Sorbonne Université
Master 2 Ingénierie mathématique
Programmation en C++

Examen du 25 octobre 2019
Durée : 1h30

— **Tous** *les fichiers rendus devront contenir vos* **nom et prénoms**.
— *Il est très fortement suggéré d'envoyer des fichiers bien organisés, et qui ne créent pas d'erreurs à la compilation.*
— *Les deux parties sont indépendantes entre elles.*

1 Maximum d'un pont Brownien

On appelle *pont Brownien* le processus stochastique $(X_t)_{t\in[0,1]}$ obtenu en conditionnant un mouvement Brownien $(W_t)_{t\geq 0}$ à valoir 0 au temps $t = 1$:

$$\mathcal{L}((X_t)_{t\in[0,1]}) = \mathcal{L}((W_{t\in[0,1]})|W_1 = 0).$$

On cherche à calculer la probabilité $p(b) = \mathbf{P}(\max_{t\in[0,1]} X_t \geq b)$ qu'un pont Brownien dépasse la valeur b à un moment de sa trajectoire.

Écrire dans un fichier `pont.cpp` un programme qui calcule $p(b)$ pour $x \in \{1, 2, 3\}$. Pour cela, on utilisera la relation de récurrence

$$X_{k+1/N} = X_{k/N} \times \frac{N-k-1}{N-k} + \sqrt{\frac{N-k-1}{N(N-k)}} G_q,$$

où les G_k sont des Gaussiennes indépendantes centrées réduite, et on approchera l'évènement $\{\max_{t\in[0,1]} X_t \geq b\}$ par l'évènement (plus petit) $\{\max_{k=0}^{N} X_{k/N} \geq b\}$.

On fera 10^5 simulations pour ce calcul, avec $N = 100$ pas de discrétisation en temps.

2 Temps local d'une marche aléatoire

On considère une suite de variables aléatoires $(Z_n)_{n\geq 1}$ indépendantes de loi uniforme sur $\{-1, 1\}$, et on pose, pour $n \geq 0$

$$X_n = \sum_{k=1}^{n} Z_k.$$

La suite $(X_n)_{n\geq 0}$ est donc une marche aléatoire symétrique sur $\mathbf{Z}$. On s'intéresse au *temps local* de cette marche, défini comme le nombre de fois que la marche est passée en chaque point de $\mathbf{Z}$. On définit donc

$$L_n^a = \sum_{k=0}^{n} \mathbf{1}_{X_k=a},$$

et L_n^a est alors le nombre de passages en a avant l'instant n.

La famille $(L_n^a)_{a\in\mathbf{Z}}$ est une suite indexée par les entiers positifs *et* négatifs. On va donc écrire une classe `fonction` permettant de représenter informatiquement les suites doubles.

1. Dans des fichiers `fonction.hpp` et `fonction.cpp` écrire le code d'une classe `fonction` qui contient comme membres privés :
   — deux pointeurs `gauche` et `droite` qui vont pointer vers les adresses de deux tableaux d'entiers ;
   — deux entiers `ng` et `nd` qui vont représenter le nombre d'entiers stockés respectivement dans les deux tableaux `gauche` et `droite`.
2. Écrire un constructeur par défaut pour cette classe, initialisant les deux tableaux avec une taille de `1` et contenant chacun la valeur `0`.
3. Écrire le code du constructeur par copie, de l'opérateur `=` et du destructeur pour cette classe, en utilisant correctement les instructions `new` et `delete` pour gérer la mémoire.
4. Écrire une surcharge de l'opérateur `[]` permettant d'accéder aux éléments des deux tableaux `gauche` et `droite`. Si `X` est de type `fonction`, on veut que les éléments de `X.droite` soient accessible par `X[0]`, `X[1]`, `X[2]`, etc., et que ceux de `X.gauche` le soient par `X[-1]`, `X[-2]`, `X[-3]`, etc. Si `n` correspond à un entier qui est hors de la capacité mémoire de `gauche` et `droite`, `X[n]` renverra `0`.
5. Écrire une méthode `ajout` qui prend en argument un entier `n` et telle que `X.ajout(n)` augmente de `1` la valeur de `X[n]`. Si besoin, on augmentera la taille du tableau `gauche` ou `droite` en la *multipliant par deux*. En particulier, les tailles de `gauche` et `droite` seront toujours des puissances de 2.
6. Dans un fichier `temps_loc_n.cpp`, écrire un programme qui utilise la classe `fonction` pour écrire dans un fichier `temps_loc_n.dat` toutes les valeurs (a, L_n^a) telles que $L_n^a \neq 0$, pour $n = 1000$.
7. Dans un fichier `temps_loc_a.cpp`, écrire un programme qui utilise la classe `fonction` pour écrire dans un fichier `temps_loc_a.dat` les 1000 premiers termes de la suite $(L_n^0)_{n\in\mathbf{N}}$.