UNIVERSITÉ PARIS-SACLAY 2022-2023 première session d'examen

Licence 3 et Magistère 1ère année de Physique Fondamentale

Examen d'informatique

vendredi 12 mai 2023 8h30 à 11h00 (durée : 2h30)

— *Aucun document n'est autorisé.*

— *Les programmes doivent être écrits en C/C++.*

— *Respecter les notations de l'énoncé : par exemple la variable notée n_t dans l'énoncé devra être écrite* `nt` *dans un programme, n'' devra être écrite* `ns`.

— *L'utilisation de fonctions est laissée à l'appréciation de l'étudiant, sauf dans les cas où elle est imposée.*

— *On suppose que tous les* `#include<iostream> ... #include<math.h> using namespace std;` *nécessaires sont sous-entendus, il n'y a pas à les écrire.*

— *Ne pas faire lire les données au clavier ou dans un fichier par un* `cin >>`*, ou l'équivalent pour un fichier, sauf si cela est explicitement demandé. Par défaut, les données seront donc fournies dans le programme lui-même, par des instructions du type :*

```
dx=0.01; a=1.7; b=1.1;
```

— *Tous les tableaux dont il s'agit dans l'énoncé sont des tableaux dynamiques, à déclarer avec un ou plusieurs* `malloc`.

— *Dans chaque exercice on suppose que le programme principal (s'il y en a un) et les fonctions (s'il y en a) sont écrits dans un unique fichier.*

— *Les 4 exercices sont indépendants les uns des autres.*

— *Le nombre de points (sur un total de 22.5) attribué à chaque question est indiqué entre parenthèses (c'est un barème indicatif). Un point est réservé à la qualité de la présentation, pour un total de 23.5 points.*
Ce barème indicatif ayant été trop généreux (on avait oublié que pour la première fois la durée de l'examen était 2h30 au lieu de 2h), on a finalement renormalisé les notes sur un barème de 21.

— *Le corrigé pourra être consulté sur le site du cours ultérieurement.*

1. Indicateurs statistiques

L'objectif de cet exercice est de définir et implémenter des fonctions qui calculent quelques indicateurs statistiques sur deux ensembles de valeurs réelles.

Question 1 : (1 point)

Écrire une fonction C, ayant comme prototype :

```
void lit_fichierXY(int n, double* vx, double* vy)
```

qui lit `n` paires de valeurs (x, y) réelles (du type double) depuis un fichier nommé *lespaires.txt* et les stocke dans les tableaux `vx` et `vy` fournis en argument. Chaque ligne du fichier contient une paire de valeurs, séparées par un espace, comme dans l'exemple ci-dessous. On supposera que les deux tableaux `vx, vy` ont été alloués et que `n` > 2.

Exemple de lignes du fichier *lespaires.txt* :

```
12.3 8.5
18.6 -14.4
9.9  15.8
...
```

Réponse à la question 1 :

```
void lit_fichierXY(int n, double* vx, double* vy){
  int i;
  fstream fich;
  fich.open("lespaires.txt", ios::in);
  for(i = 0; i < n; i++)
    fich >> vx[i] >> vy[i];
  fich.close();
}
```

Question 2 : (1 point)

Écrire une seconde fonction C, ayant comme prototype :

```
double calc_moy(int n, double* v)
```

qui calcule et retourne la moyenne $\bar{\mathrm{v}}$ des `n` ($n > 2$) valeurs stockées dans le tableau `v` fourni en argument. Pour rappel, $\bar{\mathrm{v}} = \frac{1}{n}\sum_{i=0}^{n-1} \mathrm{v}_i$.

Réponse à la question 2 :

```
double calc_moy(int n, double* v){
  int i;
  double vbar = 0.;
  for(i = 0; i < n; i++)
    vbar += v[i];
  return vbar / n;
}
```

Question 3 : (1 point)

Définir le prototype et écrire le corps d'une troisième fonction C, appelée `calc_max`, qui calcule et retourne la valeur maximum des `n` ($n > 2$) éléments du tableau `v` fourni en argument.

Réponse à la question 3 :

```
double calc_max(int n, double* v){
  int i;
  double max = v[0];
  for(i = 1; i < n; i++)
    if(v[i] > max)
      max = v[i];
  return max;
}
```

Question 4 : (1 point)

Écrire le programme principal (fonction `int main()`) :

— Qui alloue d'abord deux tableaux `vx, vy`, capable chacun de stocker un nombre `n` d'éléments de type double. La valeur de `n` sera fixée au début du `main` et on suppose que le fichier *lespaires.txt* contient au moins `n` lignes.

— Qui appelle les trois fonctions définies ci-dessus pour lire les `n` paires de valeurs depuis le fichier, et pour calculer ensuite la moyenne et le maximum pour les valeurs du seul tableau `vx`.
— Qui affiche la moyenne et le maximum à l'écran.

Réponse à la question 4 :

```
int main(){
  int n = 3;
  double *vx, *vy;
  vx = (double*)malloc(n*sizeof(double));
  vy = (double*)malloc(n*sizeof(double));
  lit_fichierXY(n,vx,vy);
  cout << "Moyenne : " << calc_moy(n,vx) << endl;
  cout << "Maximum : " << calc_max(n,vx) << endl;
  free(vx); free(vy);
  return 0;
}
```

Question 5 : (1 point)

Définir le prototype et écrire le corps d'une fonction C, du nom de `calc_CXY`, qui peut être appelée avec les deux tableaux `vx, vy` et qui calcule (en utilisant si besoin la fonction `calc_moy`) et renvoie la valeur de la corrélation C_{xy} entre les deux ensembles de valeurs [1] :

$$C_{xy} = \frac{1}{n}\sum_{i=0}^{n-1}(x_i - \bar{x})\,(y_i - \bar{y})$$

Réponse à la question 5 :

```
double calc_CXY(int n, double* vx, double* vy){
  int i;
  double moy_x, moy_y, Cxy = 0.;
  moy_x = calc_moy(n,vx);
  moy_y = calc_moy(n,vy);
  for(i = 0; i < n; i++)
    Cxy += (vx[i]-moy_x)*(vy[i]-moy_y);
  return Cxy / n;
}
```

1. Noter que l'expression de C_{xy} ne correspond pas à la définition du coefficient de corrélation statistique, qui doit être normalisé par la dispersion des deux variables. On vous demande d'implémenter l'expression donnée par souci de simplification.

2. Pointeurs

Soit le programme suivant, où il manque les opérateurs `*` et `&` :

```
 1. #include<iostream>
 2. #include<stdlib.h>
 3. using namespace std;
 4. int X = 0;      // compteur global
    // La fonction separer retourne les parties entiere et decimale de a dans i et r
 5. void separer(double a, int i, double r){
 6.   i = (int) a;
 7.   r = a - i;
 8.   X++;
 9. }
    // La fonction declarer alloue et renvoie un tableau a un indice de j entiers
10. int declarer(int j){
11.   int t = (int)malloc(j*sizeof(int));
12.   X++;
13.   return t;
14. }
    // La fonction remplir remplit le tableau t a un indice en utilisant un tableau m a deux indices
15. void remplir(int t, int i, int j){
16.   int k, l;
17.   int m = (int)malloc(i*sizeof(int));
18.   for(k = 0; k < i; k++){
19.     m[k] = declarer(j);
20.     m[k] = t[k*j];
21.     for(l = 0; l < j; l++)
22.       m[k][l] = k + l + 2;
23.   }
24.   X++;
25. }
26. int main(){
27.   int i, j;
28.   double r;
29.   int t;
30.   separer(3.49, i, r);
31.   j = (int)(10*r);
32.   t = declarer(i*j);
33.   remplir(t,i,j);
34.   t[10] += t;
35.   cout << i << " " << j << " " << X << " " << t[10] << endl;
36.   return 0;
37. }
```

Question 1 : (2.5 points)

Ajouter les opérateurs `*` et `&` aux bons endroits et en bon nombre pour créer un programme qui tourne sans messages d'erreur et dont les fonctions font ce qui est indiqué. Vous devez aussi ajouter des parenthèses si elles sont indispensables pour le bon fonctionnement de ces opérateurs. Il est interdit de faire d'autres changements au programme.
Pour répondre à cette question, vous copiez sur votre feuille seulement les lignes qui sont à changer, avec leur numéro. Marquer clairement ce que vous avez ajouté dans ces lignes.

Réponse à la question 1 :

```
 5. void separer(double a, int* i, double* r){
 6.   *i = (int) a;
 7.   *r = a - *i;
10. int* declarer(int j){
11.   int* t = (int*)malloc(j*sizeof(int));
15. void remplir(int* t, int i, int j){
17.   int** m = (int**)malloc(i*sizeof(int*));
20.     m[k] = &(t[k*j]);
```

```
29.   int* t;
30.   separer(3.49, &i, &r);
34.   t[10] += *t;   // *t = t[0]
```

Question 2 : (1.5 points)

Indiquer ce qui sera affiché à l'écran par la commande `cout` dans la ligne 35.
Une ligne du programme contient des instructions inutiles. Identifier cette ligne et expliquer pourquoi.
Enfin, il manque les commandes `free` dans le programme. Ecrire les commandes `free` manquantes et indiquer où il faut les ajouter.

Réponse à la question 2 :

```
3   4   6   8
```

Ligne 19 ne sert à rien, parce que l'adresse mise dans `m[k]` est tout de suite remplacée par une autre adresse en ligne 20. Encore pire, la mémoire réservée par la commande `malloc` pour `m[k]` (dans la fonction `declarer`) va rester bloquée pour rien jusqu'à la fin du programme, parce qu'on aura perdu l'adresse pour la libérer.

Il faut ajouter `free(m);` à la fin de la fonction remplir (avant ou après la commande `X++;`) (mais non pas des `free(m[k])` parce que cela libérerait le tableau `t` ce qu'on ne veut pas faire à cet endroit) ainsi que `free(t);` à la fin de la fonction `main`, juste avant la commande `return 0;`.

Question 3 : (0.5 point)

Quels changements faut-il faire au programme si l'on veut que la fonction `declarer` soit une fonction du type `void` qui utilise un argument pour renvoyer l'adresse de `t` au lieu d'utiliser un `return` ? (La commande `malloc` doit rester dans la fonction.)

Réponse à la question 3 :

```
10. void declarer(int j, int** t){
11.   *t = (int*)malloc(j*sizeof(int));
13. (ligne supprimée)
19.     declarer(j, &(m[k]));
32.   declarer(i*j, &t);
```

Attention, pour pouvoir changer la valeur du (c'est-à-dire l'adresse dans le) pointeur `t` déclaré dans une autre fonction, il faut utiliser un pointeur sur un pointeur comme argument de la fonction `declarer`.

3. Pendule défectueux

On considère un poids ponctuel de masse $m = 1$ kg suspendu au bout d'un fil supposé sans poids et non élastique. Dans un premier temps, aucun frottement ne s'applique à ce pendule, qui a alors comme équation du mouvement :

$$\ddot{\theta} = -\frac{g}{L}\sin\theta \tag{1}$$

avec $g = 9.81$ m.s^{-2} et $L = 0.5$ m. Cette modélisation avec un fil (et non une barre rigide) n'est valable que pour $\theta \in [-\pi/2, \pi/2]$.

Question 1 : (1 point)

1. Déclarer les constantes du problème `m`, `g` et `L` comme variables globales.
2. Écrire une fonction avec le prototype `void systeme(double* q, double t, double* qp, int n)` qui décrit l'équation différentielle (1) dans une forme qui peut être utilisée par la méthode d'Euler ou la méthode de RK4.

Réponse à la question 1 :

```
double m = 1, g = 9.81, L = 0.5;

void systeme(double* q, double t, double* qp, int n){
  qp[0] = q[1];
  qp[1] = -g/L * sin(q[0]);
}
```

Question 2 : (2 points)

Écrire le programme principal (fonction `int main()`) qui calcule la position θ du pendule en $N = 10\,000$ valeurs de t allant de $t_{\text{début}} = 0$ s à $t_{\text{fin}} = 60$ s, avec comme conditions initiales $\theta_0 = \pi/3$ rad et $\dot{\theta}_0 = 0$ rad.s^{-1}. On utilisera la fonction `rk4` de la bibliothèque du Magistère implémentant la méthode de Runge-Kutta d'ordre 4. [2]
Le programme doit s'arrêter automatiquement si le pendule se retrouve avec $\theta < -\pi/2$ ou $\theta > \pi/2$ et afficher un avertissement à l'écran, car notre modélisation physique n'est alors plus valide.
Les valeurs de $\theta(t)$ seront écrites dans un fichier nommé *pendule.res* en mettant un couple t, $\theta(t)$ par ligne.

Réponse à la question 2 :

```
int main(){
  int i, N = 10000, n = 2;
  double t = 0., tfin = 60, dt = tfin/(N-1);
  double* q;
  fstream fich;

  q = (double*)malloc(n*sizeof(double));
  q[0] = M_PI/3.; q[1] = 0.;
  fich.open("pendule.res", ios::out);
  for(i = 0; i < N; i++){
    if(q[0] < -M_PI/2. || q[0] > M_PI/2.){
      cout << "|Angle| trop grand : " << q[0] << endl;
      break;
    }
    // <--- Ici on ajoutera les bouts de code des questions 3 et 7.
    fich << t << " " << q[0] << endl;
    rk4(systeme, q, t, dt, n);
    t += dt;
  }
  fich.close();
  free(q);
  return 0;
}
```

Question 3 : (0.5 point)

Ajouter au programme précédant du code permettant d'afficher à l'écran le couple t, $\theta(t)$ une fois toutes les 100 itérations.

2. Pour rappel, la fonction `rk4` a cinq arguments : un pointeur sur la fonction avec le système des équations différentielles, le tableau `q`, `t`, `dt` et `n` (la dimension du système).

Réponse à la question 3 :

On ajoute à l'endroit indiqué dans la réponse précédente :

```
if(i % 100 == 0)
  cout << t << " " << q[0] << endl;
```

On considère maintenant un pendule défectueux qui comporte un frottement fluide mais seulement pour certains angles. L'équation du mouvement de ce pendule défectueux devient alors :

$$\ddot{\theta} = -\frac{g}{L}\sin\theta - \mathrm{id}(\theta)\, f\, \dot{\theta} \tag{2}$$

avec $f = 0.5\ \mathrm{s}^{-1}$ le coefficient du frottement que subi le pendule lorsque $\theta \in [-\pi/8, \pi/6]$. La fonction indicatrice id est définie par :

$$\begin{aligned} \mathrm{id}(\theta) &= 1 \ \text{ si } \theta \in [-\pi/8, \pi/6] \\ &= 0 \ \text{ sinon.} \end{aligned} \tag{3}$$

Question 4 : (0.5 point)

Implémenter la fonction `id` qui prend en argument θ et retourne un entier 0 ou 1 selon la définition ci-dessus.

Réponse à la question 4 :

```
int id(double theta){
  if(theta > -M_PI/8. && theta < M_PI/6.)
    return 1;
  else
    return 0;
}
```

Question 5 : (0.5 point)

Écrire une fonction `systeme_frott` pour ce pendule défectueux (on pourra prendre modèle sur la fonction écrite à la question 1).

Réponse à la question 5 :

```
double f = 0.5;

void systeme_frott(double* q, double t, double* qp, int n){
  qp[0] = q[1];
  qp[1] = -g/L * sin(q[0]) - id(q[0]) * f * q[1];
}
```

L'énergie totale du pendule vaut :

$$E_{\mathrm{tot}} = \frac{1}{2}m(L\dot{\theta})^2 + mgL(1-\cos\theta) \tag{4}$$

Question 6 : (0.5 point)

Écrire une fonction avec le prototype `double energie_tot(double theta, double d_theta)` qui prend en argument θ et $\dot{\theta}$ et retourne l'énergie totale du pendule.

Réponse à la question 6 :

```
double energie_tot(double theta, double d_theta){
  return 0.5*m*L*L*d_theta*d_theta + m*g*L*(1-cos(theta));
}
```

Question 7 : (0.5 point)

Ajouter une condition d'arrêt au programme principal si l'énergie totale du pendule est inférieure à 0.01 J, avec un avertissement à l'écran.

Réponse à la question 7 :

On ajoute à l'endroit indiqué dans la réponse à la question 2 :

```
if(energie_tot(q[0],q[1]) < 0.01){
  cout << "Energie totale trop petite !" << endl;
  break;
}
```

(Bien sûr, il faut aussi remplacer `systeme` par `systeme_frott` dans l'appel à la fonction `rk4`.)

4. Les méthodes de recuit simulé et de Metropolis

Dans cet exercice, nous allons résoudre l'équation suivante pour un vecteur $\mathbf{x}$ de taille N :

$$M\,\mathbf{x} = \mathbf{y} \tag{5}$$

où M est une matrice carrée (donnée) de taille $N \times N$ et $\mathbf{y}$ un vecteur (donné) de taille N. On utilisera un tableau dynamique à deux indices pour la matrice et des tableaux dynamiques à un indice pour les vecteurs. Les composantes de la matrice et des vecteurs sont des réels du type double.

Pour résoudre cette équation, nous allons utiliser l'algorithme de recuit simulé, qui est basé sur la mécanique statistique. D'abord, nous inventons une fonction "énergie" dont le minimum est la solution de notre problème. Dans notre cas, cette fonction est

$$E(\mathbf{x}) = ||M\mathbf{x} - \mathbf{y}||^2 \tag{6}$$

Question 1 : (0.5 point)

Écrire une fonction qui prend en argument une matrice M et un vecteur (= tableau à un indice) $\mathbf{x}$ et qui calcule le produit $M\,\mathbf{x}$. La fonction doit également prendre en argument un vecteur supplémentaire $\mathbf{v}$ pour stocker le résultat, et la dimension des vecteurs N.

Réponse à la question 1 :

```
void mult(double** M, double* x, double* v, int N){
  int i, j;
  for(i = 0; i < N; i++){
    v[i] = 0.;
    for(j = 0; j < N; j++)
      v[i] += M[i][j] * x[j];
  }
}
```

Question 2 : (1 point)

Écrire une fonction appelée `energie` du type `double` qui renvoie l'énergie du système définie dans l'équation (6). La fonction doit prendre en argument la matrice M, les vecteurs $\mathbf{x}$ et $\mathbf{y}$, et la dimension des vecteurs N. Utiliser la fonction définie dans la question précédente.

Réponse à la question 2 :

```
double energie(double** M, double* x, double* y, int N){
  int i;
  double en = 0.;
  double* v = (double*)malloc(N*sizeof(double));
  mult(M, x, v, N);
  for(i = 0; i < N; i++)
    en += (v[i]-y[i])*(v[i]-y[i]);
  free(v);
  return en;
}
```

Ensuite, nous allons explorer l'espace des configurations du système (c'est-à-dire les vecteurs possibles $\mathbf{x}$) de manière aléatoire. La probabilité d'une configuration dépend de son énergie et de sa température T selon

$$P(\mathbf{x}) = \frac{1}{Z} e^{-\beta E(\mathbf{x})} \tag{7}$$

où Z est une constante (la fonction de partition) et $\beta = \frac{1}{T}$. Pour générer des configurations aléatoires distribuées selon l'équation (7), nous utiliserons la méthode de Metropolis :

1. On choisit un vecteur initial $\mathbf{x}_p$ (p pour "précédent").
2. On génère $\mathbf{x}_n$ (n pour "nouveau") par une petite variation aléatoire de $\mathbf{x}_p$.
3. On calcule le quotient des probabilités, $Q = P(\mathbf{x}_n)/P(\mathbf{x}_p)$.
4. On génère uniformément w dans l'intervalle $[0, 1]$.
5. Si $Q > w$, $\mathbf{x}_n$ devient le nouveau $\mathbf{x}_p$; sinon, on garde $\mathbf{x}_p$.
6. On revient à l'étape 2.

Question 3 : (0.5 point)

Écrire une fonction appelée `alea` qui renvoie un nombre aléatoire uniformément distribué sur l'intervalle $[a, b]$. Utiliser la fonction `drand48()` de la bibliothèque `stdlib.h`.

Réponse à la question 3 :

```
double alea(double a, double b){
  return (b-a)*drand48() + a;
}
```

Question 4 : (0.5 point)

Le nouveau vecteur $\mathbf{x}_n$ (étape 2) doit être généré à partir de $\mathbf{x}_p$ selon $\mathbf{x}_n = \mathbf{x}_p + \delta\mathbf{x}$, où $\delta\mathbf{x}$ est un vecteur dont les composantes δx_i sont uniformément distribuées sur l'intervalle $[-\epsilon, \epsilon]$, avec $\epsilon \ll 1$.
Écrire une fonction appelée `petit_changement`, dont les arguments sont `x_p`, `x_n`, `eps` et `N`, qui calcule un nouveau vecteur $\mathbf{x}_n$ (qui sera alloué dans une question suivante) à partir de $\mathbf{x}_p$ en utilisant la fonction `alea` de la question précédente.

Réponse à la question 4 :

```
void petit_changement(double* x_p, double* x_n, double eps, int N){
  int i;
  for(i = 0; i < N; i++)
    x_n[i] = x_p[i] + alea(-eps,eps);
}
```

Question 5 : (0.5 point)

Écrire une fonction appelée `quotient_prob` qui calcule Q. (Attention, l'appel à la fonction `energie` pour calculer les valeurs de $E_n = E(\mathbf{x}_n)$ et $E_p = E(\mathbf{x}_p)$ se fera dans les fonctions à écrire dans les questions suivantes; la fonction `quotient_prob` reçoit directement ces valeurs en argument.)

Réponse à la question 5 :

```
double quotient_prob(double beta, double E_n, double E_p){
  return exp(-beta*E_n + beta*E_p);
}
```

Question 6 : (1.5 points)

Écrire le corps d'une fonction

```
void evolution(double** M, double* y, double* x_p, double* E_p, double beta, double eps, int N)
```

qui exécute les étapes 2 à 5 de la méthode de Metropolis (une seule itération). Le tableau `x_p` et la variable `E_p` contiennent le vecteur de départ $\mathbf{x}_p$ et son énergie. Si le changement est accepté (étape 5), la fonction doit stocker dans ces variables le nouveau vecteur $\mathbf{x}_n$ et son énergie (sinon la fonction ne change rien). Utiliser les fonctions des questions précédentes.

Réponse à la question 6 :

```
void evolution(double** M, double* y, double* x_p, double* E_p, double beta, double eps, int N){
  int i;
  double Q, w, E_n;
  double* x_n = (double*)malloc(N*sizeof(double));
  petit_changement(x_p, x_n, eps, N);
  E_n = energie(M, x_n, y, N);
  Q = quotient_prob(beta, E_n, *E_p);
  w = drand48();
  if(Q > w){
    *E_p = E_n;
    for(i = 0; i < N; i++)
      x_p[i] = x_n[i];
  }
  free(x_n);
}
```

Nous répétons l'évolution de Metropolis `n_MC` fois. Nous diminuons ensuite la température (on augmente $\beta = \beta + \delta\beta$) et nous répétons l'exploration de l'espace des configurations. Au total, on fait `n_beta` $\times$ `n_MC` appels à la fonction `evolution` définie dans la question 6. Au fur et à mesure que nous diminuons la température, le système évolue vers des configurations à plus faible énergie, et nous finissons par trouver le minimum global et la solution de notre problème.

Question 7 : (3 points)

Écrire la fonction `main` pour compléter le programme qui résout l'équation (5) selon la méthode expliquée ci-dessus en faisant appel aux fonctions précédentes. Voici une liste (non complète!) de ce qu'elle doit faire :

— Définir et initialiser les variables `n_beta` (nombre de changements de température : 1000), `n_MC` (nombre de pas de Metropolis : 5000), `N` (dimension des vecteurs : 2), `beta` (valeur de β, initialisée à 1), `delta_beta` (1) et `eps` (ϵ, voir question 4 : 0.01).

— Définir et allouer un tableau à deux indices `M` et un vecteur (= tableau à un indice) `y`. Leur composantes doivent être demandées à l'utilisateur en utilisant `cin`.

— Définir un vecteur initial `x_p`. Initialiser ses composantes de façon aléatoire uniformément distribuées sur l'intervalle $[-1, 1]$.

— Définir un vecteur auxiliaire `x_min` dans lequel vous stockerez la meilleure solution trouvée (c'est-à-dire celle avec l'énergie minimale ; cette solution sera donc mise à jour chaque fois que vous trouverez une meilleure). Initialisez-le en copiant les composantes de `x_p`.

— Définir deux variables `E_p` et `E_min` (la dernière est pour stocker l'énergie associée à `x_min`) et initialiser les deux avec l'énergie du vecteur `x_p`.

— Écrire les boucles pour faire évoluer le système comme expliqué ci-dessus, en mettant à jour `x_min` et `E_min` si nécessaire.

— À la fin, afficher le vecteur solution et son énergie à l'écran.

Réponse à la question 7 :

```
int main(){
  int i, j, k, n_beta = 1000, n_MC = 5000, N = 2;
  double beta = 1., delta_beta = 1., eps = 0.01, E_p, E_min;
  double** M;
  double *y, *x_p, *x_min;

  srand48(time(NULL));
  M = (double**)malloc(N*sizeof(double*));
  cout << "Donner les " << N*N << " composantes de M :" << endl;
  for(i = 0; i < N; i++){
    M[i] = (double*)malloc(N*sizeof(double));
    for(j = 0; j < N; j++)
      cin >> M[i][j];
  }
  y = (double*)malloc(N*sizeof(double));
  cout << "Donner les " << N << " composantes de y :" << endl;
  for(i = 0; i < N; i++)
    cin >> y[i];
  x_p = (double*)malloc(N*sizeof(double));
  x_min = (double*)malloc(N*sizeof(double));
  for(i = 0; i < N; i++){
    x_p[i] = alea(-1,1);
    x_min[i] = x_p[i];
  }
  E_p = energie(M, x_p, y, N);
  E_min = E_p;
  for(i = 0; i < n_beta; i++){
    for(j = 0; j < n_MC; j++){
      evolution(M, y, x_p, &E_p, beta, eps, N);
      if(E_p < E_min){
        E_min = E_p;
        for(k = 0; k < N; k++)
          x_min[k] = x_p[k];
      }
    }
    beta += delta_beta;
  }
  cout << "Solution pour x : ";
  for(i = 0; i < N; i++)
    cout << x_min[i] <<  " ";
  cout << endl;
```

```
  cout << "L'energie minimale trouvee est " << E_min << endl;
  free(x_p); free(x_min); free(y);
  for(i = 0; i < N; i++)
    free(M[i]);
  free(M);
  return 0;
}
```