Université de Lille **Département de Mathématiques**

M1 M1
Statistiques

Formulaire de R et de python

Pour obtenir l'aide de `R` sur une commande, il faut taper `help("nom de la commande")`. L'aide est en anglais.

Pour `R`, utilisez le logiciel `RStudio`, pour Python, le logiciel `Spyder`. On tape les programmes dans l'éditeur de `R` ou de Python, puis on les applique dans la console. Cela permet de retrouver facilement ce qu'on a fait avant.

Le logiciel `R` est un logiciel matriciel. Il sait bien manipuler de gros tableaux de données. C'est aussi un très bon logiciel de statistique, qui sait générer des variables aléatoires suivant de nombreuses lois, et peut faire automatiquement beaucoup de tests différents.

1 Les différents types de structures (vecteurs, séries temporelles, matrices, tableaux, tableaux de données, listes)

Pour cela, on peut utiliser différents types de structures dans le logiciel `R`.

Les vecteurs sont des suites de données du même type (nombres, chaînes de caractères, ...). Par exemple, $v = (1, -5, 0.245667, 3.974)$ ou nom = ('Marie', 'Nicole', 'Thomas').

- Création
  - `v<-c(nombre1,nombre2,...)` ou `v<-c(nom1,nom2,...)`
  - `seq(from,to,by ou length.out)`. Crée une séquence allant de *from* à *to*, soit par pas de *by*, soit de longueur totale *length.out*.
  - `rep(vecteur ou nombre, nombre de fois)`. Répète un nombre (ou même un vecteur) le nombre de fois demandé.
  - On peut donner des noms aux éléments grâce à la commande `names(v)<-c('nom1','nom2',...)`.
- Accès aux éléments : `v[numéro élément]`

Les séries temporelles correspondent à la mesure d'une quantité de façon régulière au cours du temps. C'est un vecteur auquel on associe une date de début, une date de fin et l'intervalle de temps entre deux observations. On peut regarder `LakeHuron`, `AirPassengers` et `EuStockMarkets`.

- Création : `ts(vecteur,début,fin,frequency(nb obs/unité tps) ou deltat(int entre 2 obs)`
- Accès aux éléments : les données sont traitées comme un vecteur par `R`. Pour accéder aux autres caractéristiques de la série temporelle, on utilise les commandes `start(t)`, `end(t)`, `frequency(t)`.

Les matrices sont des tableaux de données toutes du même type en deux dimensions. Par exemple,

$$M = \begin{pmatrix} 2 & -1.5 \\ 0 & 5 \end{pmatrix} \quad N = \begin{pmatrix} \text{'Marie'} & \text{'Nicolas'} \\ \text{'Adrien'} & \text{'Sandrine'} \end{pmatrix}$$

- Création :
  - `matrix(dvecteur, nblignes, nbcol, byrow=TRUE ou FALSE,dimnames)`. `dvecteur` est l'ensemble des éléments de la matrice, donné sous forme de vecteur, `nblignes` et `nbcol` le nombre de lignes et de colonnes de la matrice, `byrow=TRUE` si on remplit la matrice ligne à ligne, `FALSE` si on la remplit colonne par colonne, `dimnames` est une liste de deux vecteurs donnant le nom des lignes et des colonnes (par défaut, les lignes et les colonnes n'ont pas de nom).
  - d'une matrice diagonale : `diag(vecteur)`
- Accès aux éléments :
  - `M[i,j]` affiche l'élément sur la ième ligne et la jème colonne
  - `M[i:k,]` affiche les lignes i, $i+1$, ..., jusqu'à la kième ligne.

Les tableaux (array) sont un tableau de données toutes du même type, mais qui peut être en dimension plus grande que deux. On peut regarder le jeu de données `Titanic` qui est un array.

- Création : `T<-array(données(vecteur),dim,dimnames)`.
- Accès aux éléments : `T[dim 1, dim2, dim 3,...]`.

Les tableaux de données (data.frame) sont utilisés pour stocker des résultats d'expérience. C'est un tableau à deux dimensions. Chaque ligne représente le résultat d'une expérience, et chaque colonne correspond à quelque chose qui a été mesuré. Par exemple :

étudiant	nom	prénom	âge	cours math L3	moyenne L3
1	Dupont	Marie	21	oui	14,4
2					

Le logiciel `R` compte un grand nombre de tableaux de données déjà implémentés. On peut par exemple regarder `iris` ou `ChickWeight`. Chaque colonne peut être d'un type différent.

- Création : utiliser `data.frame`.
  En réalité, on crée très peu de tableaux de données. Si on dispose d'un tableau de données déjà créé (par exemple sur Excel ou OpenOffice, ou récupéré sur le site de l'Ined), on peut l'importer dans `R`. Pour cela, il faut d'abord le transformer en `.csv` (c'est l'équivalent du `.text` pour les tableaux, tous les tableurs font la conversion), puis utiliser la fonction `read.table`, ou `read.csv` (pour les fichiers codés à la manière anglaise) et `read.csv2` (pour les fichiers codés à la manière française) (en s'assurant que le fichier est dans le bon dossier).
  On peut stocker des tableaux dans un fichier grâce à la commande `write.table`.
- Accès aux éléments :
  - soit comme pour une matrice (en précisant les numéros de lignes et de colonnes qui nous intéressent)

– soit en utilisant les noms des colonnes. Par exemple, pour le tableau de données `iris` : `iris$Sepal.Length` extrait la colonne des longueurs des sépales, `iris$Sepal.Length[3]` donne la longueur du sépale de la troisième fleur étudiée.
– Si on ne veut pas retaper `iris` à chaque fois : `attach(iris)` puis `Sepal.Length[3]`. La commande `attach` permet d'accéder directement aux colonnes d'un tableau de données.

Les listes sont des suites d'objets qui peuvent ne pas être du même type. Par exemple,

```
marie<-list(age=21,sexe='F',filiere='M1 nutrition')
```

Les résultats de beaucoup de fonctions de `R` sont donnés sous forme de liste.

- Création : `list(nomdonnée1=donnée1,nomdonnée2=donnée2,...)`
- Accès aux éléments : `L[[numéro de la donnée]]` soit `L$nomdonnée`.

2 Commandes usuelles

Le logiciel connaît toutes les commandes usuelles que peut faire une calculatrice. Il est capable de traiter en une fois un gros tableau de données, il ne faut donc pas s'en priver. Par défaut, les calculs se font terme à terme (si on a deux matrices A et B de même taille,

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \quad B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

la commande `A*B` va calculer la multiplication terme à terme :

$$\begin{pmatrix} a_{11}b_{11} & a_{12}b_{12} \\ a_{21}b_{21} & a_{22}b_{22} \end{pmatrix}.$$

Ce n'est pas la multiplication mathématique de deux matrices, mais c'est très pratique quand on doit manipuler de grands tableaux de données.

Les fonctions sont listées suivant qu'elles peuvent s'appliquer à des nombres, des vecteurs (et séries temporelles), des matrices (et tableaux de données), et des tableaux.

- Fonctions usuelles à utiliser pour un nombre, un vecteur, une matrice, un tableau (les calculs se font pour chaque élément du vecteur/matrice/tableau dans ce cas).
  – `x^2` (carré du nombre x), `sqrt` (racine), `x^n` (puissance n du nombre x), `abs` (valeur absolue), `exp` (exponentielle), `log` (logarithme népérien).
  – fonctions trigonométriques : `cos`, `sin`, `tan`, `acos`, `asin`, `atan` pour les fonctions inverses.
  – fonctions hyperboliques : `cosh`, `sinh`, `tanh` (pour le cosinus, le sinus et la tangente hyperboliques), `acosh`, `asinh`, `atanh` pour leurs inverses.
  – algèbre : `factorial(a)` calcule $a!$.
  – arrondi : Les fonctions `round`, `ceiling` et `floor` permettent de donner des arrondis : `round` arrondi à l'entier le plus proche, `ceiling` à l'entier supérieur, `trunc` à l'entier inférieur. La fonction `round(x,nb)` permet de garder nb chiffres après la virgule.

- probabilité : fonction de répartition, densité ou fonction de masse, et fonction quantile, par exemple pour la loi normale : `pnorm`, `dnorm`, `qnorm`.

• Fonctions usuelles pour deux nombres, ou deux vecteurs/matrices/tableaux de même taille (les calculs se font terme à terme dans ce cas, ce ne sont pas des calculs matriciels).
- `+`, `-`,`*`,`/`
- comparaison : `<`, `<=` (...)
- algèbre : `a%%b` donne le reste de la division de a par b, `choose(n,k)` calcule C_n^k.

• Fonctions spécifiques aux vecteurs (et séries temporelles) :
- dimension : `length`
- tri des données : `sort`.
- minimum/maximum : `max`, `min`, (maximum et minimum) `which.max`, `which.min` (à quel indice se situe le maximum/minimum), `cummin` (minimum sur les éléments déjà vus), `cummax` (maximum sur les éléments déjà vus).
- `sum` (somme de tous les éléments),`prod` (produitde tous les éléments), `cumsum` (sommes cumulées), `cumprod` (produit cumulé)
- statistique descriptive : `mean` (moyenne empirique), `var` (variance empirique corrigé), `sd` (écart-type empirique), `summary` (calcule moyenne, médiane, premier et troisième quartile, minimum et maximum). La fonction `table(v)` crée la table de contingence du vecteur v.

• Fonctions spécifiques aux matrices
- dimension : `nrow`, `ncol`, `dim`. La fonction `dim` s'utilise aussi pour les tableaux à plus de 2 dimensions.
- opération sur les lignes ou les colonnes d'une matrice ou d'un tableau de données : `rowSums` calule la somme de chaque ligne, `rowMeans` calcule la moyenne de chaque ligne, `colMeans` calcule la moyenne de chaque colonne, `colSums` calcule la somme de chaque colonne. Attention à ne pas oublier les majuscules !
- La fonction `apply(matrice, 1 pour lignes ou 2 pour colonnes, f)` permet d'appliquer une fonction aux colonnes ou aux lignes d'une matrice. Typiquemente, $f = mean$ ou toute autre fonction du même type ($f = sd$,...). Ces fonctions marchent de la même manière que `rowSums` ou `colMeans`.
- manipulation de matrices : `%*%` pour la multiplication matricielle, `solve` pour l'inverse, `t` pour la transposée.

• Fonctions spécifiques aux tableaux (à deux dimensions ou plus).
- dimension : `dim`
- Si on a un tableau de contingence de grande dimension, on veut souvent récupérer un tableau de dimension plus petite (donc plus maniable). Pour cela, on utilise : `apply(nom_Tableau, dimensions qu'on veut garder, sum )`

• Concaténation de vecteurs ou de matrices : `c`, `cbind`,`rbind`. Pour une chaîne de caractères, on utilise `paste`.

• La fonction `(outer(u,v,'f')` permet à partir de deux vecteur u de longueur m

et v de longueur n de créer la matrice

$$\begin{pmatrix} f(u(1),v(1)) & f(u(1),v(2)) & \dots & f(u(1),v(n)) \\ f(u(2),v(1)) & f(u(2),v(2)) & \dots & f(u(2),v(n)) \\ \dots & \dots & \dots & \dots \\ f(u(m),v(1)) & f(u(m),v(2)) & \dots & f(u(m),v(n)) \end{pmatrix}$$

f est une fonction de deux variables : $f = +$ ou $*$, par exemple.

3 Probabilités

3.1 Avec R

R connaît la plupart des lois de probabilités usuelles. On y accède en chargeant le package stats (`library(stats)`). Il peut :

- Simuler une variable suivant cette loi grâce à la commande `rloi(n, paramètres)`
- Donner la densité (pour une variable continue) ou les probabilités marginales (pour une loi discrète) : `dloi(x, paramètres)`
- Donner la fonction de répartition : `ploi(x,paramètres)`
- Donner la fonction quantile : `qloi(q,paramètres)`.

x peut être un réel, un vecteur ou une matrice, tout comme q (mais toutes les valeurs de q doivent être comprises entre 0 et 1). Les paramètres dépendent de la loi.

Loi usuelles :

- Loi normale : `norm` (paramètres : espérance et écart-type)
- loi exponentielle : `exp` (paramètre : λ)
- loi uniforme : `unif`(paramètres : a et b)
- loi Gamma : `gamma` (paramètres a et λ)
- loi binomiale : `binom` (paramètres n et p)
- loi de Poisson : `pois` (paramètre λ)
- loi du χ^2 : `chisq` (paramètre :nombre de degrés de liberté)
- loi de Student : `t` (paramètre : nombre de degrés de liberté)

Par exemple, la commande `dexp(2,1)` donne la densité d'une loi exponentielle de paramètre 1 au point 2. La fonction `sample` permet de générer aléatoirement des entiers :

- `sample(v)` génère une permutation aléatoire du vecteur v.
- `sample(v,n)` choisit au hasard n nombres parmis ceux du vecteur v (tirage sans remise).
- `sample(m,n)` choisit au hasard n nombres parmis les m premiers entiers (tirage sans remise), (c'est un raccourci pour `sample(seq(1,m),n)`).

Le logiciel `R` a de plus beaucoup de fonctions qui permettent d'étudier les données : par exemple `mean`, `var` (qui calcule la variance corrigée, ce qui est légèrement différent de la variance empirique), `sd` (écart-type), `cov`, `cor`, `median`. La fonction `summary` permet d'afficher un "résumé" du vecteur : minimum, maximum, premier et troisième quartile, médiane et moyenne.

3.2 Avec Python

Tout comme `R`, Python connaît la plupart des lois de probabilités usuelles. On y accède avec la commande `scipy import stats`, puis en important chacune des lois nécessaires : `scipy.stats import norm` pour la loi normale, par exemple.

- rvs(size=) : générateur de variables aléatoires.
- pdf : densité (probability density function) pour les variables continues
- pmf : fonction de masse (probability mass function) pour les variables discrètes
- cdf : fonction de répartition (cumulative distribution function)
- ppf : fonction quantile (percentile point function)
- stats : calcule la moyenne, la variance, kurtosis et skew.
- moment : calcule les moments

Lois usuelles :

- loi normale : norm (paramètre espérance et écart-type)
- loi uniforme : uniform (paramètre `loc=a`, `scale=b-a`)
- loi exponentielle : expon (paramètre `scale=1/lambda`)
- loi gamma : gamma (paramètre `a,scale=1/lambda`)
- loi du $\chi 2$: chi2 (paramètre degré de liberté)
- loi de Fisher : t (paramètre t)
- loi binomiale : binomial (paramètre n, p)
- loi de Poisson : poisson (paramètre λ)

Toutes les lois peuvent avoir pour paramètre `loc` et verb'scale'. Le paramètre `loc` translate la loi, le paramètre `scale` la dilate : si Z est la loi standard (`scale`=1 et `loc`=0), alors si X suit une loi de paramètres `loc` et `scale`, $X = \text{scale} \times Z + \text{loc}$. Cela suffit pour transformer la loi uniforme et la loi exponentielle, néanmoins, ce ne sont pas du tout les paramètres standard des lois. Pour d'autres lois (Gamma, mais aussi toutes les lois discrètes) on a besoin d'un paramètre supplémentaire (noté `a` ici).

La commande `stats.norm(1,2).pdf(0)` calcule la densité d'une loi normale $\mathcal{N}(1, 2^2)$ au point 0. La commande `stats.norm(1,2).mean()` donne l'espérance d'une loi normale $\mathcal{N}(1, 2^2)$.

4 Graphiques

Il existe différents type de graphiques sous `R` : les graphes classiques, les histogrammes, les boîtes à moustaches ... .

4.1 Graphe classique

Pour les graphes les plus simples, on utilise les trois commandes `plot(x,y,options)`, `points(x,y,options)` et `lines(x,y,options)`. Ces trois commandes permettent de tracer y en fonction de x.
On utilise toujours `plot` pour commencer un graphique, `points` permet de superposer un nuage de points, `lines` de superposer une fonction. Les options les plus utilisées sont :

- `type='p'` par défaut (ne trace que des points), `='l'` si on veut tracer une ligne, `='s` pour une fonction en escalier.
- `col ='black'` par défaut (couleurs possibles : red, blue, green, yellow, orange, magenta, pink, ...)
- `pch` (c'est le style de point utilisé, il peut être égal à 1,2,3,...)
- `lty` (c'est le style de ligne utilisé, il peut être égal à 1,2,3,...)
- `ylim=c(a,b)` (option très utile si on veut superposer plusieurs graphiques. Par défaut, `R` affiche les y entre min(y) et max(y), on peut forcer l'affichage d'autres valeurs). Ne fonctionne qu'avec la commande `plot`.
- `xlim=c(a,b)` (même chose que ylim, mais pour x. Peut-être moins utile).
- `main=` donne un titre au graphe.
- `xlab=`, `ylab=` change le nom des axes.

Pour superposer un deuxième graphique, on utilise soit `points` (pour tracer des points) soit `lines` (pour tracer des lignes), soit utiliser l'option `add=TRUE` (utilisable avec tous les types de graphes). La fonction `abline(a=,b=,h=,v=)` permet de tracer une droite, soit en précisant l'abscisse à l'origine (a) et la pente (b), soit de tracer une ligne horizontale d'ordonnée h, soit de tracer une ligne verticale d'abscisse v.

On peut aussi tracer plusieurs graphiques côte à côte : pour cela on commence par partager la fenêtre des graphes en plusieurs cases grâce à la commande `split.screen(c(i,j))` (partage en i lignes et j colonnes). Pour tracer un graphe dans une case, on peut donner le numéro de la case : `screen(k)` et on trace ensuite le graphique grâce aux commandes habituelles. Il faut fermer les graphes à la fin, sinon tout se superpose. Pour cela, on utilise la commande `close.screen(all=TRUE)`.

4.2 Données quantitatives

- La fonction `hist` trace l'histogramme d'un vecteur : elle compte (et représente) combien on a d'observations dans chaque catégorie. La commande `hist(x,breaks,freq=TRUE)` donne le nombre de variables dans chaque intervalle, et la commande `hist(x,breaks,freq=FALSE)` donne la proportion de variables dans chaque intervalle. Pour changer la couleur, on peut utiliser `border=`.
  Attention : sauf dans le cas d'une série temporelle, il faut toujours préciser les deux vecteurs !.
- La fonction `density(X)` estime la densité des données.
- La fonction `boxplot` trace la boîte à moustache d'un vecteur. Elle permet de tracer soit la boîte à moustache d'un seul vecteur, soit la boîte à moustache de plusieurs vecteurs côte à côte.
- La fonction `qqnorm` trace les quantiles empiriques d'un vecteur X par rapport aux quantiles théoriques de la loi normale $\mathcal{N}(0,1)$.
- La fonction `qqplot` permet de tracer le graphe quantile/quantile de deux vecteurs (X,Y). Les deux vecteurs n'ont pas besoin d'être de même longueur. Avec `qqline`, on peut ajouter la droite qui passe par les quartiles $Q1$ et $Q3$.

Le logiciel `R` permet de tracer simplement pas mal de graphiques. En particulier, si on a un tableau de données appelé `Donnees` avec trois colonnes `longueur`, `largeur`, `espece`,

on peut tracer les boîtes des longueurs pour chaque espèce : `boxplot(longueur~espece,data=Donnees)` ou la largeur en fonction de la longueur pour l'espèce 1 :
`plot(largeur~longueur, data=Donnees, subset=espece==1)`.

4.3 Données unidimensionnelles et qualitatives

Pour tracer les graphiques, il faut d'abord avoir les données sous forme synthétiques (combien de fois on a observé la caractéristique A, combien de fois on a observé la caractéristique B,.... Il faut donc un tableau de la forme :

classe	A	B	...
nombre d'observations	5	7	...

Si on a des informations sous la forme :

indidividu	classe
1	A
2	A
3	B
4	A
...	...

il faut commencer par compter les données dans chaque classe, ce qu'on fait grâce à la commande `table`. Si les données sont multidimensionnelles, la commande `table` crée une *table de contingence*, c'est à dire qu'elle compte le nombre de fois où chaque couple (ou triplet, ou quadruplet) apparaît.

Pour des données unidimensionnelles :

- Camembert : La fonction `pie(X,labels=noms, main='nom de graphique')` permet de tracer des camemberts. Attention : la représentation en camembert est moins précise que les autres représentations.
- La fonction `barplot` permet de tracer le diagramme en bâtons. Pour des données unidimensionnelles, on utilise `barplot(M,main='nom du graphique')`. Si on veut comparer plusieurs vecteurs, c'est possible s'ils sont rangés par ligne : `barplot(M,beside=TRUE,main='`
- La fonction `dotchart` trace à peu près le même graphique, mais horizontallement et pas verticalement. `dotchart(M, main='nom du graphique')`

Pour des données multidimensionnelles, on peut utiliser `mosaicplot`.

5 Test

Le logiciel `R` est un logiciel de statistiques. Il permet de faire énormément de tests et beaucoup sont déjà implémentés. Beaucoup de tests sont aussi implémentés dans Python.

5.1 Avec R

Un certain nombre de tests sont déjà préimplémentés dans `R`. À part les deux premiers, `binom.test` et `poisson.test`, il faut rentrer le vecteur entier des observations (alors que parfois, on ne dispose que de la moyenne et de la variance empiriques). Le début du test (modélisation, choix des hypothèses, choix de la statistique de test et forme de la zone de rejet) se fait toujours à la main. Les tests de `R` calculent la p-valeur, pas la zone de rejet, ni la taille, ni la puissance. Certains tests (test sur la moyenne à variance connue, par exemple), bien que très utilisés, ne sont pas implémentés.

Même si on parle de test, la plupart de ces fonctions permettent aussi de calculer des intervalles de confiance. C'est particulièrement appréciable pour les lois binomiales et les lois de Poisson, car les fonctions `binom.test` et `poisson.test` sont capables de calculer ces intervalles de façon exacte, sans passer par l'approximation par une loi normale.

Test sur une proportion, loi binomiale

```
binom.test(15, 20, 0.5,alternative='greater',conf.level=0.95)
```

Pour faire les tests sur les proportions quand n est petit (calculs exacts avec la loi binomiale). Prend en entrée S_n, n, p_0, l'alternative H_1 ($p \neq p_0$, $p > p_0$, $p < p_0$) notée `two.sided`, `greater` et `less`, et le niveau de confiance $1 - \alpha$.
Le test calcule la p-valeur, ainsi qu'un intervalle de confiance de niveau $1 - \alpha$. Il estime aussi p.

Test sur une proportion, loi de Poisson

```
poisson.test(3,r=5,alternative='less',conf.level=0.95)
```

Pour faire les tests sur une proportion quand l'approximation par une loi de Poisson est valide. Prend en entrée S_n, λ_0, l'alternative H_1 ($\lambda \neq \lambda_0$, $\lambda > \lambda_0$, $\lambda < \lambda_0$) noté `two.sided`, `greater` et `less`,et le niveau de confiance $1 - \alpha$.
Le test calcule la p-valeur, ainsi qu'un intervalle de confiance de niveau $1 - \alpha$. Il estime aussi p. Ne pas oublier le `r=`, sinon vous allez avoir des problèmes.

Test sur une proportion, approximation loi normale

```
prop.test(x,n,p=0.5,alternative='greater',conf.level=0.95)
```

Calcule le carré de la statistique de test, la p-valeur et l'intervalle de confiance par approximation par la loi normale.

Test sur la moyenne, loi normale

```
t.test(x,mu=0,alternative='less',conf.level=0.95)
```

Calcule la statistique de test, la p-valeur et l'intervalle de confiance grâce à la loi de Student.

Test sur la médiane, test du signe Le test du signe repose sur la loi binomiale : utiliser `binom.test`.

Test sur la médiane, signe et rang

```
wilcox.test(x,alternative='two.sided',mu=0,conf.int=TRUE,conf.level=0.95)
```

Calcule la statistique de test et la p-valeur. Si `conf.level=TRUE`, `R` calcule aussi un intervalle de confiance pour la médiane.

Test d'adéquation à une loi discrète

```
\verb+chisq.test(x, p=p)
```

Il faut regrouper les données dans les classes et calculer la probabilité d'être dans chaque classe (ce que donne le vecteur p). Attention à bien regrouper les classes pour que $np \geq 5$! La somme des probabilités doit être égale à 1. Le logiciel `R` calcule la statistique de test et la p-valeur.

Test d'adéquation à une loi continue

```
ks.test(x,pnorm,0,1)
```

Le test de Kolmogorov-Smirnov est très rapide à faire. Il suffit de rentrer le vecteur x, et H_0 (la loi que l'on considère et ses paramètres). `R` calcule la statistique de test et la p-valeur.

Test d'égalité de deux moyennes, loi normale

```
t.test(x,y,alternative='two.sided',var.equal=FALSE,conf.level=0.95)
```

Fait le test de Student d'égalité des espérances pour des variables normales. Renvoie la p-valeur et l'intervalle de confiance de la différence des espérances.

Test d'égalité de deux proportions, approximation

```
prop.test(x,y,alternative='greater',conf.level=0.95).
```

Fait le test asymptotique d'égalité des proportions par approximation par la loi normale. Renvoie la p-valeur et l'intervalle de confiance de la différence des proportions.

Test d'égalité des variances, loi normale

```
var.test(x,y,alternative='greater',conf.level=0.95)
```

Fait le test de Fisher d'égalité des variances. Renvoie la p-valeur et l'intervalle de confiance du rapport des variances.

Test d'égalité des médianes

```
wilcox.test(x,y,alternative='greater',conf.int=TRUE,conf.level=0.95)
```

Calcule la statistique de test et la p-valeur. Si `conf.int=TRUE`, calcule aussi un intervalle de confiance pour la différence des médianes.

Test d'homogénéité, loi continue

```
ks.test(x,y,alternative='two.sided')
```

Calcule la statistique de test et la p-valeur pour le test de Kolmogorov-Smirnov pour deux échantillons.

Test d'homogénéité, loi discrète

`\verb'chisq.test(x,y)'`

Calcule la statistique de test et la p-valeur pour le test du χ^2 d'homogénéité.

Remarque 1. *Les commandes* `prop.test`*,* `t.test`*,* `wilcoxon.test`*,* `ks.test` *et* `chisq.test` *sont les mêmes que l'on veuille faire un test sur un seul échantillon ou sur deux échantillons indépendants. Il faut donc faire très attention aux commandes que l'on donne en argument. Heureusement, le logiciel* `R` *donne souvent le nom du test qu'il fait, et reformule l'hypothèse alternative H_1, ce qui permet de vérifier qu'on ne s'est pas trompé.*

5.2 Avec Python

Test sur une probportion, loi binomiale
`stats.binom_test(X,n,p,alternative='greater')`. Pour faire un test sur les proportions. Prend en entrée S_n, n, p_0, l'alternative H_1 ($p \neq p_0$, $p > p_0$ ou $p < p_0$) notées `two-sided`, `greater` et `less`. Le test calcule la p-valeur.

Test sur la moyenne, loi normale
`stats.ttest_1samp(X,m)`. Test bilatéral sur la moyenne d'une loi normale. Prend en entrée l'échantillon X, et l'espérance m_0. Renvoie la p-valeur et la statistique de test.

Test sur la médiane, test du signe : utiliser `stats.binom_test`.

Test sur la médiane, test du rang
`stats.wilcoxon(X,Y,zero_method='zsplit',alternative='two-sided')` X et Y sont les deux vecteurs d'observations, ensuite on donne la méthode pour traiter les valeurs égales (`zsplit` sépare les 0 en deux groupes égaux, un positif, un négatif), et l'alternative H_1. Renvoie la statistique de test et la p-valeur.

test du chi2 d'adéquation
`stats.chisquare(X,E)` avec X les observations et E les valeurs attendues. Renvoie la statistique de test et la p-valeur.

test d'adéquation à une loi continue
`stats.kstest(X,fdr)` avec `fdr` une fonction de répartition (par exemple, `stats.norm(1,2).cdf`). Renvoie la statistique de test et la p-valeur. Et `stats.anderson(X,famille_loi)` réalise un test d'adéquation à une famille de loi (la famille de loi peut être `'norm'` ou `'expon'`). Il renvoie la statistique de test, et une liste de quantiles.

Test d'égalité de deux moyennes, loi normale
`stats.ttest_ind(X,Y,equal_var=FALSE)`. Test de Student (ou test de Welch quand `equal_var=FALSE`) d'égalité des variances. Renvoie la p-valeur et la statistique de test.

Test d'homogénéité, loi continue
`stats.kstest_2samp(X,Y,alternative)` test de Kolmogorov-Smirnov à deux échantillons. Renvoie la p-valeur et la statistique de test.

6 Régression linéaire

La commande `lm` permet de faire une régression linéaire multiple. Pour faire une régression linéaire de y en fonction de x, il faut taper : `lm(y~x)`. Le logiciel `R` estimera a et b.

Attention :

- Si on pense que $b = 0$, il faut alors faire `lm(y~0+x)`.
- Si $Y = a_0 + a_1X + a_2X^2$, il faudra taper : `lm(y~x+I(x^2))`.

La commande `lm` calcule entre autre :

- les coefficients (`coefficients`)
- les résidus (`residuals`)
- le critère de détermination et le critère de détermination ajustée.

De plus, `R` fait un test de Student pour savoir si chaque coefficient est nul et un test de Fisher pour tester l'ensemble des coefficients.

On peut utiliser `summary` pour obtenir toutes les informations sur la régression. Cette commande affiche

- les quartiles des résidus.
- la valeur des coefficients, leur écart-type estimé, et le résultat du test "Ce coefficient est nul".
- les coefficients de détermination et de détermination ajustée.

7 Un peu de programmation

Le logiciel `R` n'aime pas du tout les boucles. À chaque fois que cela est possible, il faut passer par les matrices et les opérations sur les vecteurs. (Une boucle pour 5 ou 10 opérations n'est bien sûr pas génante, une boucle for de 1000 itérations peut énormément ralentir le programme).

- Structure d'une boucle for :

```
for (k in 1:n)
{
lignes codes
}
```

  Les accolades ne sont pas nécessaires s'il n'y a qu'une seule ligne de code.
- Conditions

```
if (condition)
{
lignes codes
}
else
```

```
{
lignes codes
}
```
On n'est pas obligé de mettre un `else`. `R` ne sait comparer pas utiliser la condition `if` sur un vecteur. La structure `if/then/else` s'utilise quand on a qu'une seule variable à tester. Pour faire des opérations conditionnelles sur un vecteur, on peut simplement utiliser les fonctions `<` ou `==`.

- Boucles while
```
while (condition)
{
lignes commandes
}
```
- Fonctions
```
nomfonction <- function (nomvariables)
 {
une ligne de code
}
```
ou si la fonction est plus compliquée :
```
nomfonction<- function (nomvariables)
{
lignes code
return(expression)
}
```
Dans ce dernier cas, le résultat de la fonction ne s'affichera pas automatiquement.