Laboratoire de programmation en C++

2ème informatique de gestion
(3ère partie : 3ème quart)

Année académique 2010-2011

Racing WorldChampionship Statistics

**Claude Vilvens – Mounawar Madani - Denys Mercenier –
Jean-Marc Wagner**

1. Le contexte : l'analyse des données de grands prix

A la suite des travaux du laboratoire de C++ des deux premiers quarts, nous allons à présent nous intéresser plus précisément à un aspect "gestion" des données : l'analyse statistique des résultats de l'ensemble des grands prix de Formule 1 disputés par Michael Schumacher tout au long de sa carrière, essentiellement pour ce qui concerne ses positions sur la grille de départ et à l'arrivée.

L'idée de base est de reprendre une version simplifiée de la classe Race qui doit comprendre au minimum comme variables membres le pays où s'est déroulé le grand prix, la position (type int) sur la grille de départ, la position à l'arrivée (type int) et le nombre de spectateurs (type int) présents sur le circuit lors de la course.

Moyennant ces aménagements, on peut donc dire que les données brutes sont disponibles pour les analystes. En fait, toutes les informations des grands prix ont été encodées dans un fichier de type texte ***MichaelSchumacher.csv***. Le contenu d'un tel fichier est confectionné selon les règles simples suivantes :

- 1ère ligne = noms des champs : <Numero du grand prix> ; <Annee> ; <Pays> ; <Ecurie> ; <Constructeur> ; <Moteur> ; <Pneu> ; <Position sur la grille de départ> ; <Position à l'arrivée> ; <Remarques> ; <Nombre de spectateurs> ;
- lignes suivantes = les t-uples correspondant

Le séparateur est donc à priori le point-virgule, mais on pourrait opter pour un autre un autre séparateur. Il devient alors facile de le parcourir ligne par ligne puis de découper chaque ligne avec strtok() pour isoler les données et instancier les objets correspondants. Ce fichier vous sera gracieusement fourni. Pour information, toutes les données présentes dans ce fichier sont réelles et peuvent être trouvées sur le site www.statsf1.com, mis à part le nombre de spectateurs, données fictives créées pour l'exercice.

Reste à présent à traiter ces données pour les présenter de manière à être exploitées.

2. Classe de calcul et classes source de données

La classe **ParametresStatistiques1D** a pour rôle, sans surprise, de calculer les paramètres statistiques d'une série statistique à une dimension :

- tendance centrale : moyenne (getMoyenne()), mode (getModes(…)) (attention au cas multimodal), médiane (getMediane())
- dispersion : écart-type (getEcartType()), étendue (getEtendue()), coefficient de variation (getCV()).

Elle va réaliser ces calculs sur les données contenues dans un objet passé à son constructeur. On attend de cet objet qu'il soit capable

- de fournir un couple (xi, ni) (c'est-à-dire valeur-effectif) par getData(int i);
- de fournir le nombre de valeurs distinctes xi (supposées consécutives) par getN();
- de fournir le tableau de tous ces couples par getData().

Ces couples sont des instances de la classe **DataX** qui encapsule simplement un couple valeur-effectif. L'objet qui contient ces données peut être de nature assez variable. Aussi, on définira la classe abstraite **DataSourceX** qui déclare (sans les implémenter) les deux méthodes getData() et la méthode getN(). En prévision des statistiques à deux dimensions, on fera dériver cette classe d'une classe abstraite encore plus générale, **DataSource**, qui déclare simplement que toute source de données possède un effectif (n) et un nom – elle déclare donc les méthodes getNom() et getn() (dans notre cas, ce n est la somme des ni – ne pas confondre avec getN() évoqué ci-dessus qui fournit le nombre de valeurs distinctes observées !).

L'objet que l'on passe au constructeur de ParametresStatistiques1D est donc en définitive une instance d'une classe dérivée de DataSourceX. Bien sûr, l'objet ParametresStatistiques1D lance les exceptions ad hoc si il constate que le DataSourceX est mal formé (effectif non positif, référence du tableau de (xi, ni) nulle, etc).

3. Génération d'une source de données

Concrètement, comment une classe qui contient certaines données, comme notre classe Race par exemple, va-t-elle pouvoir être associée à une classe dérivée de DataSourceX ? En fait :

- tout d'abord, la classe **Races** est un simple container d'objets Race; cependant, les données que l'on trouve dans une telle classe ne sont pas présentées clairement : elles ne sont ni triées, ni regroupées;
- la classe **EchantillonPositionGrilleDepart**, qui hérite de DataSourceX et dont le constructeur reçoit un objet Races, sert à contenir les couples (xi, ni) – son constructeur a donc pour tâche de réaliser les comptages et la mise en ordre. (xi représente donc une position sur la grille de départ et ni le nombre de fois que M. Schumacher est parti de cette position sur l'ensemble de ses grands prix).

Il n'y a donc plus qu'à passer l'objet **EchantillonPositionGrilleDepart** au constructeur d'un objet ParametresStatistiques1D pour enfin recueillir les paramètres statistiques souhaités : le constructeur va donc réaliser toutes les opérations de calcul des différents paramètres statistiques. L'ensemble de ceux-ci sera placé dans une variable membre qui est un objet **Map** de la STL (voir chapitre 11 du cours de théorie, paragraphe 16, pages 443-449) : le nom des paramètres en constituera la clé. Une méthode permettra également de produire un fichier texte imprimable et formaté reprenant ces paramètres avec le nom de la série.

La démarche est clairement reproductible pour les positions à l'arrivée des différents grands prix (classe **EchantillonPositionArrivee**) – à faire donc. Attention que Michael Schumacher n'a pas terminé tous les grands prix auxquels il a participé. L'effectif total (le n donc) n'est donc pas le même dans les 2 séries statistiques. (Rem : ab = abandon, dsq = disqualifié)

Mais, par contre, pour le nombre de spectateurs présents sur le circuit à chaque grand prix, un petit problème se fait jour …

4. Les sources de données avec classes

En effet, si la position sur la grille de départ est une variable statistique discrète au nombre de valeurs possibles limitées (disons de 1 à 25), c'est plus discutable pour le nombre de spectateurs (de 50000 à 100000), dont les valeurs possibles sont très nombreuses avec un très grand nombre de valeurs intermédiaires possibles : on arrive ainsi à des variables statistiques continues ou pseudo-continues. Dans ce cas, le remède est connu : il faut grouper les valeurs possibles en classes – c'est le terme consacré en statistique pour désigner des intervalles successifs de la forme [50000, 55000[, [55000, 60000[, etc et cela n'a bien sûr rien à voir avec le terme de "classe" de la POO. Quoique …

On peut imaginer que la classe **EchantillonNombreSpectateurs** possède un constructeur ayant pour tâche de réaliser les comptages par classes. Ce constructeur reçoit comme paramètre, outre un objet Races

- soit le nombre de classes, l'étendue d'une classe, la borne inférieure de la première classe; une exception est lancée si une valeur est trouvée en dehors de ces classes;
- soit simplement le nombre de classes : celles-ci sont alors déterminées à partir de toutes les valeurs observées.

Dans les deux cas, le rôle des valeurs xi évoquées précédemment est joué par le centre des classes.

5. Les statistiques à deux dimensions

Un dernier problème intéressant à traiter (si on se limite à la statistique descriptive sans déborder sur l'inférence statistique) est celui de la régression-corrélation. Par exemple, existe-t-il une corrélation entre la position de Michael Schumacher à l'arrivée et sa position sur la grille de départ (s'il a terminé la course bien sûr) ? On va donc imaginer, de manière similaire à ce qui a déjà été fait :

* la classe **DataXY**, classe qui encapsule simplement un couple valeur_x-valeur_y;
* la classe abstraite **DataSourceXY** (héritée de DataSource) qui déclare les méthodes

- getDataX(int i) et getDataY(int i) qui fournit la valeur xi ou yi;
- getDataXY(int i) qui fournit un couple (xi, yi) sous forme d'un objet DataXY;
- getData().qui fournit le tableau des objets DataXY;
- getn() qui fournit le nombre de couples (xi, yi);

* la classe **ParametresStatistiques2D** a pour rôle, sans surprise, de calculer les paramètres statistiques d'une série statistique à deux dimensions :

- tendance centrale : moyennes (getMoyenneX() et getMoyenneY()):
- dispersion : écart-types (getEcartTypeX()) et getEcartTypeY()), étendue (getEtendueX() et getEtendueY());
- coefficient de corrélation linéaire r (getR()) et paramètres de régression (getA() et getB());

et qui permet d'extrapoler une valeur au moyen de la droite de régression avec la méthode getYest(int x) . La classe **EchantillonPositionArriveeDepart** qui hérite de DataSourceXY sera la classe dont on passera une instance à l'objet ParametresStatistiques2D. Donc pour un grand prix donné, yi représente la position à l'arrivée tandis que xi représente la position sur la grille de départ.

6. Le laboratoire

Le laboratoire de programmation C++ 3ème partie qui vous est demandé devra donc produire comme résultat :

- une librairie reprenant les classes ParametresStatistiques1D, DataX, DataSourceX, DataSource, ParametresStatistiques2D, DataXY, DataSourceXY;
- les diverses classes à objectif statistique à une et deux dimensions associées aux grands prix;
- une application qui permet de réaliser les tests correspondants;
- un diagramme de classes UML reflétant les classes intervenant dans cette application (peu importe le logiciel utilisé pour réaliser ce diagramme, mais il ne peut s'agir d'un reverse engineering).

Le dossier est prévu à priori pour une équipe de deux étudiants qui devront donc se coordonner intelligemment et se faire confiance. Il est aussi possible de présenter le travail seul (les avantages et inconvénients d'un travail par deux s'échangent).

Il s'agit bien d'un laboratoire de C++ sous UNIX. La machine de développement sera Sunray. Même s'il n'est pas interdit (que du contraire) de travailler sur un environnement de votre choix (Dev-C++ sur PC/Windows sera privilégié car compatible avec C++/Sunray – à la rigueur Visual C++ sous Windows, g++ sous Linux, etc …) à domicile, seul le code compilable sous Sunray sera pris en compte !!!

Cette troisième partie sera évaluée durant les premières semaines du 4ème quart, selon l'horaire établi par le professeur de laboratoire.

Quelques formules utiles (voir "Inférence statistique" de Claude Vilvens, ed. DEFI).

moyenne :

$$m = \bar{x} = \frac{\sum_{i=1}^{N} n_i * x_i}{n}$$

médiane m_e (classes [ai, bi[ ):

$$m_e = b_i + \frac{\frac{n}{2} - \sum_{j=1}^{i-1} n_j}{n_i} * a_i$$

mode m_o (mais il peut y avoir plusieurs modes) :

$$m_o = b_i + \frac{n_i - n_{i-1}}{(n_i - n_{i-1}) + (n_i - n_{i+1})} * a_i$$

écart-type s (carré = **variance)** :

$$s = \sqrt{\frac{\sum_{i=1}^{N} n_i.(x_i - \bar{x})^2}{n}} = \sqrt{\sum_{i=1}^{N} f_i.(x_i - \bar{x})^2}$$

coefficient de variation :

$$CV = \frac{s}{\bar{x}}.100$$

régression linéaire :

$$a = \frac{\sum_{i=1}^{n} x_i * y_i - \bar{x} * \sum_{i=1}^{n} y_i}{\sum_{i=1}^{n} x_i^2 - \bar{x} * \sum_{i=1}^{n} x_i} = \frac{\sum_{i=1}^{n} X_i * Y_i}{\sum_{i=1}^{n} X_i^2}$$

$$b = \bar{y} - a * \bar{x}$$

où

$$X_i = x_i - \overline{x} \quad \text{et} \quad Y_i = y_i - \overline{y}$$

coefficient de corrélation linéaire :

$$r = \frac{\sum_{i=1}^{n} (x_i - \overline{x}).(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n} (x_i - \overline{x})^2} . \sqrt{\sum_{i=1}^{n} (y_i - \overline{y})^2}} = \frac{\sum_{i=1}^{n} X_i * Y_i}{\sqrt{\sum_{i=1}^{n} X_i^2} * \sqrt{\sum_{i=1}^{n} Y_i^2}}$$

C. Enoncé du 3ème quart : *Racing WorldChampionship Statistics*

Date de rentrée du dossier (papier) : **premier lundi du 4ième quart à 12h00 au plus tard**
Evaluation définitive de ce dossier : à partir du **21/03/2011** selon les modalités indiquées par le professeur de laboratoire

point principaux de l'évaluation	subdivisions
Classes statistiques 1D de base	Classe abstraite **DataSource**
	Classe abstraite **DataSourceX** et classe **DataX**
	Classe **ParametresStatistiques1D** (constructeur et calcul des paramètres, Map, getters, création fichier texte formaté avec résultats)
	Exceptions en cas de source de données mal formée ?
	Tests préliminaires de ces classes ?
Analyse des sinistres (1D) : données discrètes	Classe **Race** remaniée et classe **Races**
	Parsing du fichier .csv
	Classe **EchantillonPositionGrilleDepart** et son constructeur (comptage)
	Classe **EchantillonPositionArrivee** et son constructeur (comptage)
	Tests de ces classes et analyse des résultats obtenus à partir du fichier .csv
Analyse des sinistres (1D) : données en classes	Classe **EchantillonNombreSpectateurs**
	Constructeurs polymorphes de ces classes ?
	Tests de ces classes et analyse des résultats
Classes statistiques 2D	Classe abstraite **DataSourceXY** et classe **DataXY**
	Classe **ParametresStatistiques2D** (constructeur et calcul des paramètres, Map ?, getters, création d'un fichier texte formaté ?)
	Exceptions en cas de source de données mal formée ?
	Tests préliminaires de ces classes ?
Analyse des sinistres (2D)	Classe **EchantillonPositionArriveeDepart**
	Calcul des paramètres et estimation
	Tests des classes et analyse des résultats