File*. On peut faire exécuter un bloc de commandes en sélectionnant ce bloc de commandes avec la souris et en cliquant sur le bouton *Run*. On peut exécuter un fichier complet en utilisant le bouton *Source on save*.

NB : Si on a lancé `R` depuis un terminal, pour exécuter les commandes tapées dans un fichier appelé *nomp.R*, on tape dans le terminal `source`('*adresse/nomp.R*') où *adresse* doit être remplacé par l'adresse du répertoire contenant le fichier nomp.R. Par exemple, l'adresse du répertoire progR est $\sim$*/Documents/progR*.

Exercice 14.

1. Ouvrir un nouveau fichier dans l'éditeur de `RStudio` (menu *File*, puis *New*).
   Taper les commandes ci-contre dans ce fichier. Le sauvegarder dans le répertoire `progR` en lui donnant un nom suivi de l'extension `.R` (le nom de fichier ne doit contenir que des lettres, ne doit contenir ni accent, ni espace). Vous venez de créer un programme. Exécuter ce programme en cliquant sur le bouton *Source on Save*.
   Que représentent les figures obtenues ?

```
v <- 0:10;
P <- dpois(x=v,lambda=4)
par(mfrow=c(2,1))
plot(v,P,type='h')
F <- ppois(q=v,lambda=4)
plot(v,F,type='s')
```

2. Exécuter les deux commandes suivantes : `P[4 :6]` et `F[6]`.

3. Dire que représentent les différentes valeurs obtenues.
4. Créer un autre programme qui trace sur une même figure, le diagramme en bâtons de la loi binomiale $\mathcal{B}(100, 0.04)$ et le diagramme en bâtons de la loi de Poisson de paramètre 4 sur l'intervalle [0,10] (on décalera les abscisses du second diagramme de 0.1 pour mieux voir) et qui calcule l'écart maximal qu'il y a entre les coefficients de ces deux lois. Qu'observez-vous ?
   Indication : on pourra utiliser les fonctions `max` et `abs` pour calculer l'écart maximal.

Exercice 15.

1. Regarder l'aide des fonctions `dhyper` et `phyper`.
2. On fait un sondage parmi les $T = 500$ personnes votant à un bureau de vote donnée avant un référendum. On note S le nombre de personnes pour le « oui » parmi les $A = 31$ personnes différentes interrogées. Quelle est la loi de S ?
3. Ecrire un programme qui trace le diagramme en bâtons de la loi de S :
   a) si parmi ces 500 personnes, 200 personnes sont pour le « oui ».
   b) si parmi ces 500 personnes, 250 personnes sont pour le « oui ».
4. Compléter le programme pour qu'il calcule dans les deux cas :
   – l'espérance de S dans les deux cas,
   – la probabilité d'avoir au moins 16 personnes qui sont pour le « oui » parmi les personnes sondées.
5. On note p la proportion de personnes qui sont pour le « oui » dans la population. Le journaliste qui rapporte le résultat du sondage déclarera que le « oui » va l'emporter si S est au moins égale à 16 et que le « non » va l'emporter si S est inférieure ou égale à 15. On note $g(p)$ la probabilité qu'il se trompe (c'est-à-dire que le résultat du vote des 500 personnes soit différent de ce qu'il aura déclaré).
   Compléter le programme pour qu'il calcule $g(0.45)$ et $g(0.48)$.
6. Compléter le programme afin qu'il trace la valeur de $g(p)$ en fonction de p pour p compris entre 0.3 et 0.7.
   Indication : on fera déjà un tracé de $g(p)$ pour $p \in [0.3, 0.5[$. Puis on superposera le tracé de $g(p)$ pour $p \in]0.5, 0.7]$.

Exercice 16. On constitue une séquence de 40 nucléotides de gauche à droite en tirant au hasard chaque nucléotide indépendamment les uns des autres suivant la loi :

a	c	g	t
0.3	0.2	0.4	0.1

On note X la variable aléatoire donnant le nombre de nucléotides `t` dans une telle séquence.

1. Utiliser les fonctions de `R` pour calculer la probabilité :
   – qu'il y ait au plus 2 nucléotides `t` dans une telle séquence,
   – qu'il y ait au moins 8 nucléotides `t` dans une telle séquence.
2. Utiliser la fonction dans `R` qui détermine les coefficients de la loi de X pour calculer l'espérance de X, son moment d'ordre 2 puis sa variance.
3. On note Y la variable aléatoire qui donne le nombre de dinucléotides `ag` si on voit cette séquence comme une suite de 20 dinucléotides. Déterminer sa loi et représenter le diagramme en bâtons de sa loi. Combien de dinucléotides `ag` a-t-on le plus de chance d'observer dans une telle séquence ?
4. On note Z le numéro de la première base qui porte le nucléotide `c` (on posera $Z = 41$ si la séquence n'a pas de nucléotides `c`). Déterminer la loi de Z. Ecrire un programme qui représente sa loi, calcule son espérance et son écart-type.
5. On note W la seconde base qui porte le nucléotide `c` dans la séquence (on posera $W = 41$ si la séquence n'a pas de nucléotides `c` ou si elle n'en a qu'un).
   (a) Décrire l'événement $A_{k,\ell} = \{Z = k \text{ et } W = \ell\}$ pour $1 \leq k \leq 40$ et $\ell = 41$ puis pour $1 \leq k < \ell \leq 40$.
   (b) En déduire la probabilité que l'événement $A_{k,\ell}$ se réalise en fonction de k et ℓ.
   (c) Déduire du calcul précédent la loi de W.
   (d) Compléter le programme précédent afin qu'il représente aussi la loi de W en dessous du graphique de la loi de Z et calcule l'espérance de W et l'écart-type de W.
   (e) Que représente la variable aléatoire $W - Z$? Quelle est sa loi ?

Simulation de tirages

La fonction `sample(x,size,replace,prob)` permet de simuler des tirages, sans remise ou avec remise, d'objets parmi une liste d'objets, en affectant à chaque objet une certaine probabilité d'être tiré.
Description des arguments de `sample` :
– `x` : vecteur contenant les différents objets que l'on peut tirer,
– `size` : nombre de tirages à effectuer,
– `replace=TRUE` pour un tirage avec remise, `replace=FALSE` pour un tirage sans remise,
– `prob` : vecteur définissant les probabilités de tirer chaque objet défini dans `x`. Il doit être de même taille que `x`.
La fonction `sample(x,size,replace,prob)` retourne alors un vecteur ayant `size` éléments.
Dans un tirage avec remise, le 1er élément de `prob` contient le probabilité de tirer le 1er objet de `x`, le 2ème élément de `prob` contient la probabilité de tirer le 2ème objet de `x` ....
Dans un tirage sans remise, les objets tirés sont exclus des tirages suivants et les probabilités associées aux objets restants sont proportionnelles aux valeurs mises dans le vecteur `prob`. Par exemple, si au premier tirage, on a tiré l'objet `x[2]`, alors au second tirage les objets `x[i]` pour $i \neq 2$ seront tirés avec une probabilité égale à $\frac{\texttt{prob}[i]}{\sum_{j\neq 2}\texttt{prob}[j]}$.

Exemple :

```
sample(x=c('pile','face'),size = 1000,replace = TRUE, prob = c(0.4,0.6))
```

permet de simuler 1000 lancers d'une pièce qui a une probabilité 0.4 de tomber sur pile à chaque lancer.

Exercice 17. Quelle expérience aléatoire permet de simuler les instructions suivantes ?

```
Lobj <- c(rep(x=1, times=30),rep(x=0, times=20))
sample(x = Lobj, size = 10, replace = FALSE, prob = rep(x=1/50, times=50))
```

Exercice 18. (*suite de l'exercice 16*) On note respectivement X_a, X_c, X_g et X_t le nombre de nucléotides `a`, `c`, `g` et `t` dans une séquence d'ADN de longueur 40 construite d'après le modèle défini à l'exercice 16.

1. Ecrire un programme appelé `etudADN.R` qui
   (a) simule la constitution d'une séquence de 40 nucléotides suivant le modèle décrit à l'exercice 16 et met le résultat dans une variable appelée `ADN` ;
   (b) affiche le contenu de `ADN` ;
   (c) détermine les réalisations obtenues pour les variables aléatoires X_a, X_c, X_g et X_t que l'on notera `xa`, `xc`, `xg` et `xt`.
2. Lancer ce programme et noter les valeurs de `xa`, `xc`, `xg` et `xt`.
3. Si vous lanciez une seconde fois ce programme, quelle serait la probabilité d'obtenir une valeur pour `xa` identique à celle que vous avez obtenue la première fois ? quelle serait la probabilité d'obtenir pour `xt` une valeur supérieure ou égale à celle obtenue la première fois ?

Répéter une série d'instructions

La boucle `for` est utilisée pour répéter une suite d'instructions un nombre prédéfini de fois. La syntaxe est la suivante :
`for` (*compteur* `in` *vecteur*){
....
.... #liste d'instructions à exécuter
....}
La liste d'intructions apparaissant entre les deux accolades est exécutée autant de fois qu'il y a d'éléments dans *vecteur* ; *compteur* est une variable qui prendra successivement chaque élément présent dans *vecteur*.

Exemple : Les lignes suivantes calculent la somme des 4 premiers entiers et 4! :

```
p <- 1; s <- 0
for(i in 1:4){
  s <- s + i
  p <- p*i
}
```

Détails du fonctionnement du programme :
- Avant le premier passage dans la boucle `for`, `s` a la valeur 0.
- Au premier passage dans la boucle `for`, les deux commandes suivantes
  ```
  s <- s + i
  p <- p*i
  ```
  sont exécutées avec `i`= 1. Les valeurs de `s` et `p` sont modifiées, `s` vaut 1 et p vaut 1.
- Ces deux instructions sont ensuite réexécutées avec `i`= 2. Donc après le second passage dans la boucle `for` on a `s`= $1+2=3$ et `p`= $1*2=2$.
- Au troisième passage dans la boucle `for`, les deux instructions sont éxécutées avec `i`= 3 et donc les valeurs de `s` et `p` seront modifiées en `s`= $3+3=6$ et `p`= $2*3=6$.
- Ensuite les deux instructions seront éxécutées avec `i`=4. Le programme s'arrête alors.

A la fin du programme, la valeur de `s` sera : $6+4=10$ et la valeur de `p` sera : $1*1*2*3*4=24$.

NB : Dans beaucoup de cas on peut éviter d'utiliser des boucles en utilisant que de nombreuses fonctions de **R** sont faites pour travailler sur des vecteurs. Par exemple, le programme précédent fait la même chose que les deux lignes suivantes :

```
s <- sum(1:4)
p <- prod(1:4)
```

Exemple : détaillons le fonctionnement du programme suivant :

```
x <- c(2,4,10,-3)
v <- numeric(length=4)
v[1] <- x[1]
for(i in 2:4){ v[i] <- v[i-1]+x[i] }
```

- avant d'exécuter la boucle `for`, `v` est le vecteur (2,0,0,0)
- au 1er passage dans la boucle `for`, `i`= 2, le 2ème coefficient de `v` est modifié et donc `v` devient le vecteur (2,6,0,0).
- au 2ème passage dans la boucle `for`, `i`= 3 le 3ème coefficient de `v` est modifié et donc `v` devient le vecteur (2,6,16,0).
- au 3ème passage dans la boucle `for`, `i`= 4, le 4ème coefficient de `v` est modifié.

Le programme s'arrête ensuite ; `v` est donc à la fin le vecteur (2,6,16,13).

Exercice 19. Dans les trois programmes suivants `n` est un entier supérieur à 3 fixé. Expliquer ce que font les trois programmes suivants et répondre aux questions. Vous trouverez dans le fichier `ex_boucle1.R` les trois programmes avec des valeurs fixées pour `n` vous permettant d'exécuter ces programmes en faisant varier la valeur de `n` et en faisant afficher les valeurs des variables créées par chacun des programmes.

programme 1

```
v <- seq(from=2, to=n, by=2)
w <- numeric(n)
for (i in v){
  w[i-1] <- i-1
  w[i] <- -i
}
```

On se donne un entier j compris entre 1 et n. Quelle est la valeur du j-ème coefficient de `w` à la fin du programme 1 ?

programme 2

```
k <- 3
s <- 0
x <- 2
for (i in k:n){
  x <- 2*x
  s <- s + x
}
```

Donner une expression de la valeur de `x` et de `s` en fonction de `n` lorsque le programme 2 est terminé.

programme 3

```
a <- 0.4
b <- 0.6
f <- 0
for (i in 1:n){
  for (j in 1:i){
    f <- f + dbinom(i,n,a)*dbinom(j,n,b)
  }
}
```

On note U et V des variables aléatoires de lois binomiales $\mathcal{B}(n,a)$ et $\mathcal{B}(n,b)$ respectivement. La valeur de `f` à la fin du programme 3 permet de calculer la probabilité d'un évenement dépendant de U et V. Lequel ?

Exercice 20. On dispose de 20 rats. On les répartit au hasard en deux groupes de 10 rats notés A et B. On ajoute à la nourriture des rats du groupe A un produit M. Le groupe B constitue un groupe de contrôle. Trois rats parmi le groupe B développent une tumeur au cours de la 1ère année alors que c'est le cas pour 5 rats du groupe A. Que peut-on en conclure ? Pour essayer de répondre à cette question, on note p_c la probabilité qu'un rat développe une tumeur au cours de la 1ère année et p_m la probabilité qu'un rat nourri avec un régime contenant le produit M développe un tumeur au cours de la première année. Enfin, on note X le nombre de rats qui développent une tumeur cancereuse au cours de la première année sur 10 rats nourris normalement et Y le nombre de rats qui au cours de la 1ère année développe un tumeur cancereuse parmi 10 rats nourris avec une alimentation contenant le produit M.

1. Quelles lois suivent X et Y ? Les variables aléatoires X et Y sont-elles indépendantes ? (on précisera les hypothèses faites)
2. Donner une expression de la probabilité qu'il y ait au moins deux rats de plus développant une tumeur dans le groupe A que dans le groupe B en fonction des coefficients des lois de X et de Y.
3. Ecrire un programme qui calcule cette probabilité si $p = p_c = p_m$ (on introduira une variable p). Lancer ce programme pour différentes valeurs de p ($p = 0.1$, $p = 0.2$, $p = 0.3$, $p = 0.5$, $p = 0.6$) et noter les valeurs obtenues.
4. Que peut-on conclure des résultats de l'expérience menée sur les rats ?

Exercice 21. Le fichier `ex_traceboucle.R` contient le programme suivant. Exécuter-le deux fois et regarder les figures obtenues. Analyser ce que fait ce programme en détaillant ce que contiennent les vecteurs `u`, `v` et `m` à la fin du programme.

```
n <- 20
p <- 0.3
u <- sample(x=c(-1,1), replace=TRUE, size=n, prob=c(p,1-p))
v <- numeric(n)
m <- numeric(n)
v[1] <- u[1]
m[1] <- v[1]
for (i in 2:n){
  v[i] <- v[i-1] + u[i]
  m[i] <- v[i]/i
}
par(mfrow=c(2,1));par(lty=1);par(pch='.')
plot(x=1:n, y=v, type='b')
plot(x=1:n, y=m, type='b')
```

Compléter le programme pour mettre des titres aux deux tracés.
Changer le programme pour donner à `n` la valeur de 5000 et relancer-le deux fois. Changer le programme pour donner à `p` la valeur de 0.6 et relancer-le deux fois. Qu'observez-vous de commun ? de différent sur les figures entre ces 4 exécutions ?

Exercice 22. Le fichier `ex_traceboucle2.R` contient le programme suivant.

```
n <- 1000
p <- 0.5
q <- 0.5
u1 <- sample(x=c(-1,1), replace=TRUE, size=n, prob=c(p,1-p))
u2 <- sample(x=c(0,1), replace=TRUE, size=n, prob=c(q,1-q))
X <- numeric(n)
Y <- numeric(n)
D <- numeric(n)
k <- 1
m <- D[1]
for (i in 1:n){
  V1 <- (1-u2[i])*u1[i]
  V2 <- u2[i]*u1[i]
  X[i+1] <- X[i] + V1
  Y[i+1] <- Y[i] + V2
  D[i+1] <- sqrt(X[i]^2+Y[i]^2)
  r <- max(c(m,D[i+1]))
  k <- k + (r > m)
  m <- r
}
print(k)
par(mfrow=c(1,1))
plot(x=X, y=Y,type='l')
points(x=c(X[1],X[n+1]),y=c(Y[1],Y[n+1]), col='red')
readline("courbe suivante") #taper sur 'Entree' pour continuer
par(mfrow=c(2,2))
plot(x=X, y=Y,type='l')
points(x=c(X[1],X[n+1]),y=c(Y[1],Y[n+1]), col='red')
plot(x=1:(n+1),D,type='l')
```

Pour tout j, le j-ème élément du vecteur `u1` peut être considéré comme une réalisation d'une variable aléatoire $U_1^{(j)}$. Les variables aléatoires $U_1^{(1)}, U_1^{(2)}, \ldots$ sont toutes de même loi et indépendantes entre elles. De même, pour tout j le j-ème élément du vecteur `u2` peut être considéré comme une réalisation d'une variable aléatoire $U_2^{(j)}$. Les variables aléatoires $U_2^{(1)}, U_2^{(2)}, \ldots$ sont toutes de même loi, indépendantes entre elles et indépendantes des variables aléatoires $U_1^{(1)}$, $U_1^{(2)}$, ...

1. Déterminer la loi du vecteur (Z, W) avec $Z = (1 - U_2^{(1)})U_1^{(1)}$ et $W = U_2^{(1)}U_1^{(1)}$.
2. Analyser ce que fait ce programme en précisant ce que représentent les élements des vecteurs `X`, `Y` et `D`, puis ce que représente la valeur de `k` à la fin du programme.
3. Exécuter le programme avec deux valeurs différentes de `p`, noter les valeurs de `k` obtenues à chaque fois.
4. En prenant `n=1000`, exécuter de nouveau ce programme 2 fois le programme pour `p=0.5`, 2 fois pour `p=0.45` puis pour `p=0.6`. Noter les valeurs de `k` obtenues à chaque fois.
5. Commenter les résultats et courbes obtenues (qu'observez-vous de différents ? de commun ? lorsque vous relancez le programme pour une même valeur de `p` ? pour une valeur différente de `p` ?)

Ecrire des fonctions

On peut écrire ses propres fonctions qui se rajouteront aux fonctions déjà présentes dans `R`. La syntaxe est la suivante :
nomf <− `function(arg1,arg2,...){`
...
... #lignes d'instructions
....}
La fonction retournera par défaut le résultat de la dernière expression évaluée. On peut préciser le résultat renvoyé ou retourner plusieurs résultats en constituant une liste grâce à la commande `return`(*liste*) où *liste* désigne la liste des variables que la fonction doit retourner.

Exemple :

```
esperance <- function(val,prob){
# fonction qui retourne l'esperance d'une v.a X dont
# la loi est decrite par les deux vecteurs val et prob:
```

```
# P(X=val[i])=prob[i]) pour tout i
e <- sum(val*prob)
return(e)
}
```

La commande `res <- esperance(val=c(-1,1,3), prob=c(0.3,0.1,0.6))` crée un objet `res` qui contient l'espérance d'un variable aléatoire X à valeurs dans $\{-1,1,3\}$ et telle que $P(X=-1)=0.3$, $P(X=1)=0.1$ et $P(X=3)=0.6$.
NB : Le caractère `#` sert à marquer le début d'un commentaire ; le reste de la ligne n'est pas interprété par `R`.

Exemple : La fonction suivante retourne la moyenne et la variance empirique des données (si x est un vecteur de taille n contenant des données, la variance empirique de x est $\frac{1}{n}\sum_{i=1}^{n}(x[i]-\texttt{mean}(x))^2$).

```
statdescript <- function(data){
# fonction qui retourne la moyenne et la variance empirique
# des valeurs entrees dans x
n <- length(data)
m <- mean(data)
v <- (1-1/n)*var(data)
return(list(moyenne=m,varempir=v))
}
```

La commande `res <- statdescript(data=c(1,2,3,2,1,3))` crée une liste `res` constituée de deux objets `res$moyenne` et `res$varempir`.

NB : On peut taper la fonction ligne après ligne dans `R`, mais dans ce cas on n'a pas de sauvegarde. On peut aussi l'écrire dans un fichier avec le nom `statdescript.R`. Avant de pouvoir l'utiliser, il faut alors la charger en cliquant sur le bouton *Source* de RStudio ou en tapant la commande `source('statdescript.R')` dans la fenêtre de commande de `R`.
A chaque fois que l'on fait une modification dans le fichier, il faut refaire cette manipulation avant de l'utiliser pour que les modifications apportées soient prises en compte.

Exercice 23.

1. Ecrire la fonction `statdescript` décrite précédemment dans un fichier en utilisant l'éditeur de texte de RStudio, la sauvegarder en lui donnant le nom `statdescript.R`, puis charger cette fonction dans `R` en cliquant sur le bouton *Source* de RStudio ou en tapant la commande source('statdescript.R') dans la fenêtre de commandes de `R`. Vous pouvez maintenant l'utiliser comme n'importe quelle fonction de `R` (dans les prochains exercices, je ne détaillerai plus ces étapes nécessaires à la création d'une nouvelle fonction).
2. Taper la commande `res <- statdescript(data=c(1,2,3,2,1,3))` et faire afficher le contenu de `res$moyenne` et `res$varempir`.
3. Exécuter les commandes suivantes :

   ```
   u <- sample(x=1:10, replace=TRUE, size=1000, prob=rep(x=1,times=10))
   statdescript(u)
   ```

   Expliquer ce que font les 2 commandes et ce que représentent les valeurs obtenues.

Exercice 24. Si `x` est un vecteur, la fonction `sort(x)` ordonne les éléments du vecteur `x` par ordre croissant.

1. Analyser ce que fait la fonction suivante :

   ```
   FR <- function(x){
      n <- length(x)
      s <- sort(x)
      t <- (1:n)/n
      plot(x=s, y=t, type='s',col='red')
   }
   ```

   Si `x` est le vecteur `c(-1,0,2,3,1)`, quel tracé obtient-on ?
2. Copier cette fonction dans un fichier `FR.R` et faire lire ce fichier par `R`.
3. Simuler le lancer d'un dé 1000 fois et enregistrer les résultats dans un vecteur `D`. Puis appliquer la fonction `FR` au vecteur D obtenu. Que représente la valeur de la courbe au point 3.4 ? au point 5 ?

Histogramme

On va illustrer la représentation de données sur un exemple. Supposons que l'on dispose d'un vecteur x contenant les nombres obtenus en lançant un dé 100 fois. On peut représenter graphiquement ces données en calculant d'abord la la proportion de fois où les chiffres $1, 2, \ldots, 6$ apparaissent et en représentant ces proportions sous forme d'une diagramme en bâtons : le résultat est ce que l'on appelle un histogramme.
Dans `R`, la fonction `hist` va permettre de faire facilement le calcul des proportions. Plus généralement, la fonction `hist` appliquée à un vecteur de données x permet d'obtenir le nombre de données appartenant à des intervalles $[a_1, a_2[$, $[a_2, a_3[$, $\ldots$, $[a_{k_1}, a_k[$ dont les bords $a_1, \ldots, a_k$ sont à préciser dans une variable appelée `breaks`.

Quelques options de la fonction `hist`	
`breaks`	vecteur définissant les bords des intervalles
`freq`	(freq=TRUE, option par défaut) pour obtenir l'histogramme des effectifs, *frequency en anglais* (freq=FALSE).pour obtenir l'histogramme des proportions
`col`	pour indiquer la couleur des barres
`plot`	plot=FALSE, pour que `hist` renvoie la liste des points de ruptures et les effectifs sans faire de tracé.
`right`	right=TRUE, pour que les intervalles de la partition soient de la forme $]a, b]$ (par défaut ils sont de la forme $[a, b[$)

Exercice 25.

1. Exécuter le programme suivant

```
n <- 100
# simulation
u <- 1:6
d <- sample(x = u, prob = rep(x=1/6, times=6), size = n, replace = TRUE)
print(d)
# calcul des proportions
bords <- seq(from = 0.5, to = 6.5, by = 1)
h <- hist(x = d, breaks = bords, plot = FALSE)
print(h)
```

2. `h` est un objet de type 'list' : chaque objet de la liste `h` peuvent être extrait en tapant h$*nom* où *nom* désigne le nom de l'objet. Utiliser l'aide de `hist` pour comprendre le contenu des différents objets de `h`.
3. Compléter ce programme en ajoutant la ligne de commande
   `plot(h$mids,h$density,type='h',xlim=c(0,7), ylim=c(0,0.3))`
   pour qu'il trace le diagramme en bâtons des proportions.
4. Réexécuter cette simulation. Observe-t-on le même histogramme ?
5. Réexécuter cette simulation deux fois avec $n = 1000$, puis $n = 10000$. Qu'observe-t-on ?

Exercice 26. La fonction suivante permet de simuler plusieurs sondages dans une population.

```
rsondage <- function(nsim,n,na,nb){
  N <- na + nb
  res <- numeric(nsim)
  for(i in 1:nsim){
    u <- sample(x=c(rep(1,na),rep(0,nb)),size=n,replace=FALSE,prob=rep(1/N,N))
    res[i] <- mean(u)
  }
return(res)
}
```

1. Décrire le fonctionnement de cette fonction : quel type de sondage permet de simuler cette fonction ? Que représentent les arguments de la fonction `nsim`, `n`, `na` et `nb` ? Que retourne la fonction ? (taille du vecteur retourné par la fonction ? signification des valeurs conservées dans ce vecteur ?)
2. Ecrire un programme qui :
   - utilise cette fonction pour simuler 1000 sondages de 31 personnes dans une population de taille 500 composée de 200 personnes votant pour A et 300 personnes votant pour B.
   - représente les résultats de ces 1000 sondages sous forme d'un diagramme en bâtons, la hauteur d'un bâton d'abscisse $k/31$ représentant la proportion de sondages dans lesquels k personnes sur les 31 unes interrogées votent pour A.
   - calcule le nombre de sondages où au moins de 26 personnes votent pour A.

3. Exécuter ce programme deux fois et commenter les résultats obtenus en comparant avec les résultats théoriques obtenus dans l'exercice 15.

La condition si ... alors ... sinon ...

– Un test `si ... alors ...` permet d'exécuter des instructions seulement si une condition est réalisée.

```
if( condition ){
....                    #liste d'instructions à exécuter
....}
```

Exemple : La fonction `test(x)` suivante retourne 1 si `x` est un nombre strictement positif et retourne 0 dans les autres cas.

```
test <- function(x){
  e <- 0
  if(x > 0){e <- 1 }
  return(e)
}
```

– Un test `si ... alors ... sinon ...` permet d'exécuter des instructions différentes selon qu'une condition est vraie ou fausse.

```
if( condition ){
....                    #liste d'instructions à exécuter si condition est vraie
} else {  ....
....                    #liste d'instructions à exécuter si condition est faux
....}
```

Exemple : La fonction `signe(x)` suivante retourne 1 si `x` est un nombre strictement positif, -1 si `x` est un nombre strictement négatif et 0 si $x = 0$.

```
signe <- function(x){
  e <- 0
  if(x > 0){
    e <- 1
  }
  else{
    if(x < 0){e <- -1}
  }
  return(e)
}
```

Exercice 27. Analyser le programme suivant :

```
x <- c(0,0)
u <- sample(x=1:2,size=2,replace=TRUE,prob=c(1/3,2/3))
if( u[1] == 1 ) { x[1] <- x[1] + 1} else { x[1] <- x[1] - 1}
if( u[2] == 1 ) { x[2] <- x[2] + 1} else { x[2] <- x[2] - 1}
```

Quelles sont les valeurs possibles pour `u` ? Pour chaque valeur possible pour `u`, trouver quelle sera la valeur de `x` à la fin du programme. Ce programme simule une réalisation d'un vecteur aléatoire X. Quelle est la loi de X ?

La boucle `while`

La boucle `while` permet d'exécuter une listes d'instructions tant qu'une condition est satisfaite :

```
while( conditions ){
....                    #liste d'instructions à exécuter tant que conditions vraies
....}
```

Exemple : Le programme suivant

```
s <- 0
i <- 1
while (i<=4){
  s <- s + i
  i <- i + 1
  print(s)
}
```

exécute 4 fois les instructions qui sont entre accolades, la variable `i` prend donc successivement les valeurs 1, 2, 3 et 4 et `s` prendra successivement les valeurs 0, 1, $1+2$, $1+2+3$, $1+2+3+4$. Comme la commande `print(s)` affiche à l'écran la valeur de `s`, ce programme affiche

```
[1] 1
[1] 3
[1] 6
[1] 10
```

Exercice 28. Expliquer ce que fait la fonction suivante :

```
trouver <- function(x,a){
  # x est un vecteur
  # a est un nombre
  n <- length(x)
  i <- 1
  t <- 0
  while (x[i]!= a & i <= n){
    i <- i+1
  }
  if ( i<=n){ t <- i }
  return(t)
}
```

Qu'obtiendra-t-on si après avoir fait lire la fonction à `R`, on tape la commande `trouver(c(0,1,0,3,1),1)` ? on tape la commande `trouver(c(0,1,0,3,1),2)` ?

Exercice 29. Expliquer ce que fait le programme suivant :

```
n <- 500
w <- numeric(n)
for(i in 1:n){
  u <- sample(x = 1:6, prob = rep(1/6,6), size = 1, replace = TRUE)
  k <- 1
  while(u < 6){
    u <- sample(x = 1:6, prob = rep(1/6,6), size = 1, replace = TRUE)
    k <- k + 1
  }
  w[i] <- k
}
mean(w)
```

Le vecteur `w` peut être vu comme contenant 500 réalisations d'une variable aléatoire W. Quelle est la loi de W ? Compléter ce programme pour qu'il affiche la loi empirique associée aux données contenues dans le vecteur `w` ainsi que la loi de W. Expliquer ce que fait le programme suivant :

```
n <- 500
w <- numeric(n)
for(i in 1:n){
  u <- sample(x = 1:6, prob = rep(1/6,6), size = 1, replace = TRUE)
  k <- 1
  while(u < 6){
    u <- sample(x = 1:6, prob = rep(1/6,6), size = 1, replace = TRUE)
    k <- k + 1
  }
  w[i] <- k
}
mean(w)
```

Compléter ce programme pour qu'il affiche l'histogramme associé aux données contenues dans le vecteur `w`. Le vecteur `w` peut être vu comme contenant 500 réalisations d'une variable aléatoire W. Quelle est la loi de W ?

Les matrices

Une *matrice de dimensions* $\ell \times c$ est un tableau de nombres à ℓ lignes et c colonnes. Le coefficient (i,j) de la matrice est le coefficient qui se trouve sur la i-ème ligne et la j-ème colonne.

$$M = \begin{pmatrix} M(1,1) & \cdots & M(1,j) & \cdots & M(1,c) \\ \vdots & & \vdots & & \vdots \\ M(i,1) & \cdots & \mathbf{M(i,j)} & \cdots & M(i,c) \\ \vdots & & \vdots & & \vdots \\ M(c,1) & \cdots & M(c,j) & \cdots & M(\ell,c) \end{pmatrix}.$$

Exemple : $M = \begin{pmatrix} 1 & 2 & 3 \\ -2 & -1 & 0 \end{pmatrix}$ est une matrice à 2 lignes et 3 colonnes. Le coefficient $(2,1)$ de M est égale à -2.

Si $\ell = 1$, on parle de vecteur ligne : $M = (M(1), \ldots, M(c))$.

Si $c = 1$, on parle de vecteur colonne : $M = \begin{pmatrix} M(1) \\ \vdots \\ M(\ell) \end{pmatrix}$.

Création d'une matrice dans R

Pour créer une matrice dans **R**, on utilise la fonction `matrix`.

Exemples :

– La commande suivante crée une matrice A à 2 lignes et 3 colonnes dont tous les éléments sont nuls.

```
A <- matrix(data=0, nrow=2, ncol=3)
```

NB : Pour créer un vecteur ligne dont tous les éléments valent 0, il suffit d'utiliser la commande `numeric`.

– La commande suivante crée la matrice $B = \begin{pmatrix} 1 & 2 \\ 3 & -2 \\ -1 & 0 \end{pmatrix}$.

```
B <- matrix(data = c(1, 2, 3, -2, -1, 0), nrow = 3, ncol = 2, byrow=TRUE)
```

NB : Par défaut, les éléments mis dans `data` sont entrés par colonne (i.e. `byrow=FALSE`).

La commande `dim(B)` donne les dimensions de la matrice B, c'est-à-dire un vecteur contenant le nombre de lignes et de colonnes de B.

L'extraction ou la modification d'un élément

Dans **R**,

`M[i,j]` désigne l'élément de la i-ème ligne et de la j-ème colonne de la matrice M,

`M[i, ]` est le vecteur contenant les éléments de la i-ème ligne,

`M[ ,j]` est le vecteur contenant les éléments de la j-ème colonne.

NB : Rappelons que si `v` est un vecteur ligne `v[i]` désigne le `i`-ème élément de `v`.

Exercice 30. Exécuter les intructions suivantes, puis analyser les différentes commandes :

```
M <- matrix(data = c(1, 2, 3, -2, -1, 0), nrow = 2, ncol = 3); M
Q <- matrix (data = 0, nrow = 2, ncol = 3); Q
Q[1,1] <- 10; Q[2,2:3] <- c(5,10) ; Q
```

Comment créer un vecteur v qui contienne les éléments de la 2ème ligne de la matrice `Q` ?

Quelques opérations élémentaires sur les matrices

On peut :

1. **additionner ou soustraire deux matrices A et B de mêmes dimensions** : $C = A + B$ est une matrice de même dimensions que A et B dont le coefficient (i,j) est la somme des coefficients (i,j) des matrices A et B. Dans **R**, pour définir la matrice C on utilise la commande : `C <- A+B`
   Exemple : Si $A = \begin{pmatrix} 4 & 0 \\ 1 & 2 \end{pmatrix}$ et $B = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$, alors $A + B = \begin{pmatrix} 5 & 2 \\ 4 & 6 \end{pmatrix}$.
2. **multiplier une matrice A par nombre** c : cela revient à créer une matrice de même dimensions que A en multipliant chaque élément de A par c. Dans **R**, la commande est : `c*A`
   Exemple : Si $A = \begin{pmatrix} 4 & 0 & 1 \\ 1 & 2 & 3 \end{pmatrix}$ alors $2A = \begin{pmatrix} 8 & 0 & 2 \\ 2 & 4 & 6 \end{pmatrix}$
3. **intervertir les coefficients des lignes et des colonnes d'une matrice** : si A est une matrice à n lignes et m colonnes alors la commande **R** : `B <- t(A)` définit une matrice à m lignes et n colonnes de sorte que

$B(i,j) = A(j,i)$ pour tout $1 \leq i \leq m$ et $1 \leq j \leq n$ (on dit que B est la transposée de la matrice A, on la note souvent ${}^tA$ en mathématique).

Exemple : Si $A = \begin{pmatrix} 4 & 0 & 1 \\ 1 & 2 & 3 \end{pmatrix}$ alors la transposée de A est ${}^tA = \begin{pmatrix} 4 & 1 \\ 0 & 2 \\ 1 & 3 \end{pmatrix}$.

Exercice 31. (suite de l'exercice 30)

1. Calculer à la main la matrice $M - 2Q$. Vérifier le résultat en utilisant **R**.
2. Exécuter la commandes suivante : `W <- t(M)` .
   Comparer la matrice W à celle de M.
3. Comment diviser tous les éléments de `W` par 4 ?

La multiplication de deux matrices

Il existe deux opérations différentes dans **R** qu'il ne faut pas confondre, la multiplication élément par élément et ma multiplication matricielle.

Multiplication élément par élément Si A et B sont deux matrices de même dimension, dans **R** la commande `C <- A*B` crée une matrice C de même taille que A et B dont le coefficient (i,j) est $A[i,j] * B(i,j]$. On appellera cette opération la « multiplication élément par élément ».

Multiplication matricielle Supposons que A soit une matrice avec n lignes et m colonnes et que B soit une matrice avec m lignes et p colonnes, alors la multiplication de A par B notée AB donne une matrice C qui a le même nombre de lignes que A et le même nombre de colonnes que B ; c'est donc une matrice à n lignes et p colonnes. Le coefficient (i,j) de la matrice $C = AB$ est défini par la formule suivante :

$$C(i,j) = \sum_{k=1}^{m} A(i,k)B(k,j) = A(i,1)B(1,j) + \cdots + A(i,m)B(m,j).$$

Les coefficients qui interviennent dans le calcul de $C(i,j)$ sont ceux de la i-ième ligne de A et de la j-ième colonne de B (ils sont écrits en gras ci-dessous).

$$\begin{pmatrix} C(1,1) & \cdots & C(1,k) & \cdots & C(1,m) \\ \vdots & & \vdots & & \vdots \\ C(i,1) & \cdots & \mathbf{C(i,j)} & \cdots & C(i,m) \\ \vdots & & \vdots & & \vdots \\ C(\ell,1) & \cdots & C(\ell,k) & \cdots & C(\ell,m) \end{pmatrix} = \begin{pmatrix} A(1,1) & \cdots & A(1,j) & \cdots & A(1,n) \\ \vdots & & \vdots & & \vdots \\ \mathbf{A(i,1)} & \cdots & \mathbf{A(i,k)} & \cdots & \mathbf{A(i,n)} \\ \vdots & & \vdots & & \vdots \\ A(\ell,1) & \cdots & A(\ell,j) & \cdots & A(\ell,n) \end{pmatrix} \begin{pmatrix} B(1,1) & \cdots & \mathbf{B(1,j)} & \cdots & B(1,m) \\ \vdots & & \vdots & & \vdots \\ B(k,1) & \cdots & \mathbf{B(k,j)} & \cdots & B(k,m) \\ \vdots & & \vdots & & \vdots \\ B(n,1) & \cdots & \mathbf{B(n,j)} & \cdots & B(n,m) \end{pmatrix}$$

La multiplication d'une matrice A par une matrice B au sens mathématique (notée AB) s'obtient, dans **R**, par la commande `C <- A%*%B` .

Exemple : Considérons les matrices $A = \begin{pmatrix} 2 & 0 \\ 1 & 3 \end{pmatrix}$ et $B = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$. La matrice $C = AB$ est une matrice de dimension 2×3. Les shémas suivants indiquent une façon de positionner les matrices pour faire les calculs plus aisément. Les coefficients écrits en gras sont ceux qui sont utilisés pour faire le calcul de $C(1,1)$ à gauche puis de $C(1,2)$ à droite.

$$B = \begin{pmatrix} \downarrow & 2 & 3 \\ \mathbf{1} & 5 & 6 \\ \mathbf{4} & & \end{pmatrix} \qquad B = \begin{pmatrix} 1 & \downarrow & 3 \\ 4 & \mathbf{5} & 6 \end{pmatrix}$$

$$A \rightrightarrows \begin{pmatrix} \mathbf{2} & \mathbf{0} \\ 1 & 3 \end{pmatrix} \quad C = \begin{pmatrix} \mathbf{2} & ? & ? \\ ? & ? & ? \end{pmatrix} \qquad A \rightrightarrows \begin{pmatrix} \mathbf{2} & \mathbf{0} \\ 1 & 3 \end{pmatrix} \quad C = \begin{pmatrix} 2 & \mathbf{4} & ? \\ ? & ? & ? \end{pmatrix}$$

$$\begin{aligned} C(1,1) &= A(1,1)B(1,1) + A(1,2)B(2,2) \\ &= 2 \times 1 + 0 \times 4 = 2 \end{aligned} \qquad \begin{aligned} C(1,2) &= A(1,1)B(1,2) + A(1,2)B(2,2) \\ &= 2 \times 2 + 0 \times 5 = 4. \end{aligned}$$

En continuant ainsi on obtient : $C = AB = \begin{pmatrix} 2 & 4 & 6 \\ 13 & 17 & 21 \end{pmatrix}$. **NB :**

– La multiplication de matrices est une opération associative : si A est une matrice de dimensions (m,n), B est une matrice de dimensions (n,r) et C est une matrice de dimensions (r,s) alors $(AB)C = A(BC)$.

– La multiplication de matrices n'est pas une opération commutative : si A est de dimensions (m,n) et B est une matrice de dimensions (n,ℓ) alors AB est une matrice de dimension (m,ℓ) et le produit de B par A n'est définie que si $m=\ell$. Lorsque $m=\ell$, BA est une matrice carrée de taille m. Même si A et B sont des matrices carrées de même taille, AB et BA sont des matrices de même taille, mais différentes en général.
Exemple :

$$\text{Si } A=\begin{pmatrix}1&0\\0&2\end{pmatrix} \text{ et } B=\begin{pmatrix}1&2\\3&4\end{pmatrix}, \text{ alors } AB=\begin{pmatrix}1&2\\6&8\end{pmatrix} \text{ et } BA=\begin{pmatrix}1&4\\3&8\end{pmatrix}$$

Exercice 32.

1. Soit $A=\begin{pmatrix}1 & 10\end{pmatrix}$ et $B=\begin{pmatrix}2&3\\0&-1\end{pmatrix}$. Calculer la matrice AB (multiplication matricielle) à la main et vérifier ensuite le résultat obtenu en utilisant **R**.
2. Soit $C=\begin{pmatrix}1&-1\\2&2\end{pmatrix}$ et $D=\begin{pmatrix}3&2\\3&4\end{pmatrix}$ Calculer les matrices CD et DC avec **R**.
3. Trouver un vecteur ligne U tel que $UB=(2,4)$. Vérifier le résultat obtenu en utilisant **R** pour faire le calcul de UB.

NB : Pour résoudre un système linéaire dans **R**, on dispose de la fonction `solve` : si A est une matrice carrée ayant m lignes et m colonnes, si b est un vecteur colonne à m lignes et si $Ax=b$ a une unique solution alors `solve(A,b)` calcule numériquement cette solution (un message d'erreur s'affiche quand la solution n'est pas unique).

La matrice identité On appelle *matrice identité de taille n* et on la note I_n, la matrice définie par $I_n(i,i)=1$ pour $i\in\{1,\dots,n\}$ et $I_n(i,j)=0$ si $i\neq j$ (matrice diagonale dont tous les coefficients sur la diagonale valent 1) :

$$I_n=\begin{pmatrix}1&0&\cdots&\cdots&0\\0&1&0&\cdots&0\\\vdots&\ddots&\ddots&\ddots&\vdots\\0&\cdots&0&1&0\\0&\cdots&\cdots&0&1\end{pmatrix}.$$

Pour toute matrice carrée A de dimension n, on a $AI_n=I_nA=A$.

Les puissances d'une matrice carrée Si A est une matrice carrée, la multiplication de A par elle-même se note A^2. Plus généralement, on définit la puissance n-ième de la matrice A récursivement par : $A^1=A$ et $A^{m+1}=A^mA$ pour tout $m\in\mathbb{N}^*$. Par convention, on définit A^0 comme la matrice identité de même taille que A. Avec cette convention, on a pour tout entiert positif n et m, $A^{n+m}=A^nA^m$.

Exercice 33. On considère la matrice suivante $Q=\begin{pmatrix}0.1&0.9\\0.3&0.7\end{pmatrix}$. Déterminer $T=Q^2$ à la main. Vérifier le résultat obtenu en utilisant **R**, puis toujours avec **R**, calculer $U=Q^3$ et $V=Q^4$. Que vaut la somme des coefficients de chaque ligne des matrices Q, T, U et V ?

Exercice 34.

1. Analyser la fonction suivante

```
itermat <- function(v,P,nb){
# v est un vecteur
# P est une matrice carree ayant meme nb de colonnes que P
# nb est un entier supérieur à 1
  L <- matrix(data=0, nrow=nb+1, ncol=length(v))
  L[1,] <- v
  for( i in 1:nb){
    L[i+1,] <- L[i,]%*%P
  }
  return(L)
}
```

On désigne par x le vecteur formé par les coefficients de la 3-ème ligne de la matrice L. Donner une expression de x en fonction de v et de P.
On désigne par y le vecteur formé par les coefficients de la dernière ligne de la matrice L. Donner une expression de y en fonction de v et de P.

2. Exécuter la fonction `itermat(v=u,P=Q,nb=20)` , Q étant la matrice définie dans l'exercice précédent et u étant le vecteur $c(1,0)$. Faire de même en remplaçant u par le vecteur $c(0,1)$ puis par le vecteur $c(0.2,0.8)$. Qu'observez-vous ?

La superposition de plusieurs tracés avec `matplot`

La fonction `matplot(x,y)` permet de tracer sur une même figure avec des symboles différents plusieurs séries de points, toutes les séries de points devant contenir le même nombre de points :
– si `x` est un vecteur de longueur n et si `y` est une matrice ayant n lignes et k colonnes, il y aura k tracés différents pour chaque colonne de la matrice `y` avec toujours comme abscisses les valeurs définies dans `x`.
– Si `x` et `y` sont deux matrices de même taille, il y aura un tracé pour les points définis par `x[,1]` et `y[,1]`, un 2ème tracé pour les points définis par `x[,2]` et `y[,2]`, etc.
Cette fonction dispose d'arguments comme `plot` pour préciser le type de tracés, les légendes pour les axes ... (voir l'aide en ligne).

Exemple : Les commandes suivantes permettent d'afficher les graphes des fonctions $x \mapsto \cos(x)$ et $x \mapsto 2\sin(x)$ sur l'intervalle $[0, 4\pi]$:

```
t <- seq(from=0, to=4*pi, by=0.1)
F <- matrix(data=c(cos(t),2*sin(t)),nrow= length(t), ncol=2, byrow=FALSE)
matplot(x=t, y=F, type='l')
```

Exercice 35. Exécuter le programme suivant :

```
n <- 4 ; k <- 10000 ;
t <- 1:k
A <- matrix(data=0,nrow=k,ncol=n)
for(i in 1:n){
  U <- sample(x=c(0,1),size=k,prob=c(1/4,3/4),replace=TRUE)
  A[,i] <- cumsum(U)/t
}
matplot(x=t,y=A,type='s')
```

Analyser ce programme. Que représentent les différents tracés ? Y-a-t-il un résultat dans le cours qui explique le comportement de ces courbes ?

NB : Si on veut superposer des tracés qui ne sont pas construits avec le même nombre de points, on utilisera :
– soit plusieurs fois la commande `plot` séparée par la commande `par(new=TRUE)` qui indique de ne pas effacer. L'échelle en x et en y est à définir pour chaque tracé.
– soit pour le premier tracé la commande `plot` et pour les autres tracés la commande `points` (même syntaxe que `plot`). L'échelle en x et en y est alors celle fixée pour le 1er tracé.
Exemple : On obtiendra le même tracé qu'à l'exemple précédent avec les commandes suivantes :

```
t <- seq(from=0, to=4*pi, by=0.1)
plot(t,cos(t),type='l',ylab=' ',ylim=c(-2,2))
par(lty=2)
points(t,2*sin(t),type='l',col='red')
```

Si on n'avait pas mis `ylim=c(-2,2)` dans la commande `plot`, l'échelle en y irait de -1 à 1 et donc le second tracé ne serait pas visible entièrement.
On peut aussi utiliser 2 fois `plot` ce qui donne :

```
t <- seq(from=0, to=4*pi, by=0.1)
plot(t,cos(t),type='l',xlab=' ',ylim=c(-2,2))
par(new=TRUE,lty=2)
plot(t,2*sin(t),type='l',col='red',ylim=c(-2,2))
```