Sorbonne Université **M1 Mathématiques**
2022–2023 **MU4M056 : Programmation en C++**

TP noté 2 : Lois de mélange

L'objectif de ce TP est d'implémenter en C++ la notion de loi de mélange, pour des mélanges finis et dénombrables.

Les fichiers fournis doivent conserver les mêmes noms et les fichiers rendus doivent être compilables sans erreur, quitte à commenter les parties de code qui ne marchent pas.

Remarque : dans le texte ci-dessous, le symbole **(T)** *indique un fichier à télécharger sur Moodle avant le début du partiel.*

1 Introduction mathématique et implémentation

1.1 Description mathématique

Une loi de mélange est la loi de probabilité d'une variable aléatoire s'obtenant à partir d'une famille paramétrique de variables aléatoires de la manière suivante : une variable aléatoire est choisie au hasard parmi la famille de variables aléatoires donnée, puis la valeur de la variable aléatoire sélectionnée est réalisée. La famille est dite paramétrique dans le sens que les variables aléatoires ont même loi, mais avec des valeurs de paramètres différentes, par exemple des gaussiennes avec moyenne et/ou variance différentes.

Nous allons traiter dans ce TP seulement le cas de familles finies et dénombrables. Les différentes lois de la famille sont dites *modes* du mélange et les chances de choisir chacune des lois sont des poids positifs de somme totale 1, autrement dit des probabilités. Une loi de mélange peut être utilisée pour modéliser une population statistique avec des sous-populations, où les différentes lois correspondent aux sous-populations, et les poids sont les proportions de chaque sous-population dans la population globale.

Autrement dit, si $f_\theta(x)$ est la densité d'une variable aléatoire de loi μ_θ, la densité du mélange associé à l'ensemble de paramètres $(\alpha_i, \theta_i)_{i=1,\dots,n}$, où les poids α_i sont tels que $\sum \alpha_i = 1$, est donnée par $\sum \alpha_i f_{\theta_i}(x)$. Il en est de même pour les lois discrètes.

1.2 Implémentation par des templates

Nous choisissons d'implémenter une loi de mélange de la façon suivante dans un fichier `loi_melange.hpp` ***(T)*** :

```
template <class RV>
class Melange{
   private:
      std::vector<RV> modes;
      std::discrete_distribution<int> choose_mode;
   public:
      //cf fichier fourni
};
```

avec les spécifications suivantes :

- l'objet `modes` de type `std::vector<RV>` contient les n distributions paramétriques de type `RV` avec les paramètres θ_i pour $i = 1, \dots, n$ et `choose_mode` est la distribution qui renvoie l'indice i (c'est à dire choisi le i-ème mode du mélange) avec probabilité α_i ;
- le constructeur par défaut construit un mélange vide ;
- la méthode `nb_of_modes()` renvoie le nombre de modes ;

— la méthode `add_mode(alpha,to_add)` ajoute une distribution `to_add` au mélange, avec une probabilité `alpha` $\in (0,1)$ d'être choisie dans le mélange : `to_add` est ajoutée au vecteur des modes, les probabilités de `choose_mode` sont multipliées par `1-alpha` avant d'y ajouter `alpha` , de sorte que les poids, après l'ajout du nouveau mode, gardent la propriété d'avoir une somme totale 1 ;
— les autres méthodes sont documentées plus bas dans les questions correspondantes.

1.3 Documentations sur les objets de type `std::discrete_distribution<int>` et sur les distributions de la bibliothèque `random`

La distribution `std::discrete_distribution` contenue dans la bibliothèque `random` produit des entiers aléatoires sur l'intervalle $[0,n)$, où la probabilité de chaque entier i est définie comme $\alpha_i/\sum \alpha_i$, c'est-à-dire le poids du i-ème entier divisé par la somme des n poids (dans notre cas les poids seront déjà normalisés). Si `alpha` est un objet de type `std::vector<double>` contenant `n` poids et `G` est un générateur de nombres aléatoires :

— un objet `X` de type `std::discrete_distribution<int>` se construit à partir des probabilités `alpha` via l'appel
```
std::discrete_distribution<int> X(alpha.begin(), alpha.end());
```
— l'appel `X(G)` renvoie l'entier `i` à valeurs dans `0, ..., n-1` avec probabilité `alpha[i]` ;
— l'appel `X.probabilities()` renvoie le vecteur des poids.

Si `RV` est une distribution de la bibliothèque `random` :

— `RV::return_type` indique le type de retour d'un appel de `RV(G)` avec `G` générateur de nombres aléatoires ;
— `RV::param_type` indique le type des paramètres de la distribution ;
— si `p` est un objet de type `RV::param_type` , on peut créer un objet de type `RV` de paramètres `p` via l'appel `RV(p)` , c'est-à-dire que `RV` a une constructeur de signature
```
RV(const param_type& p)
```
;
— il existe un opérateur `<<` , ami de la classe `RV` , qui à la signature suivante
```
std::ostream& operator<<( std::ostream& o, const RV& d )
```
et qui permet d'afficher une représentation textuelle des paramètres de la distribution et de l'état interne du flux `o` ;
— la méthode `RV.param()` permet de récupérer les paramètres de la loi `RV` sous forme d'un objet de type `RV::param_type` .

2 Implémentation pas à pas

2.1 Fonctionnalités élémentaires

1. Dans le fichier `loi_melange.hpp` ***(T)***, compléter les prototypes des méthodes manquantes (type d'arguments complet, `const` éventuels, etc.). Ajouter le constructeur par défaut.

2. Écrire le code de la méthode `nb_of_modes` .

3. Écrire le code de la méthode `weights` .

4. Écrire le code de `operator[]` qui prend en argument un entier `i` et renvoie le `i` -ème mode.

5. Nous fournissons la méthode
`void print_type() const { std::cout << typeid( RV() ).name() << "\n ";}`
qui affiche le type d'une distribution. Après avoir ajouté les entêtes nécessaires, tester le code dans le fichier `main.cpp` ***(T)*** en vérifiant qu'il compile et fonctionne.

6. Écrire le code de la méthode `add_mode(alpha,to_add)` . Attention, lors d'un ajout dans un mélange vide, le poids du premier mode inséré sera forcément 1 (le poids donné en paramètre de la méthode `add_mode` sera dans ce cas ignoré). On pourra, si on le souhaite, utiliser la fonction `transform` de la bibliothèque `algorithm` pour transformer les poids. Cette fonction a la définition suivante :

```
template<class InputIt, class OutputIt, class UnaryOperation>
OutputIt transform(InputIt first1, InputIt last1,
                   OutputIt d_first, UnaryOperation unary_op)
{
    while (first1 != last1)
        *d_first++ = unary_op(*first1++);

    return d_first;
}
```

7. Tester la méthode `add_mode` dans le `main.cpp` en ajoutant au mélange vide `M_bernoulli` une loi de Bernoulli de paramètres 0.5 avec un poids 1 et ensuite une loi de Bernoulli de paramètres 0.2 de poids 0.3.

2.2 Réalisation d'un mélange

Pour simuler selon une loi de mélange de façon cohérente avec la bibliothèque `random` , il est nécessaire d'ajouter à la classe `Melange` un `operator()` qui prend en argument un générateur de nombres aléatoires `G` de type générique `RNG` (c'est donc un prototype de méthode, un template) et qui renvoie une réalisation du mélange, en choisissant un mode `i` selon la loi `choose_mode` et en renvoyant une réalisation de la loi `modes[i]` . Le type de retour de cet opérateur est le type de retour de la distribution du mélange `RV` , que l'on peut récupérer avec l'appel `typename RV::result_type` (le mot clef `typename` est nécessaire pour signifier au compilateur qu'il s'agit bien d'un type, et non pas d'une valeur).

8. Écrire le code du prototype `operator()` ainsi décrit.

9. Tester ce code dans le `main.cpp` en affichant sur le terminal une réalisation du mélange `M_bernoulli` .

Il ne vous aura peut être pas échappé qu'un mélange de lois de Bernoulli de paramètres $(p_1, \dots, p_n)$ et de poids $(\alpha_1, \dots, \alpha_n)$ n'a pas beaucoup de sens, parce que le mélange a lui aussi une loi de Bernoulli, de paramètre $p = \sum_{i=1}^{n} \alpha_i p_i$.

10. Vérifier ce résultat sur le mélange `M_bernoulli` , en calculant la moyenne empirique de 100 000 réalisations du mélange et en affichant la différence entre la moyenne empirique et la moyenne théorique (qui doit être proche de 0).

2.3 Affichage d'un mélange et lecture dans un fichier

11. Écrire le code de l'opérateur `<<` pour un objet de la classe `Melange` qui affiche une première ligne avec la taille du mélange, suivie d'une ligne pour chaque mode du mélange. Vous pourrez choisir le type de lien d'amitié (*introvertie* ou *extravertie*) que vous préférez.

12. Écrire un constructeur à partir d'un vecteur de paramètres de type `std::vector<typename RV::param_type>` et d'un vecteur de poids de type `std::vector<double>` .

13. Le fichier `"melange_gauss_1.txt"` ***(T)*** contient une première ligne avec la taille du mélange et ensuite une colonne de poids α_i et deux colonnes des paramètres de lois gaussiennes (moyenne, déviation standard). Dans le fichier `main.cpp` créer un mélange gaussien `M_gaussian_1` à partir de ce fichier, ce qui correspond dans l'ordre à :

- ouvrir un flux de lecture sur le fichier ;
- lire la taille du mélange et créer un vecteur de double et un vecteur de paramètres de distribution normale de la bonne taille ;
- lire et insérer dans ces vecteurs les poids et les paramètres ;
- appeler le constructeur approprié.

Vérifier que le mélange a bien été construit en affichant le premier mode de `M_gaussian` (qui peut être récupéré avec l'opérateur de la question 4).

2.4 Somme de deux mélanges

De la même manière que les populations peuvent être regroupées, nous pouvons définir un opérateur de somme entre deux objets de type `Melange` . Nous allons supposer que les populations regroupées ont le même nombre d'individus, et que donc la proportion de l'une et de l'autre dans le mélange final est 1/2.

14. Écrire un opérateur de somme entre deux objets de type `Melange<RV>` qui exécute les instructions suivantes :

- concatène les vecteurs de poids des deux mélanges ;
- normalise le vecteur `weights` résultant (qui n'a plus une somme totale 1) en divisant tous ses éléments par 0.5 ;
- crée un vecteur de paramètres `params` qui concatène les paramètres des deux mélanges ;
- renvoie le mélange construit à partir de `params` et `weights` .

Cet opérateur est un prototype de fonction, amie de la classe `Melange` , et le lien d'amitié (*introvertie* ou *extravertie*) est laissé libre.

15. Créer un mélange gaussien `M_gaussian_2` à partir du fichier `"melange_gauss_2.txt"` ***(T)*** en répliquant la question 13.

16. Tester l'opérateur de somme en sommant `M_gaussian_1` et `M_gaussian_2` .