Sorbonne Université — **M1 Mathématiques**

2018–2019 — **4M056 : Programmation en C++**

TP noté 2 : Algorithme de Robbins-Monro

À l'issue du TP, chaque binôme envoie par email ses fichiers `.hpp` et `.cpp`. **Il est impératif de mettre en commentaire, dans tous les fichiers, les nom, prénom et n° d'étudiant de chacun des membres du binôme.**

Description Soit $h : \mathbb{R} \to \mathbb{R}$ une fonction strictement croissante, continue. Soit $\alpha \in \mathbb{R}$ tel que l'équation $h(x) = \alpha$ admette une solution réelle unique x_α. Nous cherchons à avoir une valeur approchée du x_α inconnu de l'équation précédente. Pour cela, nous allons mettre en œuvre l'algorithme stochastique de Robbins-Monro dans sa version la plus élémentaire.

Soit $x_0 \in \mathbb{R}$. Soit (X_n) la suite définie par récurrence :

$$X_{n+1} = X_n - \epsilon_{n+1}(h(X_n) - \alpha + U_{n+1}) \tag{1}$$

où les variables $(U_n)_{n\in\mathbb{N}^*}$ sont des v.a. indépendantes et identiquement distribuées de variance finie et d'espérance nulle, et où la suite $(\epsilon_n)_{n\in\mathbb{N}^*}$ est une suite déterministe de nombres réels strictement positifs qui satisfont les trois conditions suivantes :

$$\epsilon_n \xrightarrow{n\to\infty} 0, \qquad \sum_n \epsilon_n = +\infty, \qquad \sum_n \epsilon_n^2 < +\infty.$$

On peut alors montrer que, si x_0 est suffisamment proche de x_α alors l'algorithme précédent converge presque sûrement vers x_α.

Nous vous proposons dans ce qui suit d'écrire un *template* de fonction `robbins-monro` et de le tester sur plusieurs situations avec différentes suites (ϵ_n) et différents incréments (U_n).

1. Écrire dans un fichier `robbinsmonro.hpp` un modèle de fonction :

```cpp
template <class Func, class DetermSeq, class RandomV, class RNG>
std::pair<double,double> robbins_monro(
                const Func & h, double x_init, double alpha,
                const DetermSeq & epsilon,RandomV & U, RNG & G
                long unsigned N);
```

avec les contraintes suivantes :

- `h` possède une méthode `double operator()(double) const`.
- `epsilon` possède une méthode `double operator()(long unsigned n) const` qui renvoie le réel ϵ_n.
- `U` et `G` peuvent être n'importe quelle loi de v.a. de la STL et n'importe quel générateur de nombres pseudo-aléatoires de la STL. Ils servent à générer les réalisations des v.a. U_n.
- le résultat renvoyé est la paire $(x_N, h(x_N))$ (et nous rappelons qu'on peut créer une paire avec `std::make_pair(a,b)`).

2. Écrire un programme `test1.cpp` dans lequel vous appliquez l'algorithme de Robbins-Monro précédent à la fonction $x \mapsto 1/(1+e^{-x})$ avec $\alpha = 2/3$ avec $\epsilon_n = 1/(n+1)$ et pour les U_n des v.a. réelles uniformes[1] sur $[-0.1; 0.1]$. On prendra également $x_0 = 0$. On souhaite l'affichage final suivant :

1. La STL fournit `std::uniform_real_distribution<double>` qui prend comme constructeur les bornes de l'intervalle.

```
Pour N= 1000000, on obtient x= VALEUR1 (et h(x)= VALEUR2)
Pour N= 10000000, on obtient x= VALEUR3 (et h(x)= VALEUR4)
```

La valeur attendue est $x_\alpha = 0.6931$.

Nous vous déconseillons de poursuivre si vous n'obtenez pas de valeur similaire.

3. L'algorithme de Robbins-Monro fournit un résultat aléatoire. Nous souhaitons donc étudier la variance empirique du résultat obtenu. Compléter le programme `test1.cpp` en reprenant les paramètres de la question précédente et affichant pour $N = 100000$ et $N = 1000000$, la variance empirique de x_N sur $K = 100$ échantillons, autrement dit la quantité :

$$v_{N,K} = \frac{1}{K}\sum_{k=1}^{K}(x_N^{(k)})^2 - \left(\frac{1}{K}\sum_{k=1}^{K}x_N^{(k)}\right)^2$$

où les $x_N^{(k)}$ sont indépendants et obtenus par des appels successifs à `robbins_monro` .

4. L'algorithme de Robbins-Monro marche pour des fonctions h monotones. En particulier, toute combinaison linéaire positive d'exponentielles croissantes est encore croissante. Nous introduisons ainsi, pour tous coefficients $(\alpha_i)_{0\leq i\leq k-1}$ de $\mathbb{R}_+$ et tous coefficients $(\beta_i)_{0\leq i\leq k-1}$ de $\mathbb{R}_+$, la fonction :

$$h_{\alpha,\beta}(x) = \sum_{i=0}^{k-1}\alpha_i \exp(\beta_i x)$$

Afin de décrire ce type de fonctions, nous introduisons la classe suivante dans des fichiers `explin.hpp` et `explin.cpp` :

```
class ExpCombiLin {
private:        unsigned k;
                double * alpha;         //tableau des alpha_i
                double * beta; //tableau des beta_i
public:         ExpCombiLin(unsigned);//constructeur
                double coeff(unsigned) const; //accesseur alpha
                double exponent(unsigned) const;//accesseur beta
                // ajouter mutateurs avec noms identiques
};
```

Le premier constructeur permet d'initialiser la taille, d'allouer la mémoire des deux tableaux dynamiques, d'initialiser tous les α_i à 1 et tous les β_i à i.

Écrire les codes du constructeur, des accesseurs et des mutateurs.

5. Écrire le code de `operator()` pour pouvoir appeler un objet de `ExpCombiLin` comme premier argument de `robbins_monro` .

6. (*rule of three*) Écrire le code du constructeur par copie, du destructeur et de l'opérateur d'affectation.

7. En utilisant la classe `ExpCombiLin` et le *template* `robbins_monro` , résoudre dans un programme `solution.cpp` $1 + 3e^x + e^{2x} = \alpha$ avec $\alpha = 3$. On utilisera pour les v.a. (U_n) des lois normales centrées de variance 0.1. On fera $N = 1000$ itérations à partir de $x = 0$. On fera le calcul trois fois en utilisant $\epsilon_n = C/(n+1)$ pour $C = 0.1$, $C = 0.75$ et $C = 1.$ en affichant dans le terminal x_α et $h(x_\alpha)$. On indiquera en commentaire quel est le meilleur résultat [2].

2. Le résultat officiel est $x_\alpha = -0.5770494522$.