Sorbonne Université
Master 2 Ingénierie mathématique
Programmation en C++

**Examen du 14 janvier 2020**
**Durée : 2 heures**

— **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.*

Notre but est de simuler numériquement, de manière exacte ou approchée, le *modèle d'Ising*, qui est un modèle pour les matériaux ferromagnétiques. Il s'agit de considérer un ensemble de particules placées sur un graphe, chacune pouvant avoir deux magnétisations possibles, notées + et −.

# 1 Particules chargées sur un réseau carré

On considère l'ensemble $E = \Big\{(\varepsilon_{ij})_{0\leq i,j\leq N-1},\ \varepsilon_{ij} \in \{+,-\}\Big\}$, des fonctions de $\{0,\dots,N-1\}^2$ dans $\{+,-\}$. L'ensemble $\{0,\dots,N-1\}^2$ correspond à un réseau de particules magnétisées disposées en carré, et les deux valeurs + et − correspondent aux deux sens de magnétisation possibles pour chaque particule. On peut aussi dire que les éléments de $E$ sont les matrices carrées de taille $N$ dont les coefficients sont dans $\{+,-\}$.

Deux exemples d'éléments de $E$ avec $N = 6$ sont donnés ci-dessous :

$$\begin{pmatrix} - & - & + & + & + & + \\ + & + & + & - & + & + \\ + & - & + & - & + & + \\ + & - & + & + & + & - \\ - & - & + & - & - & + \\ - & - & + & - & + & + \end{pmatrix} \qquad \begin{pmatrix} - & + & + & + & - & - \\ + & + & - & + & + & + \\ + & - & + & + & + & + \\ + & - & - & + & + & + \\ + & - & - & - & - & + \\ - & - & - & - & - & + \end{pmatrix}$$

On va commencer par définir une classe permettant de manipuler les éléments de $E$.

1. Dans des fichiers `ising.hpp` et `ising.cpp`, écrire une classe `ising`, contenant deux membres privés `N` de type `int`, et `config` de type `std::vector<bool>`. La variable `N` correspondra à la valeur de $N$ et `config` correspondra aux $(\varepsilon_{ij})$ (par convention la valeur `true` correspondra à + et la valeur `false` à −).
2. Écrire un constructeur pour la classe `ising`, prenant en argument une variable de type `int`, qui correspondra à la valeur de `N`, et une variable de type `bool` correspondant à la valeur initiale des magnétisations (et valant `true` par défaut). On donnera alors à `config` une taille de `N*N`, que l'on remplira avec la valeur fournie en argument. La valeur $\varepsilon_{ij}$ sera stockée dans la variable `config[i*N+j]`.
3. Écrire une méthode `taille` qui renvoie la valeur de `N`.
4. Surcharger l'opérateur `()` avec un fonction prenant en argument deux entier `i` et `j` et permettant de lire la valeur `config[N*i+j]`.
5. Surcharger l'opérateur `<<` pour permettre d'afficher une configuration sous la forme d'un tableau de `+` et de `-` comme ci-dessus.
6. Écrire une méthode `remise_a_zero` qui prend en argument une variable de type `bool` et remplit le vecteur `config` avec cette valeur.

7. Écrire une méthode `tirage_aleatoire` qui prend en argument une variable de type `std::mt19937&` et remplit le vecteur `config` avec des valeurs tirées uniformément et indépendamment sur $\{+,-\}$.
8. Ecrire un premier programme `prog1.cpp` qui déclare une variable de type `ising` correspondant à une grille de taille $15 \times 15$, dont les magnétisations sont tirées uniformément sur $\{+,-\}$.

## 2 Dynamique de Glauber

La *dynamique de Glauber* est une chaîne de Markov $(X_n)_{n\in\mathbf{N}}$ à valeur dans l'ensemble $E$ correspondant à une évolution "naturelle" des magnétisations, puisqu'elle favorise les configurations où des sites proches ont même magnétisation. Cette dynamique consiste, partant d'un état $\varepsilon = (\varepsilon_{ij})$, à choisir un site $(i_0, j_0)$ au hasard, puis à modifier aléatoirement sa magnétisation de la manière suivante :

$$\varepsilon'_{i_0,j_0} = \begin{cases} + & \text{avec probabilité } \frac{e^{\beta\delta}}{e^{\beta\delta}+e^{-\beta\delta}}, \\ - & \text{avec probabilité } \frac{e^{-\beta\delta}}{e^{\beta\delta}+e^{-\beta\delta}} = 1 - \frac{e^{\beta\delta}}{e^{\beta\delta}+e^{-\beta\delta}}, \end{cases} \tag{1}$$

où $\delta = \varepsilon_{i_0+1,j_0} + \varepsilon_{i_0-1,j_0} + \varepsilon_{i_0,j_0+1} + \varepsilon_{i_0,j_0-1}$ et $\beta \geq 0$ est un paramètre réel.

Pour faire un pas de la dynamique de Glauber, il suffit donc de tirer aléatoirement un triplet $\Theta = (I_0, J_0, U)$, où $I_0$ et $J_0$ sont choisis uniformément sur $\{0, \ldots, N-1\}$, et $U$ est de loi uniforme sur $[0,1]$. On défini alors $\varepsilon' = \Phi_\Theta(\varepsilon)$ par $\varepsilon'_{I_0J_0} = +$ si $U < \frac{e^{\beta\delta}}{e^{\beta\delta}+e^{-\beta\delta}}$ et $\varepsilon'_{I_0J_0} = -$ sinon. Pour tout site $(i,j) \neq (I_0, J_0)$ on garde $\varepsilon'_{ij} = \varepsilon_{ij}$.

9. Définir une classe `theta` qui contient comme membre privés deux variables `i` et `j` de type `int` et une variable `u` de type `double`, et écrire des méthodes `I`, `J` et `U` permettant de lire les valeurs de `i`, `j` et `u`.
10. Écrire dans la classe `theta` une méthode `tirage_theta` qui prend en argument la valeur de $N$ et un générateur aléatoire et qui tire `i` et `j` de manière uniforme sur $\{0, \ldots, N-1\}$ et `u` de manière uniforme sur $[0,1]$.
11. Écrire dans la classe `ising` une méthode `mise_a_jour` qui prend en argument une variable `beta` de type double (correspondant à $\beta$) et une variable `T` de type `theta` (correspondant à $\Theta$), et applique à l'état courant la dynamique de Glauber : les variables `T.i` et `T.j` donnent la position de la particule modifiée, et la variable `T.u` coresspond à la variable aléatoire $U$ ci-dessus.
12. Écire un deuxième programme `prog2.cpp` qui affiche la configuration $15 \times 15$ obtenue en partant d'un état composé uniquement de `+` et en appliquant 1000 fois la dynamique de Glauber de paramètre $\beta = 0.3$. Dans l'exemple en page 1, la matrice de gauche est obtenu par dynamique de Glauber avec $\beta = 0$, celle de droite avec $\beta = 0.3$.

## 3 L'algorithme de Propp-Wilson

La dynamique de Glauber de la partie précédente admet une unique mesure invariante sur $E$ et on pourrait même montrer que la suite des configurations successives converge en loi vers cette mesure invariante. Toutefois, il est en fait possible de simuler *exactement* une variable aléatoire dont la loi est celle de la loi invariante de la dynamique de Glauber. Pour cela, on simule une suite $(\Theta_n)$ et on considère les fonctions $\Psi_n = \Phi_{\Theta_0} \circ \Phi_{\Theta_1} \circ \ldots \circ \Phi_{\Theta_n}$. On peut alors montrer que si $\Psi_n$ envoie la configuration composée uniquement de $+$ et celle composée uniquement de $-$ sur la même configuration, alors cette dernière configuration suit la loi d'équilibre de la dynamique de Glauber.

13. Surcharger les opérateur `==` et `!=` pour tester si deux configurations sont égales.
14. Écrire une fonction `propp_wilson` prenant en argument une variable `N` de type `int`, une variable `beta` de type `double` et une variable de type `std::mt19937&` qui renvoie une simulation de la mesure invariante de Glauber utilisant la méthode présentée ci-dessus, en calculant $\Psi_n$ jusqu'à ce qu'elle envoie la configuration $+$ et la configuration $-$ sur la même image.