Université Pierre et Marie Curie — **Master M1 "Mathématiques et Applications"**
Période 2 — **Programmation en C/C++**

TD 3 : Tableaux et matrices dynamiques

Le but de ce TD est de se familiariser avec la description dynamique des tableaux et matrices : on déclarera l'adresse d'un tableau comme un pointeur et on utilisera les commandes `malloc` et `free` pour allouer et libérer la mémoire.

1 *Ecrire une fonction* `uniforme` *qui ne prend aucun argument et renvoie un nombre (pseudo-)aléatoire uniformément réparti sur* $]0,1]$ *(à l'aide de la fonction* `rand()`*).*
Tester cette fonction en écrivant le petit programme suivant :
- *demander à l'utilisateur un nombre entier* `size` *(*≥ 1*),*
- *allouer un tableau de taille* `size` *(avec la commande* `malloc`*),*
- *remplir ce tableau par des nombres aléatoires de* $]0,1]$*,*
- *afficher ce tableau (écrire une fonction* `void affiche(double* tableau,int size)`*).*

Pour générer un couple de 2 gaussiennes indépendantes à partir d'un tirage de 2 uniformes, on peut utiliser l'algorithme de Box-Müller : soit U_0 et U_1 deux variables aléatoires indépendantes uniformément distribuées sur $]0,1]$ et Z_0 et Z_1 telles que

$$\begin{cases} Z_0 = \sqrt{-2\ln(U_0)}\cos(2\pi U_1), \\ Z_1 = \sqrt{-2\ln(U_0)}\sin(2\pi U_1). \end{cases}$$

Alors Z_0 et Z_1 sont des variables aléatoires indépendantes suivant la loi normale centrée réduite.

2 *Écrire une fonction* `gaussienne` *qui ne prend aucun argument et renvoie une réalisation d'une gaussienne en utilisant l'algorithme de Box-Müller.*
Tester cette fonction.

Afin de mieux visualiser un échantillon de taille `size`, nous allons fabriquer son histogramme. Pour cela nous définissons le type suivant :

```
typedef struct {
    int          nb_cases;
    double       min;
    double       max;
    double       taille_case;
    int          *data;
} t_histo;
```

qui permet de manipuler un histogramme constitué de `nb_cases` cases, débutant en `min` et finissant en `max`. On considère qu'un réel `x` appartient à la case `k` de l'histogramme `hist` si
`hist.min+k*hist.taille_case <= x < hist.min+(k+1)*hist.taille_cas`.

3 *Écrire une fonction* `histogramme` *dont le prototype est le suivant :*

```
unsigned histogramme(double const *sample, unsigned size, t_histo *histo);
```

Cette fonction remplit l'histogramme `*histo` *en utilisant les données contenues dans* `sample`*, et renvoie le nombre de données non classées (i.e.* $<$ `min` *ou* $\geq$ `max`*).*

4 *Écrire un programme qui remplit un tableau aléatoirement (avec fonction* `uniforme` *ou* `gaussienne`*), construit son histogramme et l'affiche.*
Tous les éléments suivants seront demandés à l'utilisateur : taille de l'échantillon, taille de l'histogramme, bornes min et max de l'histogramme.

Pour écrire dans un fichier, il faut utiliser un pointeur particulier (pointeur sur `FILE`) qu'on initialise (ouvre) par la fonction `fopen` et que l'on ferme après utilisation avec `fclose`. On rappelle que la fonction permettant d'écrire dans le fichier (texte) ouvert est `fprintf`. Exemple :

```
double x = 4 * atan(1.0);
FILE *fichier;
fichier = fopen("test.dat", "w+");
fprintf(fichier, "pi = %g\n", x);
fclose(fichier);
```

5 *Implémenter une fonction* `ecrit_histogramme` *qui écrit un histogramme dans un fichier. Cette fonction prendra 2 arguments : un pointeur (vers valeur constante) sur un histogramme et une chaine de caractère (qui sera le nom du fichier dans lequel vous voulez écrire).*

Pour visualiser simplement votre histogramme, stocké par exemple dans le fichier `hist.dat`, vous pouvez utiliser le programme `gnuplot` en tapant en ligne de commande :

```
gnuplot
// puis une fois dans le programme de visualisation gnuplot:
plot 'hist.dat' w boxes
// puis quit pour quitter le programme gnuplot
```

Nous allons maintenant construire des vecteurs gaussiens corrélés, dont la matrice de covariance aura la forme suivante (exemple complètement artificiel, en général on récupère la matrice de covariance d'après un ensemble de données) :

$$A = (a_{i,j})_{1\leq i,j\leq n}, \quad \text{avec} \quad a_{i,j} = 2\frac{\min(i,j)}{i+j}.$$

6 *Déclarer et initialiser une matrice de covariance A ainsi définie, de taille n (n donné par l'utilisateur).*

Le but est maintenant de simuler des vecteurs gaussiens de $\mathbb{R}^n$ dont la matrice de covariance est A, c'est à dire d'obtenir des réalisations de V où

$$V \sim \mathcal{N}(0, A)$$

Pour cela, on peut décomposer A par la méthode de Cholesky.
La factorisation de Cholesky consiste, pour une matrice symétrique définie positive A, à déterminer une matrice triangulaire inférieure L tel que : $A = LL^*$. Les coefficients $l_{i,j}$ de la matrice L sont définis de la façon suivante :

$$l_{i,i} = \sqrt{a_{i,i} - \sum_{k=1}^{i-1} l_{i,k}^2},$$
$$l_{i,j} = \frac{1}{l_{j,j}}\left(a_{i,j} - \sum_{k=1}^{j-1} l_{i,k}l_{j,k}\right), \quad \text{si } i > j.$$

7 *Implémenter une fonction* `cholesky` *qui réalise cette factorisation. Le prototype pourra être le suivant :*

```
void cholesky(double **A, int n, double **L);
```

Pour obtenir un vecteur gaussien V de loi $\mathcal{N}(0, A)$, il suffit d'effectuer l'opération

$$V = LU \quad \text{où} \quad U \sim \mathcal{N}(0, \mathrm{Id}_n).$$

8 *Écrire une fonction* `mult_mat_vec` *qui multiplie une matrice de taille $m \times n$ par un vecteur (de taille n) et une fonction* `mult_mat_mat` *qui multiplie deux matrices entre elles pour vérifier la fonction* `cholesky` *que vous aurez codée.*
Écrire un petit programme qui génère des vecteurs aléatoires corrélés de matrice de covariance A.