T. P. n° 4
Simulation de lois avec le logiciel `R`

Commandes utiles pour ce T.P. : Quelques lois usuelles

Loi	Nom	Paramètres	Valeurs par défaut
Beta	`beta`	`shape1`, `shape2`	
Binomiale	`binom`	`size`, `prob`	
Exponentielle	`exp`	`1/mean`	1
Fisher	`f`	`df1`, `df2`	
Gamma	`gamma`	`shape`, `1/scale`	$-, 1$
Géométrique	`geom`	`prob`	
Hypergéométrique	`hyper`	`m`, `n`, `k`	
Khi-deux	`chisq`	`df`	
Normale	`norm`	`mean`, `sd`	$0, 1$
Poisson	`pois`	`lambda`	
Student	`t`	`df`	
Uniforme	`unif`	`min`, `max`	$0, 1$

Remarque : Dans ce tableau, nous remarquons que la loi gamma est définie par deux paramètres tandis que parfois la loi gamma n'est définie que par un seul paramètre noté r. En fait, la définition la plus générale de la loi gamma, d'après le livre de Foata et Fuchs, « Calcul des probabilités », Éditions Masson ou Dunod est la suivante :

Définition 0.1 *Une variable aléatoire positive X suit une loi gamma de paramètres $r > 0, \lambda > 0$ si elle est absolument continue et admet pour densité*

$$f_X(t) = \frac{\lambda}{\Gamma(r)} \exp(-\lambda t)(\lambda t)^{r-1}, \textit{pour tout } t \geqslant 0,$$

et $f_X(t) = 0$ sinon.

Exercice 1 Espérance et variance d'une loi $\gamma(r, \lambda)$.

Soit X une variable aléatoire réelle qui suit une loi gamma de paramètres r et λ.

1. Calculer l'espérance mathématique de X ($\mathbb{E}[X] = r/\lambda$).
2. Calculer la variance de X à l'aide de la formule de Huygens ($\mathrm{Var}[X] = r/\lambda^2$).

Pour chacune de ces distributions, nous disposons de quatre commandes préfixées par une des lettres `d, p, q, r` et suivi du nom de la distribution :

- `dnomdist` : il s'agit de la fonction de densité pour une distribution de probabilité continue et de la fonction de probabilité ($\mathbb{P}[X = k]$) dans le cas d'une loi discrète ;
- `pnomdist` : il s'agit de la fonction de répartition ($\mathbb{P}[X < x]$) ;
- `qnomdist` : il s'agit de la fonction des quantiles, c'est-à-dire la valeur pour laquelle la fonction de répartition atteint une certaine probabilité ; dans le cas discret, cette fonction renvoie le plus petit entier u tel que $F(u) \leqslant p$ où F est la fonction de répartition de la distribution considérée ;
- `rnomdist` : génère des réalisations aléatoires indépendantes de la distribution `nomdist`.

Exercice 2 Un premier programme.

Que font ces lignes de commande ? Indiquer ce que R vous retorune.

```
> qnorm(0.975) ;
> dnorm(0) ;
> pnorm(1.96) ;
> rnorm(20) ;
> rnorm(10,mean=5,sd=0.5) ;
> x=seq(-3,3,0.1) ;pdf=dnorm(x) ;plot(x,pdf,type="l") ;
> runif(3) ;
> rt(5,10) ;
```

Remarques :

1. Nous avons écrit `x=seq(-3,3,0.1)`. Ici `x` est un nom de variable. Les noms de variables sont très flexibles. N'importe quelle variable peut stocker n'importe quelle valeur (il n'y a pas besoin de déclarer les variables). Cependant, il faut savoir que :
   - Les noms de variables ne peuvent pas commencer par un chiffre ou un caractère spécial.
   - Les noms sont sensibles à la casse des caractères. Un caractère minuscule comme `x` est différent d'un caractère majuscule comme `X`.
   - Quelques noms courants sont déjà utilisés par R. *e.g.* `c, q, t, C, D, F, I, T` et par conséquent doivent être évités. La liste des noms prédéfinis dans la bibliothèque de base peut être consultée ainsi :
     ```
     > noms <- ls("package :base")
     > length(noms)
     ```
     Combien y en a-t-il d'installer sur votre ordinateur sur lequel vous travaillez ? Si vous souhaitez les voir apparaitre à l'écran, tapez `noms`.
2. Dans cette liste de commandes, l'opérateur `=` a été utilisé. Comme la plupart des langages de programmation, R a des variables auxquelles nous pouvons affecter une valeur. Pour cela, nous utilisons l'opérateur `<-` ou `->`. L'opérateur classique `=` marche aussi.
3. De plus dans cette liste de commandes, deux **fonctions** sont intervenues :
   - `seq()`
   - `plot()`

Vous remarquerez que les appels aux fonctions sous R sont indiqués par la présence de **parenthèses**. De plus, la plupart des choses utiles sous R sont faites par des fonctions. De plus, `length()` et `ls()` sont aussi des fonctions. Pour en savoir d'avantage sur ces fonctions, tapez `?sujet` que nous pouvons aussi écrire `help(sujet)`. Toutes les fonctions de R ont une page d'aide. Quand vous connaissez le nom de la fonction ou du sujet qui vous intéresse, c'est en général le meilleur moyen d'apprendre à l'utiliser.
Les pages d'aide sont généralement très détaillées. Elles contiennent souvent, entre autres :

- Une section `See Also` qui donne les pages d'aide sur des sujets apparentés.
- Une section `Description` de ce que fait la fonction.
- Une section `Examples` avec du code illustrant ce que fait la fonction documentée. Ces exemples peuvent être exécutés directement en utilisant la fonction `example()`, essayez par exemple :
  ```
  > example(plot)
  ```

4. La manière la plus simple de produire des graphiques sous R est d'utiliser la fonction `plot()`.
   ```
   > plot(weight~height, data=women)
   ```
   Les fonctions graphiques de comportent de nombreuses options qui permettent de contrôler de façon très fine les graphiques. Par exemple, les paramètres de la fonction `plot()` utilisés par défaut sont :
   ```
   > args(plot.default)
   function (x, y = NULL, type = "p", xlim = NULL, ylim = NULL,
   log = "", main = NULL, sub = NULL, xlab = NULL, ylab = NULL,
   ann = par("ann"), axes = TRUE, frame.plot = axes, panel.first =
   NULL,
   panel.last = NULL, asp = NA, ...)
   NULL
   ```
   L'argument ... signifie qu'il y a encore d'autres paramètres graphiques possibles. Ils sont contrôlés par la fonction `par()`.
   ```
   > names(par())
   [1] "xlog" "ylog" "adj" "ann" "ask" "bg"
   [7] "bty" "cex" "cex.axis" "cex.lab" "cex.main" "cex.sub"
   [13] "cin" "col" "col.axis" "col.lab" "col.main" "col.sub"
   [19] "cra" "crt" "csi" "cxy" "din" "err"
   [25] "family" "fg" "fig" "fin" "font" "font.axis"
   [31] "font.lab" "font.main" "font.sub" "gamma" "lab" "las"
   [37] "lend" "lheight" "ljoin" "lmitre" "lty" "lwd"
   [43] "mai" "mar" "mex" "mfcol" "mfg" "mfrow"
   [49] "mgp" "mkh" "new" "oma" "omd" "omi"
   [55] "pch" "pin" "plt" "ps" "pty" "smo"
   [61] "srt" "tck" "tcl" "usr" "xaxp" "xaxs"
   [67] "xaxt" "xpd" "yaxp" "yaxs" "yaxt"
   ```
   Pour une exploration systématique des paramètres graphiques, voir le TD 3 ou la fiche

`http ://pbil.univ-lyon1.fr/R/fichestd/tdr75.pdf`.
Un exemple de graphique utilisant quelques options :

```
> plot(weight ~height, pch=19, col="royalblue3",
+ las=1,main="Weight vs. height", xlab="Height",
+ ylab=Weight", data="women")
```

Il existe une autre fonction que `plot()` pour faire des graphiques. Cette fonction est la fonction `curve()`.

Exercice 3 À vous maintenant.

Dans le polycopié de cours, il manque le tracé de certaines densités et de fonctions de répartition.

1. Tracer la densité et la fonction de répartition pour la loi $\gamma(r, \lambda)$ pour les valeurs suivantes de r et $\lambda = 1$:

$$1/2, \quad 1, \quad 3/2, \quad 2, \quad 3.$$

   Mettre un titre sur chaque graphique. Ensuite, sur un même graphique, tracer les densités pour les différentes valeurs du paramètre r demandées et mettre un titre.

2. Tracer la densité et la fonction de répartition pour la loi bêta de type I pour les valeurs suivantes de n et p :

$$(1/2, 1/2), \quad (1, 1), \quad (1, 2), \quad (2, 1), \quad (3, 3), \quad (9, 2).$$

   Mettre un titre sur chaque graphique. Ensuite, sur un même graphique, tracer les densités pour les différentes valeurs du paramètre r demandées et mettre un titre.

3. Tracer la densité et la fonction de répartition pour la loi du chi-deux pour la valeur suivante de p :

$$1, \quad 2, \quad 3, \quad 4, \quad 5, \quad 6, \quad 7, \quad 8, \quad 9, \quad 10, \quad 15, \quad 20, \quad 25, \quad 30, \quad 40, \quad 50.$$

   Mettre un titre sur chaque graphique. Ensuite, sur un même graphique, tracer les densités pour les différentes valeurs du paramètre r demandées et mettre un titre.

4. Tracer la densité et la fonction de répartition pour la loi de Fisher-Snedecor pour les valeurs suivantes de n et p :

$$(1, 2), \quad (2, 2), \quad (3, 2), \quad (4, 2), \quad (5, 2).$$

   Mettre un titre sur chaque graphique. Ensuite, sur un même graphique, tracer les densités pour les différentes valeurs du paramètre r demandées et mettre un titre.

5. Tracer la densité et la fonction de répartition pour la loi de Student pour la valeur suivante de n :
$$1, \quad 2, \quad 5, \quad 10, \quad 50.$$
Mettre un titre sur chaque graphique. Ensuite, sur un même graphique, tracer les densités pour les différentes valeurs du paramètre r demandées et mettre un titre.

Exercice 4 Approximation d'une loi de Poisson par une loi normale.

Lorsque λ est grand, la loi de Poisson de paramètre λ peut être approchée par une loi normale de moyenne λ et d'écart-type $\sqrt{\lambda}$.
Par simulation, étudier graphiquement la validité du résultat précédent.
Vous pourrez, par exemple, simuler 1000 réalisations indépendantes d'une loi de Poisson de paramètre $\lambda = 10, 20, 50, 100$ et étudier l'évolution de l'histogramme.
Pour tracer un histogramme sous R, la fonction `hist()` peut-être utile.

Exercice 5 Le théorème de la limite centrée

L'étude de somme de variables aléatoires indépendantes et de même loi joue un rôle important en statistique. Le théorème de la limite centrée établit la convergence vers la loi normale sous des hypothèses peu contraignantes :
Si $(X_n)_{n\in\mathbb{N}^*}$ est une suite de variables aléatoires indépendantes de même loi d'espérance μ et d'écart-type σ, alors, lorsque n tend vers l'infini, la variable aléatoire
$\sqrt{n}\left(\dfrac{\overline{X}_n - \mu}{\sigma}\right)$ converge en loi vers la loi normale centrée-réduite.
Par simulation, vérifier graphiquement la validité du théorème précédent dans le cas d'un échantillon binomial et d'un échantillon exponentiel.

Exercice 6 Densité de lois normales bivariées

Le package dans R permettant d'obtenir la densité, les quantiles ou de générer des réalisations de lois normales multivariées est le package `mvtnorm`.
Que font les instructions suivantes ?

```
> library(mvtnorm)
> help(package="mvtnorm")
> dmvnorm(c(0,0), c(0,0), diag(2), log=FALSE)
```

Nous allons maintenant représenter graphiquement la fonction de densité de deux lois normales bivariées.
Éxecuter les instructions suivantes et identifier leur action. En particulier il faudra déterminer à quoi servent les fonctions `c()`, `cbind()`, `diag()`, `matrix()` ?

```
library(lattice)

# N2(c(0,0),I_2)

> g <- expand.grid(x = seq(-2,2,0.05), y = seq(-2,2,0.05))
> g$z <- dmvnorm(x=cbind(g$x,g$y),mean=c(0,0), sigma=diag(2),
+ log=FALSE)

> wireframe(z ~ x * y, data = g,colorkey = TRUE,drape=TRUE)

# N2(c(0,0),matrix(c(1,0.75,0.75,1),byrow=T,nrow=2))

> var <- matrix(c(1,0.75,0.75,1),byrow=T,nrow=2)
> g <- expand.grid(x = seq(-2,2,0.05), y = seq(-2,2,0.05))
> g$z <- dmvnorm(x=cbind(g$x,g$y),mean=c(0,0), sigma=var, log=FALSE)

> wireframe(z ~ x * y, data = g,colorkey = TRUE,drape=TRUE)
```

Les graphiques précédents sont statiques. Le package `rgl` permet de rémédier à ce défaut et de faire pivoter les graphiques à l'aide de la souris.

```
library(rgl)

# N2(c(0,0),matrix(c(1,0,0,1),byrow=T,nrow=2))

> g <- expand.grid(x = seq(-4,4,0.05), y = seq(-4,4,0.05))
> g$z <- dmvnorm(x=cbind(g$x,g$y),mean=c(0,0), sigma=diag(2),
+ log=FALSE)

> g2z <- matrix(g$z*5000,byrow=T,nrow=length(seq(-4,4,0.05)))

> g2x <- 10 * (1:nrow(g2z))
> g2y <- 10 * (1:ncol(g2z))

> zlim <- range(g2y)
> zlen <- zlim[2] - zlim[1] + 1

> colorlut <- terrain.colors(zlen) # height color lookup table

> col <- colorlut[ g2y-zlim[1]+1 ] # assign colors to heights for
each point

> open3d()
> surface3d(g2x, g2y, g2z, color=col, back="lines")
```

```
# N2(c(0,0),matrix(c(1,0.75,0.75,1),byrow=T,nrow=2))

> g <- expand.grid(x = seq(-4,4,0.05), y = seq(-4,4,0.05))
> g$z <- dmvnorm(x=cbind(g$x,g$y),mean=c(0,0),
+ sigma=matrix(c(1,0.75,0.75,1), byrow=T,nrow=2), log=FALSE)

> g2z <- matrix(g$z*5000,byrow=T,nrow=length(seq(-4,4,0.05)))

> g2x <- 10 * (1:nrow(g2z))
> g2y <- 10 * (1:ncol(g2z))

> zlim <- range(g2y)
> zlen <- zlim[2] - zlim[1] + 1

> colorlut <- terrain.colors(zlen) # height color lookup table

> col <- colorlut[ g2y-zlim[1]+1 ] # assign colors to heights for
each point

> open3d()
> surface3d(g2x, g2y, g2z, color=col, back="lines")
```