# Tous les tests sont bilatéraux !!!

## 1. Comparaison de moyennes - Echantillons indépendants - 1

Ecrire une fonction qui implémente la comparaison de moyennes calculées sur deux vecteurs x et y. *On fait l'hypothèse que les 2 variances inconnues sont égales*.

**Entrée** : x et y vecteurs

**Sortie** : t de Student, ddl et p-value

**Référence** :

- **Comp_Pop_Tests_Parametriques.pdf** (section 1.2.2, équation 1.2)
- http://www.cons-dev.org/elearning/stat/parametrique/5-2/5-2.html
- Voir **pt** pour le calcul de la p-value avec la loi de Student, attention on a un **test bilatéral**.
- Voici l'en-tête de la fonction : **comp_moyenne <- function(v1, v2)**
- La fonction équivalente sous R (package « stats » chargé par défaut) est **t.test()** (attention aux options).

Voici un exemple d'exécution du programme :

```
> #deux vecteurs
> v1 <- c(159,170,190,167,160,155)
> v2 <- c(185,168,179,176)
> #appel de votre fonction programmée ci-dessus
> z <- comp_moyenne(v1,v2)
> print(z)
$t
[1] -1.449278

$ddl
[1] 8

$pvalue
[1] 0.1853
```

## 2. Comparaison de moyennes – Echantillons indépendants - 2

Ecrire une fonction qui compare les moyennes de deux échantillons indépendants. Les données se présentent sous la forme de deux vecteurs : x le vecteur des valeurs ; y un indicateur du groupe d'appartenance, il doit être forcément un factor.

Voici un exemple pour préciser le mode d'utilisation de la fonction :

```
> #données pour tester la fonction
> x <- c(185,159,170,168,190,167,160,179,176,155)
> y <- as.factor(c("h","f","f","h","f","f","f","h","h","f"))
> #tester la fonction
> print(test_student(x,y))
$t
        f
-1.449278

$ddl
f
8

$pvalue
      f
0.1853
```

**Entrée** : x vecteur de numérique, y un factor

**Sortie** : statistique de test, ddl et p-value

**Indications :**

- Voici la signature de la fonction : **test_student(x,y)**
- En entrée de la fonction, vous devez vérifier que y est bien un factor et qu'il comporte deux modalités. Dans le cas contraire, vous devez stopper les calculs et renvoyer NULL.
- **is.factor()** permet de savoir si un vecteur est bien de type factor ou non.
- 2 stratégies possibles **:**
    1. Scinder x en deux vecteurs en s'appuyant sur les valeurs de y. Appeler alors la fonction de l'exercice précédent.
    2. Effectuer directement les calculs. Voir alors du côté de **tapply()** pour les calculs conditionnels (voir l'aide de R).

## 3. Comparaison de moyennes (échantillons appariés)

Ecrire une fonction qui implémente la comparaison de moyennes calculées sur deux vecteurs. *Attention, les deux vecteurs doivent être de même longueur. Dans le cas contraire, la fonction renvoie NULL*.

**Entrée** : x et y vecteurs

**Sortie** : t de Student, ddl et p-value

**Référence** :

- **Comp_Pop_Tests_Parametriques.pdf** (section 4.2.1, équation 4.1)
- http://www.cons-dev.org/elearning/stat/parametrique/5-2/5-2.html
- En-tête de la fonction **comp_moyenne_pairwise <- function(v1, v2)**
- Pour la fonction native R, voir de nouveau **t.test()** avec le paramétrage idoine.

```
> #deux vecteurs exemples
> v1 <- c(1.2,3.4,5.6,7.8,1.3)
> v2 <- c(3.6,1.7,5.6,9.0,1.5)
> #appel de votre fonction programmée ci-dessus
> z <- comp_moyenne_pairwise(v1,v2)
> print(z)
$t
[1] -0.6176471

$ddl
[1] 4

$pvalue
[1] 0.5702361
```

## 4. Mode d'une distribution de fréquences

Ecrire une fonction qui prend en entrée un vecteur de valeurs entières (un facteur). Il doit détecter la modalité la plus fréquente, renvoyer l'étiquette associée et la fréquence relative associée. Attention, votre fonction doit vérifier que l'on a bien affaire à un type factor en entrée. Dans le cas contraire, la valeur NULL est renvoyée.

**Entrée** : x vecteur, censé être un factor

**Sortie** : Une liste avec la modalité associée au mode et sa fréquence absolue

- **freqmax <- function(v)**
- **table(.)** vous permet de calculer une distribution de fréquence (renvoie un vecteur), **max(.)** renvoie la valeur maximale dans un vecteur, **which.max(.)** renvoie l'index correspondant à la valeur maximale.

```
> #créer le vecteur de type factor
> v <- c(1,2,1,1,1,2,1)
> sexe <- factor(v)
> levels(sexe) <- c("homme","femme")
> print(sexe)
[1] homme femme homme homme homme femme homme
Levels: homme femme
>
> #appliquer la fonction
> print(freqmax(sexe))
$modalite
[1] "homme"

$valeur
[1] 5
```

## 5. Test d'indépendance du KHI-2

Ecrire une fonction qui réalise le test d'indépendance du khi-2.

**Entrée** : x et y, deux vecteurs, de type factor

**Sortie** : statistique de test, degrés de liberté, p-value

**Indications** :

- Vous devez vérifier en entrée si les deux vecteurs sont de même longueur et s'ils sont tous deux de type factor. Dans le cas contraire, votre fonction renvoie NULL.
- Vous devez programmer explicitement les calculs et non pas vous appuyer sur chisq.test(). Utilisez **table(.)** pour calculer le tableau des effectifs observés, croisement de x et y.
- Pour calculer le tableau sous indépendance, vous pouvez vous appuyer sur une double boucle, vous pouvez aussi vous appuyer sur un produit matriciel. A vous de voir.
- En utilisant les notations du document (**Dépendance_Variables_Qualitatives.pdf** ; section 2.1), voici la formule du Khi-2 :

$$\chi^2 = \sum_{l=1}^{L}\sum_{c=1}^{C}\frac{(n_{lc} - e_{lc})^2}{e_{lc}}$$

Où $e_{lc}$ est l'effectif théorique sous hypothèse d'indépendance, obtenue avec :

$$e_{lc} = \frac{n_{l.} \times n_{.c}}{n}$$

- **`test.khi2 <- function(y,x)`**

Voici un exemple d'exécution :

```
> #deux vecteurs transformés en facteurs
> x <- c(1,2,2,3,2,2,1,1,2,2,1,1,1,1,2)
> x <- factor(x)
> y <- c(1,2,1,2,1,2,1,1,2,1,2,1,1,2,2)
> y <- factor(y)
>
> #notre fonction
> print(test.khi2(x,y))
$chi2
[1] 2.372449

$ddl
[1] 2

$pvalue
[1] 0.305372
```

- Après coup, comparez vos résultats avec ceux de **chisq.test()**, voir dans l'aide pour un paramétrage adéquat.

## 6. Calcul des résidus ajustés d'un tableau de contingence

Ecrire une fonction qui prend en entrée une matrice représentant un tableau de contingence, elle doit renvoyer en sortie la matrice des résidus ajustés du tableau.

**Entrée** : m (matrice) représentant un tableau de contingence

**Sortie** : une matrice contenant les résidus ajustés

**Indications** : Pour le principe, voir **Dépendance_Variables_Qualitatives.pdf** (Section 2.3.3) ; voici la formule à utiliser :

$$r.adj_{lc} = \frac{n_{lc} - e_{lc}}{\sqrt{e_{lc}\left(1 - \frac{n_{l.}}{n}\right)\left(1 - \frac{n_{.c}}{n}\right)}}$$

Voici un exemple d'exécution

```
> m <- matrix(c(33,94,21,63,452,115,4,13,5,8,154,38),nrow=3,ncol=4)
> print(m)
     [,1] [,2] [,3] [,4]
[1,]   33   63    4    8
[2,]   94  452   13  154
[3,]   21  115    5   38
>
> print(residus.ajustes(m))
          [,1]       [,2]       [,3]       [,4]
[1,]  4.882166 -1.0635676  1.1280048 -3.4640537
[2,] -2.268647  0.4068642 -1.2800871  1.9923202
[3,] -1.275798  0.3810093  0.5972356  0.4536956
```

## 7. Calcul des profils d'un tableau de contingence

Ecrire deux fonctions [**profil.colonne, profil.ligne**] qui prend en entrée une matrice représentant un tableau de contingence, elles doivent respectivement renvoyer en sortie la même matrice sous forme de profil colonne ou ligne (pourcentages colonne ou ligne).

**Entrée** : m (matrice)

**Sortie** : une matrice profil ligne (1) ou colonne (2)

**Indications** : Regardez du côté de la fonction **apply()** pour éviter d'avoir à faire des boucles.

Voici un exemple d'exécution :

```
> m <- matrix(c(33,94,21,63,452,115,4,13,5,8,154,38),nrow=3,ncol=4)
> print(m)
     [,1] [,2] [,3] [,4]
[1,]   33   63    4    8
[2,]   94  452   13  154
[3,]   21  115    5   38
>
> print(profil.colonne(m))
          [,1]      [,2]      [,3] [,4]
[1,] 0.2229730 0.1000000 0.1818182 0.04
[2,] 0.6351351 0.7174603 0.5909091 0.77
[3,] 0.1418919 0.1825397 0.2272727 0.19
>
> print(profil.ligne(m))
          [,1]      [,2]       [,3]       [,4]
[1,] 0.3055556 0.5833333 0.03703704 0.07407407
[2,] 0.1318373 0.6339411 0.01823282 0.21598878
[3,] 0.1173184 0.6424581 0.02793296 0.21229050
```

## 8. Analyse de variance à un facteur

Ecrire le test de comparaison de K moyennes (analyse de variance).

La fonction prend en entrée un vecteur X de valeurs (mesures), il correspond à la variable d'intérêt, et un vecteur de code Y (facteur), indiquant la population d'appartenance.

$$\sum_{i=1}^{n}(x_i-\bar{x})^2=\sum_{k=1}^{K}n_k\times(\bar{x}_k-\bar{x})^2+\sum_{k=1}^{K}\sum_{i:y_i=k}(x_i-\bar{x}_k)^2$$

$$SCT=SCE+SCR$$

$$F=\frac{SCE/K-1}{SCR/n-K}\equiv F(K-1,n-K)$$

La fonction doit renvoyer la statistique F, les deux degrés de liberté et la p-value du test.

Documentation : Voir aussi « **Comp_Pop_Tests_Paramétriques.pdf** », section 1.3.

**`test_anova_tapply <- function(x,y)`**

**Entrée** : x vecteur de numérique, y un factor

**Sortie** : statistique de test F, les deux degrés de liberté, la p-value

- En entrée de la fonction, vous devez vérifier que y est bien un factor et qu'il comporte ***au moins*** deux modalités. Dans le cas contraire, vous devez stopper les calculs et renvoyer NULL.
- Dans le cas où le nombre de modalités de y est deux, vous devriez obtenir les mêmes résultats que ceux du test de Student (avec le carré de la statistique de test).
- **tapply()** devrait vous être très utile.
- Comparez-vous avec la fonction **aov()** de R (cf. l'aide pour le paramétrage).

Voici un exemple d'exécution :

```
> x <- c(185,159,170,168,190,167,160,179,176,155)
> y <- as.factor(c("h","f","f","h","f","f","f","h","h","f"))
> print(test_anova_tapply(x,y))
$F.Anova
[1] 2.100406

$ddl1
[1] 1

$ddl2
[1] 8

$p.value
[1] 0.1853
```

Autre exemple :

```
> #autre essai
> data(iris)
> print(test_anova_tapply(iris$Petal.Width,iris$Species))
$F.Anova
[1] 960.0071

$ddl1
[1] 2

$ddl2
[1] 147

$p.value
[1] 4.169446e-85
```

## 9. Test de Wilcoxon-Mann-Whitney (pour échantillons indépendants)

Ecrire une fonction qui implémente la comparaison de deux populations selon la méthode de Wilcoxon. Nous utilisons l'approximation normale. Nous utilisons les rangs moyens en cas d'ex-æquo, mais nous ne répercutons pas les modifications sur le calcul de la variance.

**Entrée** : A et B vecteurs

**Sortie** : Statistique W, la statistique Z (centrée et réduite de W) et la p-value

En-tête de la fonction : **mann_whitney <- function(v1,v2)**

**Référence** :

***Description de la technique***.

Soient deux échantillons A et B, de tailles respectives $n_A$ et $n_B$, on supposera afin de faciliter la compréhension que $n_A \leq n_B$ (dans la pratique, il faudra que la fonction détermine comme référence celle qui contient le moins d'individu). On note $W_A$ la somme des rangs de A, sous $H_0$, la quantité $W_A$ suit asymptotiquement (pour $n_A + n_B \geq 30$ en toute rigueur, en réalité le test est assez robuste) une loi normale de paramètres

$$E(W_A) = \frac{n_A(n_A + n_B + 1)}{2}$$
$$V(W_A) = \frac{n_A n_B(n_A + n_B + 1)}{12}$$

Exemple : comparaison de taille d'individus. Voici la taille, en cm, de deux d'individus. Les rangs sont déduits sur la 3ème ligne.

A <- c(185,168,179,176)

B <- c(159,170,190,167,160,155)

| Appartenance | A | A | A | A | B | B | B | B | B | B |
|---|---|---|---|---|---|---|---|---|---|---|
| Valeur | 185 | 168 | 179 | 176 | 159 | 170 | 190 | 167 | 160 | 155 |
| **Rang** | **9** | **5** | **8** | **7** | **2** | **6** | **10** | **4** | **3** | **1** |

WA = 9 + 5 + 8 + 7 = 29

- Voir aussi : **Comp_Pop_Tests_NonParametriques.pdf** (section 2.1, et en particulier 2.1.6)
- La procédure **rank(.)** fournit le rang
- **pnorm()** permet de calculer la p-value pour la loi normale.
- Voir **wilcox.test()** pour comparer vos résultats. Attention, nous utilisons l'approximation normale (cf. l'option « exact ») sans la correction de continuité (cf. « correct ») (Remarque : Mann et Whitney est une variante de Wilcoxon, la stat. de test n'est pas la même, mais l'approximation normale et donc la p-value devraient être identiques).

```
> #exemple de données
> A <- c(185,168,179,176)
> B <- c(159,170,190,167,160,155)
>
> #appel de la fonction et affichage
> print(mann_whitney(A,B))
$Stat
[1] 29

$z
[1] 1.492405

$p
[1] 0.135593
```

## 10. Test de Kruskal-Wallis

Ecrire le test non paramétrique de comparaison de populations de Kruskal-Wallis. On fait l'hypothèse qu'il n'y a pas d'ex-aequo dans les valeurs.

La description de la méthode est disponible sur Wikipédia :

https://en.wikipedia.org/wiki/Kruskal%E2%80%93Wallis_one-way_analysis_of_variance

Voir aussi : **Comp_Pop_Tests_NonParametriques.pdf** (section 3.1)

```
kwallis <- function(x,y)
```

**Entrée** : x vecteur de numérique, y un factor ; ; les deux de même longueur

**Sortie** : statistique de test, degrés de liberté et p-value

- En entrée de la fonction, vous devez vérifier que y est bien un factor, et que le nombre de modalités est supérieur ou égale à 2. Dans le cas contraire, vous devez stopper les calculs et renvoyer NULL.

```
> x <- c(185,159,170,168,190,167,160,179,176,155)
> y <- as.factor(c("h","f","f","h","f","f","f","h","h","f"))
> print(kwallis(x,y))
$K
[1] 2.227273

$df
[1] 1

$p.value
[1] 0.135593
```

- Comparez vos sorties avec ceux de **kruskal.test()**

## 11. Test de Van der Waerden

Le test de Van der Waerden est une variante du test de Kruskal-Wallis où nous utilisons les « scores normaux » à la place des rangs bruts. La statistique de test suit une loi du KHI-2, le nombre de degrés de liberté est égal à : nombre de groupes – 1.

**Entrée** : x vecteur de numérique, y un factor ; les deux de même longueur

**Sortie** : statistique de test, degrés de liberté et p-value

- En entrée de la fonction, vous devez vérifier que y est bien un factor, et que le nombre de modalités est supérieur ou égale à 2. Dans le cas contraire, stopper les calculs et renvoyer NULL.
- Plusieurs étapes sont nécessaires (cf. **Comp_Pop_Tests_NonParametriques.pdf**) :
    - Transformez les données brutes (x) en rangs, puis en scores ($s_i$) en utilisant la fonction inverse de la loi normale centrée et réduite ($\Phi^{-1} = \text{qnorm}$) (section 5.4.3, équation 5.7)
    - Calculez en suite la statistique de test C, les degrés de liberté, la p-value (section 5.3, $S_k$ est la somme des scores du groupe n°k, $n_k$ est l'effectif du groupe n°k)

```
> x <- c(185,159,170,168,190,167,160,179,176,155)
> y <- as.factor(c("h","f","f","h","f","f","f","h","h","f"))
> print(test_vdw(x,y))
$statistic
[1] 1.842404

$ddl
[1] 1

$p_value
[1] 0.1746694
```

    - Comparez vos résultats avec la fonction **VanWaerdenTest()** du package « DescTools ».

## 12. Test de Levene

Le test de Levene est un test de comparaison de variances pour échantillons indépendants c.-à-d. il cherche à déterminer si les variances dans les sous-populations sont identiques ou non.

La fonction prend en entrée un vecteur X de valeurs (mesures), il correspond à la variable d'intérêt, et un vecteur de code Y (facteur), indiquant la population d'appartenance. Elle (la fonction) doit renvoyer la statistique F, les deux degrés de liberté et la p-value du test.

Voir aussi : **Comp_Pop_Tests_Parametriques.pdf** (section 2.5), lisez bien le paragraphe situé en dessous de l'équation 2.4. On doit pouvoir l'exploiter pour simplifier notre code.

Signature de la fonction : `my_levene <- function(x,y)`

**Entrée** : x vecteur de numérique, y un factor

**Sortie** : statistique de test, degré de liberté et p-value

Exemple :

```
> #données iris
> data(iris)
>
> #essai
> my_levene(iris$Petal.Width,iris$Species)
$F.Anova
[1] 19.65174

$ddl1
[1] 2

$ddl2
[1] 147

$p.value
[1] 2.732636e-08
```

Vous pouvez aussi comparer vos résultats avec ceux de la librairie « car » (cf. ci-dessous).

```
> #vérif. package "car"
> library(car)
>
> #test de Levene
> leveneTest(iris$Petal.Width,iris$Species,center="mean")
Levene's Test for Homogeneity of Variance (center = "mean")
       Df F value    Pr(>F)
group   2  19.652 2.733e-08 ***
      147
---
Signif. codes:
0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

## 13. Combat naval (pour s'amuser un peu, y a pas que les stats dans la vie…)

On souhaite programmer un jeu de combat naval très simplifié :

1. L'ordinateur place aléatoirement un bateau sur une case dans un tableau (7 x 7).
2. L'utilisateur doit deviner son emplacement en jouant au maximum 7 coups.
3. Pour chaque coup, il doit indiquer la case visée en spécifiant le n° de ligne et de colonne.
4. L'ordinateur doit répondre en indiquant « touché » si l'utilisateur a visé juste, « à l'eau » s'il a visé à côté. Dans ce dernier cas, l'ordinateur doit afficher la grille avec les coups déjà joués par l'utilisateur. Il doit aussi indiquer la direction de l'emplacement du bateau par rapport au dernier coup joué : nord, nord-est, est, sud-est, sud, sud-ouest, ouest, nord-ouest.
5. Le jeu se termine lorsque (a) l'utilisateur a visé juste, ou (b) il a dépassé le nombre de coups autorisé.

Remarques :

- Il faudra utiliser **scan()** pour la saisie des coordonnées (ligne, colonne) des missiles envoyés par l'utilisateur.
- Il y a plusieurs manières de générer une valeur entière aléatoire située entre 2 bornes, l'utilisation de **sample()** est une piste possible.

Voici (ci-dessous) une copie d'écran d'une étape du jeu.

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,] "."  "."  "."  "."  "."  "."  "."
[2,] "."  "."  "."  "."  "."  "."  "."
[3,] "."  "."  "."  "."  "."  "."  "."
[4,] "."  "."  "."  "."  "."  "."  "."
[5,] "."  "."  "."  "."  "."  "."  "."
[6,] "."  "."  "."  "."  "."  "."  "."
[7,] "."  "."  "."  "."  "."  "."  "."
[1] "Coord. lig = "
1: 3
2:
Read 1 item
[1] "Coord. col = "
1: 4
2:
Read 1 item
[1] "--> EH NON, A L'EAU !"
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,] "."  "."  "."  "."  "."  "."  "."
[2,] "."  "."  "."  "."  "."  "."  "."
[3,] "."  "."  "."  "x"  "."  "."  "."
[4,] "."  "."  "."  "."  "."  "."  "."
[5,] "."  "."  "."  "."  "."  "."  "."
[6,] "."  "."  "."  "."  "."  "."  "."
[7,] "."  "."  "."  "."  "."  "."  "."
[1] "VOIR : sud  est"
[1] "==> PLAY AGAIN..."
```