**Gilles HUNAULT**

2007

# Objets Statistiques

# en PHP

*Université d'Angers*

# Table des matières

# Pourquoi des objets statistiques en PHP ?

On pourrait croire que pour programmer des calculs statistiques élémentaires (moyennes, écart-types, comptages, $\chi^2$...) il suffit de quelques fonctions appliquées à des tableaux de chiffres. Notre pratique de l'automatisation de tels calculs et notre expérience de l'enseignement des statistiques nous amène à penser le contraire. Ce n'est pas tant le calcul qui pose problème – lorsqu'on connait les formules, quelques boucles "pour" et quelques structures conditionnelles en "si/alors/sinon" suffisent presque à tout calculer – que son application. Comment faire par exemple pour que le calcul automatique sur 40 colonnes de chiffres ne produise pas la moyenne de codes-sexe ou de numéros de téléphone ?

Le typage est une solution partielle car à des variables de natures différentes correspondent des actions (implémentées sous forme de fonctions) différentes. La notion de classe générale de variable statistique puis de sous-classe spécifique en fonction de la nature de la variable s'impose donc presque d'elle-même.

Pourquoi PHP alors, puisque ce n'est pas "le" langage objet par excellence ? Tout d'abord pour répondre à des besoins concrets de collègues bioinformaticiens dont les calculs sur des bases de données liées aux protéines doivent être disponibles "en ligne". Ensuite parce que les logiciels statistiques classiques, qu'ils soient payants (Sas, Statistica, S+, Spss...) ou gratuits (R) ne permettent pas d'accèder facilement aux définitions de classes pour les modifier ("surcharger") à notre convenance.

Enfin, nous avons choisi PHP pour "rentabiliser" nos cours sur le développement Web afin de montrer des exemples concrets de classes d'objets en PHP. La réalisation de calculs statistiques fournit l'occasion de détailler la construction et l'utilisation de ces classes et de leurs instances.

Ce document est conçu comme un support de cours pour le développement objets en PHP. Les prérequis en sont

- maitriser la programmation PHP classique (y compris HTML et les feuilles de style) ;
- connaitre les rudiments de la programmation objet (hors PHP) ;
- savoir calculer avec Excel ou OpenOfficeCalc une moyenne, un tri croisé, la connaissance de R ou de tout logiciel statistique étant un plus ;

Ce cours est progressif et propose des exercices (qu'il faut bien sur faire). Chacun sait qu'en programmation, au niveau des idées tout parait facile, mais lorsqu'il faut que les lignes de code fassent exactement ce qu'on veut, les heures de développement et déboggage s'accumulent. D'où des exercices corrigés avec une correction détaillée disponibles dans un autre document (*pédagogie oblige* !).

Ce manuel et les sources des classes ainsi que les solutions aux exercices sont disponibles sur le Web à l'adresse

http ://www.info.univ-angers.fr/pub/gh/Osep/

Faites-en bon usage.

# Chapitre 1.

# Implémentation d'une variable statistique (VS)

## 1.1 Données et fonctions élémentaires d'une VS

Une variable statistique (**VS** en raccourci) est avant-tout un tableau de valeurs numériques que nous désignerons par $tabVal$. Pour repérer les diverses **VS**, on leur donne un nom court (surtout utile lors de récapitulatif) auquel il convient d'adjoindre un nom long qui permet de détailler à quoi correspond exactement la variable. A ces deux données fondamentales, il faut ajouter la nature (ou "type") de la variable que nous implémenterons via une fonction nommée *nature()*.

Il y a – pour simplifier – principalement deux types de **VS** : les variables quantitatives (**QT**) et les variables qualitatives (**QL**). Une **QT** possède une unité (variable `unite` sans accent sur le e à cause de PHP) et se caractérise par le fait qu'on peut additionner ses valeurs, comme par exemple les variables `AGE`, `POIDS`, `TAILLE` pour une personne. Par contre les valeurs d'une **QL** correspondent en général à des codes arbitraires pour les *modalités* de la variable, comme par exemple la valeur 1 pour `Homme` et la valeur 2 pour `Femme` dans le cas de la variable `SEXE`.

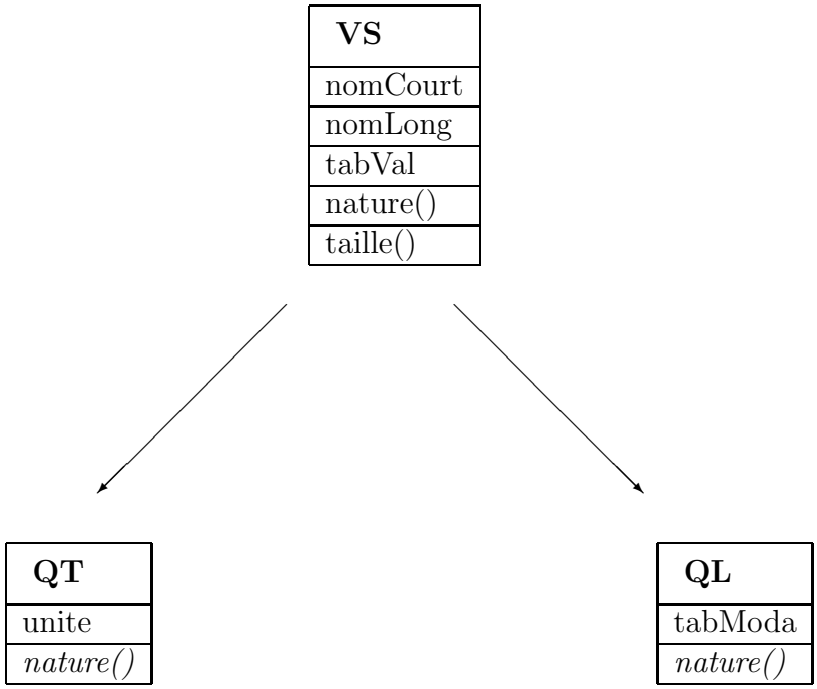
Le nombre de valeurs du tableau $tabVal$ se nomme *taille* de la variable. Nous décidons qu'elle sera accessible par la fonction `taille()` et qu'il ne sera pas possible de la modifier directement car elle doit être gérée par la création de

la variable ou le remplissage du tableau.

L'unité d'une **QT** est une donnée simple qu'on peut implémenter par une variable chaine de caractères alors que les modalités d'une **QL** doivent être définies par une table de hachage que nous nommerons *tabModa*.

## 1.2 Les classes VS, QT et QL

Pour l'instant, on peut donc représenter les classes de variables statistiques par le schéma d'héritage :




Qui dit donnée de classe dit bien sur fonction de définition de la donnée (*set*) et fonction d'accession à cette donnée (*get*). Il faut donc implémenter pour les variables scalaires

```
get_nomCourt()     get_nomLong()     get_unite()
set_nomCourt()     set_nomLong()     set_unite()
```

Comment remplir le tableau des données $tabVal$ ? Il y a plusieurs façons de créer des **VS** : à l'aide d'un tableau PHP déjà en mémoire, à l'aide d'une liste de valeurs fournies en tant que chaine de caractères, en lisant les données terme à terme dans un fichier... Il faut donc fournir une fonction adaptée à chaque cas. Nous réserverons `get_tabVal()` au remplissage de $tabVal$ via une chaine de caractères et nous nommerons `get_tabVal_fromTab()` la copie d'un tableau déja créé en $tabVal$.

Nous compléterons un peu plus tard ce remplissage de $tabVal$ avec la fonction `get_tabVal_parIndice` qui permet - notamment lors de la lecture d'un fichier texte – de remplir le tableau en fournissant la valeur de l'indice auquel est associée la donnée.

Il est d'usage de fournir un moyen de vérifier – ou tout simplement d'afficher – les informations saisies dans les objets. Nous implémenterons donc respectivement `decritVS`, `decritQT` et `decritQL` pour des variables respectivement générales, quantitatives et qualitatives. Si `decritQT` ne pose aucun problème puisqu'il s'agit d'ajouter l'affichage de l'unité à celui de `decritVS`, pour `decritQL` il faut inventer une fonction qui fusionne en une chaine de caractères les codes et les labels des modalités. Nous la nommerons `get_moda()`.

Nous n'écrirons pas de fonction `set_tabModa()` car PHP dispose de mécanismes pour définir "à la volée" des tables de hachage. Pour réaliser une implémentation simple des trois classes **VS**, **QT** et **QL** il suffit alors de regrouper les données et les fonctions cités et de leur ajouter un "constructeur", c'est à dire une fonction qui permet de créer des instances d'objet.

Une telle implémentation est disponible à l'adresse

http ://www.info.univ-angers.fr/pub/gh/Osep/

On lira soigneusement le code source pour voir l'ordre des paramètres, quels paramètres sont facultatifs et comment on réutilise les fonctions de la classe parent. Comme tous les choix de programmation, nos choix sont critiquables :

- pourquoi ne pas imposer au moins le nom de la variable ?

- pourquoi autoriser la création d'un objet **VS** avec un $tabVal$ vide ?

Ces questions font partie de la réflexion sur la conception des classes et la (tentative de) justification de ces choix sera donnée en cours.

Avec une telle implémentation mise dans le fichier `statobj1.php`, on peut écrire pour créer puis afficher nos variables statistiques :

```
include("statobj1.php") ;

# Variables générales

     $mv = new VS("MAV") ;
     $mv->decritVS() ;

     $mv = new VS("AGE","15 45 50 60 70") ;
     $mv->decritVS() ;

     $mv = new VS("SX","1 2 1","Sexe de la personne") ;
     $mv->decritVS() ;

# Variables quantitatives

     $qt1 = new QT("AG") ;
     $qt1->decritQT() ;

     $qt2 = new QT("AGE","15 45 50 60 70","ans","Age de la personne") ;
     $qt2->decritQT() ;

# Variables qualitatives

     $ql1 = new QL("SX") ;
     $ql1->decritQL() ;

     $ql2 = new QL("SEXE","1 2 1 2 1",
            array( 1 => "Homme", 2 => "Femme"),"Sexe de la personne") ;
     $ql2->decritQL() ;

# Affectations d'objets

     $otherQl = $ql2 ;
     $otherQl->decritQL() ;
     $ql1     = $ql2 ; # wrong ? à vous de le dire...
```

# Exercices du chapitre 1.

- compléter le schéma d'héritage en y incluant toutes les fonctions citées
- générer des valeurs QT automatiques avec `random()`, arrondi, a*x+b
- générer des valeurs QL avec `rep()`, avec `copies()`
- comment utiliser `range()` pour une variable uniforme ?
- quelles sont les lois usuelles en statistiques ? et leur nature ? comment les générer ?
- implémenter `get_tabVal_fromTab()` et `get_tabVal_parIndice()`
- qu'obtient-on avec `"$ql2->decritQT() ;"` ?

# Chapitre 2.

# Analyse d'une variable statistique

Au tableau des valeurs d'une **VS**, nous avions associé la fonction `taille()` qui se contente de renvoyer le nombre de valeurs du tableau. Après réflexion on peut associer facilement et simplement d'autres fonctions assez classiques pour un tableau de valeurs numériques qui viendront donc se situer au niveau général de la classe **VS**.

Ainsi `nbEgal(v)` sera la fonction qui renvoie le nombre de fois où on a trouvé la valeur $v$ dans le tableau, `nbPlusPetit(v)` renverra le nombre de valeurs strictement plus petites que $v$ et `nbPlusGrand(v)` le nombre de valeurs strictement plus grandes que $v$. Il est utile et immédiat d'implémenter la fonction `nbPlusPetitOuEgal(v)` qui renvoie le nombre de valeurs inférieures ou égales à car c'est simplement la somme `nbEgal(v) + nbPlusPetit(v)`, sauf on si veut optimiser la vitesse de calcul[1]. Il est courant en statistique de fournir des valeurs absolues mais aussi relatives. Il faudra donc au passage implémenter [hors classes?]une fonction $pct(a, b, c)$ qui donne avec $c$ décimales le pourcentage correspondant au rapport $a/b$.

Le *mode* d'une **VS** est la valeur la plus fréquente du tableau des valeurs. Son calcul est simple à implémenter par fonction de même que les fonctions `valMin()` et `valMax()` qui donnent respectivement la plus petite valeur et la plus grande valeur dans le tableau car *php* dispose des fonctions `sort()`, `rsort()` et `array_count_values()`.

[1] Pourquoi ?

La définition de la classe **VS** s'enrichit donc pour devenir

| **VS** |
|---|
| `nomCourt` |
| `nomLong` |
| `tabVal` |
| `nature()` |
| `taille()` |
| `decritVS()` |
| `set_tabVal()` |
| `get_nomCourt()` |
| `set_nomCourt()` |
| `get_nomLong()` |
| `set_nomLong()` |
| `nbEgal()` |
| `nbPlusPetit()` |
| `nbPlusPetitOuEgal()` |
| `nbPlusGrand()` |
| `nbPlusGrandOuEgal()` |
| `mode()` |
| `valMin()` |
| `valMax()` |

## 2.1 Analyse d'une variable qualitative (QL)

Réaliser l'analyse d'une **QL** se dit en termes statistiques "effectuer le tri à plat de la variable". Cela consiste simplement à fournir les effectifs absolus et relatifs des différentes modalités c'est à dire qu'il faut compter combien de fois chaque code apparait et afficher l'effectif et le pourcentage correspondant. Il est d'usage d'utiliser plusieurs affichages : par ordre numérique de code, par ordre alphabétique de label, par fréquence décroissante. Ce dernier affichage doit être celui par défaut car il montre immédiatement "quelles variables varient". Le résultat d'un tri à plat est une table de hachage (car il faut relier le comptage au label et non pas au code). Il parait donc "raisonnable" d'inventer une fonction `triAplat()` qui renvoie cette table et de lui adjoindre une fonction `afficheTriAplat()` qui à l'aide d'un paramètre bien choisi (à

définir) réalisera les divers affichages demandés.

Là encore, les fonctions `sort()`, `sort()` et `array_count_values()` de *php* facilitent grandement le travail.

On double classiquement l'affichage de ce tri à plat d'un histogramme des fréquences qui est un graphique. Nous y reviendrons sur les graphiques dans un chapitre ultérieur.

La définition de la classe **QL** s'enrichit donc pour devenir

```
QL
tabModa
nature()
set_tabModa()
get_modalites()
triAplat()
afficheTriAplat()
```

## 2.2 Analyse d'une variable quantitative (QT)

L'analyse d'une **QT** consiste à fournir un certain nombre d'indicateurs nommés moyenne, variance, écart-type, médiane, quartiles, percentiles, intervalles de confiance... et bien sûr à doubler ces indicateurs numériques de graphiques comme les "scatterplot", "boxplots" et autres histogrammes de classes...

La définition de la classe **QT** s'enrichit donc pour devenir

```
QT
unite
nature()
moyenne()
variance()
ect()
mediane()
quartile()
quantile()
```

Passons maintenant aux différentes formules à programmer. Pour $n$ valeurs $x_i$ (où $i$ varie de 1 à $n$) la moyenne (arithmétique) notée $m$ est la somme des $x_i$ divisée par $n$ et la variance est la moyenne des carrés des différences $x_i - m$. Il n'est pas très difficile de voir qu'on peut programmer différentes formules pour $m$ et pour $V$ aussi bien en itératif qu'en récursif. Dans ces formules, $s_k$ désigne la somme des $k$ premières des valeurs des $x_i$, $c_k$ la somme de leurs carrés, $m_k$ leur moyenne et $v_k$ leur variance :

$$\text{(M1)} \quad m = \sum_{i=1}^{n} (x_i/n)$$

$$\text{(M2)} \quad m = (1/n) \sum_{i=1}^{n} x_i$$

$$\text{(M3)} \quad m = s_n \ / \ n$$

$$\text{(M4)} \quad m = \frac{(n-1)\ m_{n-1} + x_n}{n}$$

$$\text{(V1)} \quad V = \sum_{i=1}^{n} (x_i - m)^2/(n-1)$$

$$\text{(V2)} \quad V = (1/(n-1)) \sum_{i=1}^{n} (x_i - m)^2$$

$$\text{(V3)} \quad V = \left( (1/n) \sum_{i=1}^{n} {x_i}^2 \right) - \left( (1/n) \sum_{i=1}^{n} x_i \right)^2$$

$$\text{(V4)} \quad n^2\ V = n\ c_n\ -\ m_n$$

$$\text{(V5)} \quad n^2\ V = n\ (\ (n-1)^2\ v_{n-1} + m_{n-1}\ ) + n\ {x_n}^2 - m$$

Quelle est "LA" bonne formule pour $m$ et quelle est "LA" bonne formule pour $V$ en termes de programmation *php* ? Nous laissons au lecteur le soin de trouver la réponse, sachant qu'il existe en *php* des fonctions comme `array_map`

et `array_sum` et que nous proposons de discuter de ces formules sous forme d'exercice en fin de chapitre.

La médiane, les quartiles, les centiles et quantiles sont des valeurs basées sur la répartition des valeurs une fois ordonnées : le quantile de valeur $t$, noté $Q_t$ est une valeur numérique $y$ ("réelle", non forcément exactement égale à un des $x_i$) telle qu'on a une fréquence $t$ de valeurs inférieure ou égale à $y$.

On note classiquement $q_1$ le quantile $Q_{0.25}$ qu'on nomme premier quartile. De même $q_2 = Q_{0.50}$ ou deuxième quartile est nommé "médiane" : on a donc en principe autant de valeurs avant la médiane qu'après[2], un quart des données avant le premier quartile. Nous n'implémenterons que la médiane, laissant la programmation des autres quantiles sous forme d'exercice.

[2] Cela ne donne donc pas une "vraie" définition (non ambigue) pour la médiane. On s'en rendra compte avec les valeurs 1 1 1 9 9 9

# Exercices du chapitre 2.

- implémenter `ind_valMin(opt)` et `ind_valMax(opt)` qui donnne l'indice de la première ou de la dernière occurence du minimum, du maximum suivant la valeur du paramètre `opt`
- implémenter `nbEgal()`, `nbPlusPetit()` *etc.* à l'aide d'une seule fonction paramétrée
- vérifier que les différentes implémentations de la moyenne fournissent le même résultat ; différences de vitesse d'exécution ?
- vérifier que les différentes implémentations de la variance ne donnent pas toutes le même résultat ; différences de vitesse d'exécution ?

# Chapitre 3.

# Analyse de deux variables statistiques

## 3.1 Analyse de deux variables quantitatives (QT)

La liaison entre deux variables **QT** `X` et `Y` peut correspondre à une dépendance fonctionnelle générale, une dépendance monotone ou à une correspondance exacte. Nous ne traiterons ici que le cas très usuel de la liaison linéaire `Y = aX + b` qui est un cas particulier de dépendance monotone. Pour ce cas, il est classique de calculer le coefficient $\rho$ de corrélation linéaire de Pearson et d'indiquer les coefficients $a$ et $b$ lorsque $\rho$ est "proche" de `+1` ou `-1` selon les formules

$$\begin{aligned} \rho(X,Y) &= \sum(X_i - m_X)(y_i - m_Y) \ / \ (\ (n-1)\sigma(X)/\sigma(Y)\ ) \\ a &= \rho.\sigma_Y/\sigma_X \\ b &= m_y - a.m_X \end{aligned}$$

Si la fonction $\rho \rightsquigarrow \rho(X,Y)$ est symétrique, les fonctions associées aux coefficients indiquées ici ne le sont pas et correspondent seulement à l'équation `Y = aX + b`. C'est pourquoi nous implémentons ce calcul sous forme d'une fonction paramétrée nommé $coefcorr$ telle que `Y->coefcorr(X)` renvoie le tableau contenant dans cet ordre $\rho$, $a$ et $b$ pour cette même équation, $a$ et

$b$ n'ayant de sens que si $\rho$ est proche de 1 en valeur absolue, comme déja indiqué.

Voici un exemple d'utilisation de cette fonction :

```
include("statobj2.php") ;

# Analyse de deux QT

$qt1 = new QT("DIST","1 2 3 4","c_km",
              "Distance parcourue en centaines de km") ;
$qt2 = new QT("CONS","6 11.5 18.6 24.4","l",
              "Consommation en litres d'essences") ;

$cor = array() ;
$cor = $qt2->coefcorr($qt1) ;
$rho = $cor[0] ;
$a   = $cor[1] ;
$b   = $cor[2] ;
```

## 3.2 Analyse de deux variables qualitatives (QL)

De même que l'analyse d'une seule variable **QL** produit un tableau nommé *tri à plat*, l'analyse de deux variables **QL** produit un tableau que l'on nomme *tri croisé*. Ce tableau vient recenser le nombre de fois où chaque couple (ou "croisement" d'où le nom du tableau) de modalités apparait. Il est d'usage d'utiliser quatre affichages différents selon les besoins :

- l'affichage direct des comptages (ou "effectifs absolus"),
- l'affichage relatif des pourcentages par rapport au total général,
- l'affichage relatif des pourcentages par rapport aux totaux en lignes,
- l'affichage relatif des pourcentages par rapport aux totaux en colonnes.

Nous implémentons ce calcul sous la forme de la fonction `triCroise` telle que `$ql1->triCroise($ql2)` renvoie le tableau des effectifs de croisement où les modalités de `$ql1` sont en lignes et celles de `$ql2` sont en colonne.

Pour afficher un tel tableau, nous implémentons la fonction `afficheTriCroise` qui en plus du paramètre correspondant à la variable **QL** à croiser avec l'objet utilise un paramètre désigné ici par `$num`. La valeur de `$num$` est 1, 2,

3 ou 4 selon l'affichage demandé, la valeur 1 étant celle par défaut et donc facultative.

Voici un exemple de l'appel de ces fonctions :

```
include("statobj2.php") ;

# Analyse de deux QL

$ql1 = new QL("REGLEMENT","1 2 3 1 1",
       array( 1 => "Mensuel", 2 => "Trimestriel") ,"Annuel") ;

$ql2 = new QL("SEXE","1 2 1 2 1",
       array( 1 => "Homme", 2 => "Femme"),"Sexe de la personne") ;

$ql1->afficheTriCroise($ql2) ;
```

# Exercices du chapitre 3.

- inverser "à la main" l'équation reliant la consommation d'essence et la distance parcourue puis écrire le progamme php qui affiche les deux équations ;
- que trouve-t-on si on calcule $\rho(X, X)$ ?
- peut-on écrire
  `echo " rho vaut ".$qt2->coefcorr($qt1)[0] ;`
- comment afficher l'effectif associé au croisement des modes pour 2 QL ?