UNIVERSITÉ PARIS-SACLAY 2020-2021 première session d'examen

Licence 3 et Magistère 1ère année de Physique Fondamentale

Examen d'informatique

jeudi 20 mai 2021 9h00 à 11h00 (2 heures)

— *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.*

— *L'examen consiste en 4 exercices, qui sont indépendants les uns des autres.*

— ***Le nombre de points (sur un total de 20) 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 21 points.***

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

1. Spectre solaire

Le soleil peut être modélisé par un corps noir, c'est-à-dire un corps dont le rayonnement thermique dépend seulement de la température. La formule de la puissance spectrale émise par un corps noir (loi de Planck) est (en $\mathrm{W/m^2/Hz/sr}$) :

$$B(\nu, T) = \frac{2h}{c^2} \frac{\nu^3}{e^{\frac{h\nu}{k_B T}} - 1} \tag{1}$$

avec $h = 6.63 \times 10^{-34}$ J·s la constante de Planck, $c = 3.00 \times 10^8$ m·s^{-1} la vitesse de la lumière et $k_B = 1.38 \times 10^{-23}$ J·K^{-1} la constante de Boltzmann.

Ce rayonnement va devoir traverser l'atmosphère terrestre pour parvenir jusqu'à nous. Nous nous intéressons ici au spectre de la lumière solaire si on l'observe au niveau de la surface terrestre. À cause de l'absorption dans l'atmosphère la puissance spectrale va diminuer :

$$S(\nu) = B(\nu, T) F(\nu) \tag{2}$$

où $S(\nu)$ est la puissance spectrale observée à la surface de la Terre et $F(\nu)$ le coefficient de transmission ($0 \leq F(\nu) \leq 1$).

On a deux tableaux à un indice, `freq` et `F`, de `n_freq` éléments, qui contiennent une liste de fréquences et les coefficients de transmission $F(\nu)$ mesurés à ces fréquences. Les fréquences sont triées dans l'ordre croissant.

Question 1 : (1 point)

Écrire une fonction de prototype `double corps_noir(double nu, double T)` qui prend en entrée une valeur de fréquence `nu` et une température `T`, et qui renvoie la puissance spectrale d'un corps noir à cette température et cette fréquence.

Réponse à la question 1 :

```
double corps_noir(double nu, double T){
  double h=6.63e-34, c=3.e8, k=1.38e-23;
  return 2*h/c/c*nu*nu*nu/(exp(h*nu/k/T)-1);
}
```

Question 2 : (1 point)

Écrire une fonction de prototype

```
void spectre(double T, int n_freq, double* freq, double* F, double* S)
```

qui calcule et stocke dans `S` le spectre $S(\nu)$ mesuré sur Terre (voir équation (2)) pour les `n_freq` fréquences du tableau `freq`. On suppose que le tableau `S` est déjà tout prêt à être utilisé (pas besoin de faire des `malloc` ici).

Réponse à la question 2 :

```
void spectre(double T, int n_freq, double* freq, double* F, double* S){
  int i;
  for(i = 0; i < n_freq; i++)
    S[i] = corps_noir(freq[i],T) * F[i];
}
```

Question 3 : (1 point)

Écrire une fonction, de prototype `void raies_atmo(int n_freq, double* freq, double* F)`, qui trouve les raies d'absorption du spectre de transmission, c'est-à-dire les minima locaux du tableau `F`. On fera en sorte d'afficher chaque minimum local et la fréquence associée dans la console. On suppose que le tableau fourni ne contient jamais deux valeurs de transmission consécutives égales et qu'il n'y a pas de raies d'absorption aux bords du tableau.

Réponse à la question 3 :

```
void raies_atmo(int n_freq, double* freq, double* F){
  int i;
  for(i = 1; i < n_freq-1; i++)
    if(F[i] < F[i-1] && F[i] < F[i+1])
      cout << F[i] << " " << freq[i] << endl;
}
```

Question 4 : (1 point)

Écrire une fonction `puissance` qui prend en entrée le nombre de fréquences `n_freq` et les tableaux `freq` et `S` qui contient le spectre que calcule la fonction `spectre` de la question 2. Cette fonction doit renvoyer la puissance totale du rayonnement solaire transmis par l'atmosphère, en faisant une intégration numérique sur le spectre. On utilisera pour cela la méthode des trapèzes, selon laquelle l'intégrale d'une fonction est approchée par :

$$\int_{x_0}^{x_{N-1}} f(x)dx \simeq \sum_{k=0}^{N-2} (x_{k+1} - x_k) \frac{f(x_{k+1}) + f(x_k)}{2}$$

Réponse à la question 4 :

```
double puissance(int n_freq, double* freq, double* S){
  int i;
  double P=0.;
  for(i = 0; i < n_freq-1; i++)
    P += (freq[i+1]-freq[i])*(S[i+1]+S[i])/2.;
  return P;
}
```

Question 5 : (1 point)

On a un fichier `Transmission.dat` contenant des informations sur les coefficients de transmission de l'atmosphère. Son format est le suivant : d'abord une ligne contenant le nombre de fréquences `n_freq` auxquelles on a mesuré la transmission, puis une ligne par fréquence (dans l'ordre croissant) avec d'abord la fréquence puis le coefficient de transmission.

Écrire une fonction de prototype `int lire_fichier(double* freq, double* F)` qui lit le fichier et remplit les tableaux `freq` et `F` (que vous supposerez tout prêts à l'utilisation, pas besoin de faire des `malloc`) et qui renvoie la valeur de `n_freq` comme valeur de retour de la fonction.

Réponse à la question 5 :

```
int lire_fichier(double* freq, double* F){
  int i, n_freq;
  fstream fich;
  fich.open("Transmission.dat", ios::in);
  fich >> n_freq;
  for(i = 0; i < n_freq; i++)
    fich >> freq[i] >> F[i];
  fich.close();
  return n_freq;
}
```

2. Pointeurs

On considère le gradient d'une fonction scalaire à deux variables $f(x, y)$, qui est un vecteur à deux composantes $\text{grad}f = \nabla f = \left(\frac{df}{dx}, \frac{df}{dy}\right) \in \mathbb{R}^2$. On peut faire une estimation numérique du gradient à l'aide de l'expression ci-dessous, pour un pas h fixé :

$$\nabla f(x, y) = (dfx, dfy) \quad \text{avec} \quad dfx = \frac{f(x+h, y) - f(x, y)}{h} \quad \text{et} \quad dfy = \frac{f(x, y+h) - f(x, y)}{h}$$

La déclaration de la fonction $f(x, y)$ en C s'écrit : `double mafonction(double x, double y);`
(vous admettez que cette fonction existe déjà dans le programme, il n'est pas nécessaire de l'écrire).

Question 1 : (1 point)

Déclarer le prototype d'une fonction C qui accepte 6 arguments, le pointeur de la fonction f, le pas de calcul numérique de la dérivée h, les coordonnées du point (x, y), et renvoie dans les deux derniers arguments les deux composantes du gradient (dfx, dfy) :

```
void grad(double (*f)(double, double), double h, ... x, ... y, ... dfx, ... dfy);
```

où vous devez compléter ce qui manque (les ...). Écrire le corps de cette fonction. Dans la fonction `main`, déclarer les variables et écrire l'appel à la fonction `grad` pour $x = 1., y = 3., h = 0.0005$ et afficher le résultat avec un `cout`.

Réponse à la question 1 :

```
void grad(double (*f)(double, double), double h, double x, double y, double* dfx, double* dfy){
  *dfx = (f(x+h,y)-f(x,y))/h;
  *dfy = (f(x,y+h)-f(x,y))/h;
}

int main(){
  double x=1., y=3., h=0.0005;
  double dfx, dfy;
  grad(mafonction, h, x, y, &dfx, &dfy);
  cout << "Gradient en (" << x << "," << y << ") = (" << dfx << "," << dfy << ")" << endl;
// La suite de la fonction main se trouve dans la question 3.
```

Question 2 : (1 point)

Soient `a, b` deux doubles, `aa, bb` deux tableaux à un indice de doubles et `aaa, bbb` deux tableaux à deux indices de doubles. Lesquelles des expressions suivantes sont correctes si utilisées comme les deux derniers arguments dans un appel à la fonction `grad` ?

1. `a, b`
2. `*a, *b`
3. `&a, &b`
4. `aa, bb`
5. `*aa, *bb`
6. `&aa, &bb`
7. `aa[0], bb[0]`
8. `&(aa[0]), &(bb[0])`
9. `aaa, bbb`
10. `*aaa, *bbb`
11. `**aaa, **bbb`
12. `aaa[0], bbb[0]`
13. `*(aaa[0]), *(bbb[0])`
14. `&(aaa[0]), &(bbb[0])`
15. `aaa[0][0], bbb[0][0]`
16. `&(aaa[0][0]), &(bbb[0][0])`

Réponse à la question 2 :

Les six expressions suivantes sont correctes : 3, 4, 8, 10, 12, 16.

Question 3 : (2 points)

On souhaite calculer le gradient de la fonction $f(x, y)$ le long d'une courbe définie par les deux fonctions `double courbe_x(double s);` et `double courbe_y(double s);` qui renvoient les deux coordonnées (x, y) d'un point de la courbe identifié par la coordonnée curviligne s. (Vous admettez que ces deux fonctions existent déjà dans le programme, il n'est pas nécessaire de les écrire).

Écrire une fonction nommée `rempli_gradf` qui alloue, remplit et renvoie un tableau contenant les deux composantes du gradient de la fonction f, pour `n` points régulièrement espacés (en s) le long de la courbe, avec $s_{min} \leq s \leq s_{max}$, et pour un pas h de calcul numérique du gradient. Vous utiliserez bien sûr la fonction `grad` de la question 1 pour faire le calcul. Les arguments de la fonction sont un pointeur sur la fonction f ainsi que

`h, n, smin, smax`. On souhaite pouvoir accéder aux valeurs des gradients à l'aide de la valeur de retour de la fonction : `grd = rempli_gradf(...);` où `grd[i][0]` et `grd[i][1]` seraient les deux composantes du gradient au point avec indice i le long de la courbe.

Dans la fonction `main`, déclarer les variables nécessaires et écrire l'appel à la fonction `rempli_gradf` pour $0 \leq s \leq 10$ et $n = 101$.

Réponse à la question 3 :

```
double** rempli_gradf(double (*f)(double, double), double h, int n, double smin, double smax){
  int k;
  double s;
  double ds = (smax-smin)/(n-1);
  double** tab = (double**)malloc(n*sizeof(double*));
  for(k = 0; k < n; k++)                          // Ces deux lignes sont a
    tab[k] = (double*)malloc(2*sizeof(double));  // remplacer dans la question 4
  for(k = 0; k < n; k++){
    s = smin + k*ds;
    grad(f, h, courbe_x(s), courbe_y(s), &(tab[k][0]), &(tab[k][1]));
  }
  return tab;
}

// Suite de la fonction main de la question 1 :
  int n = 101;
  double smin=0., smax=10.;
  double** grd;
  grd = rempli_gradf(mafonction,h,n,smin,smax);
// La fin de la fonction main n'est pas demandee.
```

Question 4 : (0.5 points)

L'allocation mémoire avec des segments de petite taille est coûteuse et peu efficace. La taille du vecteur gradient étant connue (=2), il est possible de réaliser l'allocation du tableau avec deux appels à `malloc` dans `rempli_gradf` (au lieu de $n+1$). Étant donné ces deux commandes :

```
double** tab = (double**)malloc(n*sizeof(double*));
double* pv = (double*)malloc(n*2*sizeof(double));
```

essayez de réécrire la fonction `rempli_gradf` d'une telle façon qu'elle renvoie le même tableau à deux indices que dans la question précédente, en utilisant seulement ces deux commandes `malloc`.

Réponse à la question 4 :

On remplace les deux lignes

```
  for(k = 0; k < n; k++)
    tab[k] = (double*)malloc(2*sizeof(double));
```

par

```
  double* pv = (double*)malloc(n*2*sizeof(double));
  for(k = 0; k < n; k++)
    tab[k] = &(pv[2*k]);
```

3. Oscillateur harmonique relativiste

On considère la dynamique à une dimension d'une particule relativiste de masse m et de charge $e > 0$ en présence d'un potentiel électrostatique $V(q) = \frac{1}{2}\kappa q^2$. Les équations du mouvement qui décrivent la position de la particule $q(t)$ et son impulsion relativiste $p(t)$ sont

$$\begin{cases} \dot{q} = f(p) = \dfrac{pc^2}{\sqrt{m^2c^4 + p^2c^2}}; \\ \dot{p} = g(q) = -e\kappa q. \end{cases} \tag{3}$$

On prend $mc^2 = 0.5$ MeV et $e\kappa = 5.0$ MeV/cm^2 ; la vitesse de la lumière est $c = 30$ cm/ns. Les conditions initiales sont $q(0) = -10^{-2}$ cm et $p(0) = 0$.

On résoudra les équations du mouvement avec la méthode d'Euler semi-implicite, une variante de la méthode d'Euler qui décrit la conservation de l'énergie relativiste

$$H(q,p) = \sqrt{m^2c^4 + p^2c^2} + eV(q) \tag{4}$$

avec une meilleure précision. La méthode d'Euler semi-implicite donne une solution approchée du problème par itérations de :

$$\begin{cases} q_{n+1} = q_n + f(p_n)\Delta t; \\ p_{n+1} = p_n + g(q_{n+1})\Delta t. \end{cases} \tag{5}$$

où Δt est le pas de temps et les fonctions f et g sont définies dans l'équation (3). *Lisez attentivement l'équation : il ne s'agit pas de la méthode d'Euler standard à cause du q_{n+1} dans g !*

Question 1 : (1 point)

Déclarer les trois constantes mc^2, $e\kappa$ et c comme variables globales. Écrire deux fonctions avec prototypes :
— `void ffunc(double* q, double* qp)`
— `void gfunc(double* q, double* qp)`
qui décrivent la première et la deuxième équation différentielle du système (3) et qui peuvent être utilisées dans la fonction `eulersemi` de la question suivante. Le tableau `q` a deux éléments : $q[0] = q$ et $q[1] = p$.

Réponse à la question 1 :

```
double mc2 = 0.5, ekappa = 5., c = 30.;

void ffunc(double* q, double* qp){
  qp[0] = q[1] * c*c / sqrt(mc2*mc2 + q[1]*q[1] * c*c);
}

void gfunc(double* q, double* qp){
  qp[1] = -ekappa * q[0];
}
```

Question 2 : (1.5 points)

Écrire une fonction avec prototype :

```
void eulersemi(void (*func1)(double*, double*), void (*func2)(double*, double*), double* q,
double dt)
```

qui effectue une itération de la méthode d'Euler semi-implicite. *Attention ! Il ne s'agit pas de la méthode d'Euler standard !*

Réponse à la question 2 :

```
void eulersemi(void (*func1)(double*,double*), void (*func2)(double*,double*), double* q, double dt){
  double* qp = (double*)malloc(2*sizeof(double));
  func1(q, qp);
  q[0] = q[0] + dt * qp[0];
  func2(q, qp);
  q[1] = q[1] + dt * qp[1];
  free(qp);
}
```

Question 3 : (1.5 points)

Écrire le reste du programme pour résoudre les équations du mouvement en utilisant la méthode d'Euler semi-implicite pour 10 000 points entre $t = 0$ et $t = 2\sqrt{e\kappa/m}$ (bords inclus) et qui écrit les valeurs de t (en ns), q (en cm) et p (en MeV · ns / cm) dans un fichier nommé `relatho.res`.

Réponse à la question 3 :

```
int main(){
  int i, Nt = 10000;
  double t = 0., tfin = 2*sqrt(ekappa * c*c / mc2), dt = (tfin-t)/(Nt-1);
  fstream fich;
  double* q = (double*)malloc(2*sizeof(double));

  q[0] = -0.01;
  q[1] = 0.;
  fich.open("relatho.res", ios::out);
  for(i = 0; i < Nt; i++){
    fich << t << " " << q[0] << " " << q[1] << endl; // ligne a remplacer en question 4
    // Ici on rajoute des lignes en question 5
    eulersemi(ffunc,gfunc,q,dt);
    t += dt;
  }
  fich.close();
  free(q);
  return 0;
}
```

Question 4 : (0.5 points)

Comme anticipé, la méthode d'Euler semi-implicite est intéressante car, malgré sa simplicité, elle décrit la conservation de l'énergie avec précision Δt^2. Écrire une fonction qui calcule l'énergie relativiste (4). Proposer une modification du programme qui permet d'écrire dans le fichier `relatho.res` l'énergie à chaque instant de l'évolution temporelle.

Réponse à la question 4 :

```
double energie(double* q){
  return sqrt(mc2*mc2 + q[1]*q[1] * c*c) + 0.5*ekappa*q[0]*q[0];
}
```

Puis il faut remplacer la ligne

```
fich << t << " " << q[0] << " " << q[1] << endl;
```

par

```
fich << t << " " << q[0] << " " << q[1] << " " << energie(q) << endl;
```

Question 5 : (0.5 points)

On essaie de calculer la période d'une oscillation de l'oscillateur harmonique relativiste. Proposer et motiver une modification du programme qui calcule une valeur approchée de la période de l'oscillateur et l'affiche à l'écran.

Réponse à la question 5 :

D'abord il faut rajouter au début de la fonction `main` :

```
int test = 0;
```

Ensuite on rajoute à l'endroit indiqué les lignes suivantes :

```
if(test == 0 && q[0] > 0){
  cout << "Periode = " << 4*t << " ns" << endl;
  // (avec nos conditions initiales on arrive a l'origine (depuis q < 0) apres T/4)
  test = 1;
}
```

Il y a bien sûr aussi d'autres méthodes.

4. Démineur

Le jeu de démineur (minesweeper en anglais) consiste en une grille de `n` × `n` cases. Un certain nombre des cases (choisies aléatoirement) contient des mines. Au début du jeu toutes les cases sont cachées. Le joueur ou la joueuse essaie de localiser les mines en sélectionnant une par une toutes les cases qui ne contiennent *pas* de mine, les rendant ainsi visibles. Le joueur ou la joueuse gagne quand il/elle a rendu visibles toutes les cases ne contenant pas une mine, il/elle perd s'il/elle sélectionne une case contenant une mine. Pour l'aider, chaque case visible qui ne contient pas une mine, affiche un chiffre qui indique le nombre de cases voisines contenant une mine. Chaque case, sauf celles aux bords de la grille, a 8 cases voisines (on inclut les voisines diagonales), donc le chiffre affiché est entre 0 et 8.

Dans cet exercice vous allez programmer une version simplifiée de ce jeu. La grille est représentée par un tableau à deux indices `M` d'entiers. L'interprétation des valeurs entières est comme suivant :

— La valeur 9 veut dire que la case contient une mine.
— Les valeurs 0 à 8 veulent dire que la case ne contient pas de mine elle-même mais qu'il y a 0 à 8 mines dans les cases voisines.
— Enfin, on rajoute 10 à la valeur si la case devient visible.

Finalement les valeurs des entiers sont donc comprises entre 0 et 19.

Question 1 : (0.5 points)

Écrire une fonction qui a comme arguments le tableau `M`, l'entier `n` qui décrit sa taille et une variable du type double `p`. Cette dernière, dont la valeur est comprise entre 0 et 1, donne la probabilité qu'une case contienne une mine. La fonction doit mettre les mines (donc des 9) dans le tableau de façon aléatoire, distribuées uniformément avec la probabilité `p`. La fonction doit également mettre des -1 dans les autres cases (c'est une valeur temporaire, ce qui est pratique pour la suite). À noter que la déclaration et l'allocation du tableau `M` se feront plus tard, dans la fonction `main`, voir la question 4.

Réponse à la question 1 :

```
void mines(int** M, int n, double p){
  int i, j;
  for(i = 0; i < n; i++)
    for(j = 0; j < n; j++)
      if(drand48() <= p)
        M[i][j] = 9;
      else
        M[i][j] = -1;
}
```

Question 2 : (1 point)

Écrire une fonction qui met dans chaque case du tableau ne contenant pas une mine (donc contenant un -1), le nombre de mines dans les cases voisines. N'oubliez pas de traiter correctement les cases aux bords de la grille qui ont moins de voisines.

Réponse à la question 2 :

```
void voisines(int** M, int n){
  int i, j, k, l, s;
  for(i = 0; i < n; i++)
    for(j = 0; j < n; j++)
      if(M[i][j] == -1){
        s = 0;
        for(k = i-1; k <= i+1; k++)
          for(l = j-1; l <= j+1; l++)
            if(k >= 0 && k < n && l >= 0 && l < n)
              if(M[k][l] == 9)  // pas besoin d'exclure le cas k,l=i,j, parce que M[i][j]=-1
                s += 1;
        M[i][j] = s;
      }
}
```

Question 3 : (0.5 points)

Écrire une fonction qui affiche le tableau `M` à l'écran, en forme de matrice. Chaque case encore cachée doit être représentée par un astérisque `*`, chaque mine visible par un `X` et toutes les autres cases visibles doivent afficher un chiffre de 0 à 8 pour le nombre de mines voisines. N'oubliez pas que les cases visibles contiennent leur valeur initiale plus 10.

Réponse à la question 3 :

```
void affichage(int** M, int n){
  int i, j;
  for(i = 0; i < n; i++){
    for(j = 0; j < n; j++)
      if(M[i][j] < 10)
        cout << "* ";
      else if(M[i][j] == 19)
        cout << "X ";
      else
        cout << M[i][j]-10 << " ";
    cout << endl;
  }
}
```

Question 4 : (2.5 points)

Enfin, pour compléter le programme, écrire la fonction `main`. Elle doit déclarer et allouer le tableau `M` (ainsi que les autres variables nécessaires, bien sûr) et faire appel aux fonctions des questions 1 et 2 pour le remplir. Ensuite elle doit contenir une boucle où le programme demande au joueur ou à la joueuse d'entrer les indices de la case qu'il/elle souhaite dévoiler, met à jour le tableau `M`, puis affiche la grille actuelle à l'écran, et ensuite recommence la boucle. Il faut aussi programmer trois façons pour sortir de la boucle et terminer le programme :

— Une case contenant une mine est choisie, dans ce cas le programme doit afficher un message pour indiquer que le joueur ou la joueuse a perdu.

— Toutes les cases qui ne contiennent pas de mine ont été dévoilées, dans ce cas le programme doit afficher un message pour indiquer que le joueur ou la joueuse a gagné.

— Le joueur ou la joueuse entre des indices hors grille, ce qui veut dire qu'il/elle souhaite arrêter le programme.

Réponse à la question 4 :

```
int main(){
  int i, j, NN, n = 10;
  double p = 0.1;
  int** M = (int**)malloc(n*sizeof(int*));
  for(i = 0; i < n; i++)
    M[i] = (int*)malloc(n*sizeof(int));
  srand48(time(NULL));
  mines(M, n, p);
  voisines(M, n);
  NN = n*n;
  for(i = 0; i < n; i++)
    for(j = 0; j < n; j++)
      if(M[i][j] == 9)
        NN--;
  while(1){
    cout << endl << "Devoiler quelle case ? ";
    cin >> i >> j;
    if(i < 0 || i >= n || j < 0 || j >= n)
      break;
    if(M[i][j] < 10){
      M[i][j] += 10;
      NN--;
    }
    affichage(M, n);
    if(M[i][j] == 19){
      cout << endl << "PERDU !" << endl;
      break;
    }
    if(NN == 0){
      cout << endl << "GAGNE !" << endl;
      break;
    }
  }
```

```
  for(i = 0; i < n; i++)
    free(M[i]);
  free(M);
  return 0;
}
```

Question 5 : (1 point)

Avec la fonction de la question 1, le nombre de mines dans la grille sera en moyenne `p*n*n`, mais il variera d'un jeu à l'autre. Écrire une autre fonction pour mettre les mines dans la grille, si l'on veut un nombre de mines `N` exact dans chaque jeu (toujours distribuées aléatoirement de façon uniforme).

Réponse à la question 5 :

```
void mines_alt(int** M, int n, int N){
  int i, j, k;
  for(i = 0; i < n; i++)
    for(j = 0; j < n; j++)
      M[i][j] = -1;
  for(k = 0; k < N; k++){
    do{
      i = n*drand48();
      j = n*drand48();
    } while(M[i][j] == 9);
    M[i][j] = 9;
  }
}
```