UNIVERSITÉ PARIS-SUD 2017-2018 première session d'examen

Licence 3 et Magistère 1ère année de Physique Fondamentale

Examen d'informatique

lundi 5 mars 2018 13h45 à 15h45

— *Aucun document n'est autorisé.*

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

— *Contrairement aux années précédentes, vous écrirez vos réponses sur les copies vierges fournies comme pour les autres matières et non pas sur les feuilles de l'énoncé.*

— *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 exercices sont indépendants les uns des autres.*

— *Le nombre de points (sur un total de 40) attribué à chaque question est indiqué entre parenthèses (c'est un barème indicatif). Deux points sont réservés à la qualité de la présentation, pour un total de 42 points. La note finale sera le nombre de points total divisé par deux (avec 20 comme note maximale).*

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

1. Oscillateur harmonique en physique statistique

On considère un oscillateur harmonique quantique de pulsation ω en une dimension, dont les niveaux d'énergie E_n sont donnés par

$$E_n = (n + \tfrac{1}{2})\hbar\omega$$

avec $n = 0, 1, 2, \ldots$. La fonction de partition canonique Z est définie par

$$Z = \sum_{n=0}^{\infty} \exp(-\beta E_n)$$

où $\beta = 1/(k_B T)$ avec T la température. Dans les programmes on représentera β par une variable `b`. Dans un calcul numérique on ne peut bien sûr pas sommer jusqu'à l'infini, donc on se limitera à une somme jusqu'à N inclus.

Question 1 : (2 points)

Écrire une fonction, appelée `foncpart`, dont les arguments sont `b` et `N` et qui calcule (de façon numérique) et retourne la valeur de la fonction de partition Z. On déclarera la valeur de $\hbar\omega$ comme une variable globale nommée `hom`.

Réponse à la question 1 :

```
double hom = 1.e-5;  // Cette valeur explicite n'est donnee qu'en question 4

double foncpart(double b, int N){
  int n;
  double Z = 0.;
  for(n = 0; n <= N; n++)
    Z += exp(-b*(n+0.5)*hom);
  return Z;
}
```

Question 2 : (2 points)

Écrire un programme principal (fonction `main`) qui fait appel à cette fonction et qui demande (en utilisant `cin`) à l'utilisateur de donner les valeurs de β, N et $\hbar\omega$ pour ensuite afficher la valeur de Z à l'écran.

Réponse à la question 2 :

```
int main() {
  double b;
  int N;
  cout << "Valeur de beta ? "; cin >> b;
  cout << "Valeur de N ? "; cin >> N;
  cout << "Valeur de hbarre*omega ? "; cin >> hom;
  // hom deja declare comme variable globale en question 1
  cout << "Fonction de partition Z : " << foncpart(b,N) << endl;
  return 0;
}
```

À partir de Z on peut calculer l'énergie moyenne $\bar{E}$ de l'oscillateur avec la formule :

$$\bar{E} = -\frac{\partial}{\partial\beta} \log Z.$$

Numériquement on peut calculer une dérivée avec la formule

$$\frac{\partial f}{\partial x}(x) = \frac{f(x+h) - f(x-h)}{2h}.$$

Question 3 : (2 points)

Écrire une fonction, appelée `energie`, dont les arguments sont `b`, `N` et `h` et qui calcule (de façon numérique) et retourne la valeur de l'énergie moyenne. (Vous pouvez directement faire appel à la fonction `foncpart`, pas besoin d'utiliser un pointeur sur une fonction.)

Réponse à la question 3 :

```
double energie(double b, int N, double h){
  return (-log(foncpart(b+h,N)) + log(foncpart(b-h,N)))/(2.*h);
}
```

On prend maintenant $N = 10$, $h = 0.1$ et $\hbar\omega = 1 \cdot 10^{-5}$.

Question 4 : (2 points)

Écrire un autre programme principal qui crée un tableau à une dimension et le remplit (en faisant appel à la fonction `energie`) avec les valeurs de l'énergie moyenne de l'oscillateur harmonique pour 20 valeurs de β régulièrement espacées entre $0.1/(\hbar\omega)$ et $2.0/(\hbar\omega)$ (bords inclus).

Réponse à la question 4 :

```
int main() {
  double h = 0.1;
  int i, N = 10, NN = 20;
  double* E = (double*)malloc(NN * sizeof(double));
  for(i = 0; i < NN; i++)
    E[i] = energie((i+1)/(10.*hom), N, h);
  // <-- Les lignes de la question 5 sont a rajouter ici
  free(E);
  return 0;
}
```

L'énergie moyenne de l'oscillateur harmonique se calcule aussi de façon analytique, ce qui donne

$$\bar{E} = \hbar\omega \left(\frac{1}{2} + \frac{1}{\exp(\beta\hbar\omega) - 1} \right).$$

Question 5 : (3 points)

Écrire les compléments à apporter au programme précédent pour calculer pour quelle valeur de β la différence relative $|(\bar{E}_{\text{num}} - \bar{E}_{\text{ana}})/\bar{E}_{\text{ana}}|$ entre la solution numérique dans le tableau et la solution analytique pour l'énergie moyenne est maximale. Faire afficher cette différence relative maximale ainsi que la valeur de β correspondante à l'écran.

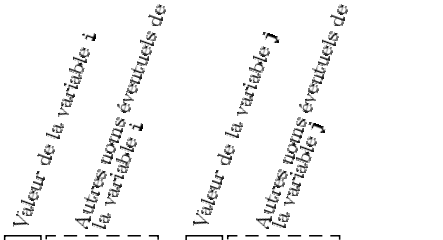
Réponse à la question 5 :

```
double b, Eana, diff, max = -1.;
for(i = 0; i < NN; i++){
  Eana = hom*(0.5 + 1./(exp((i+1)/10.)-1.));
  diff = fabs((E[i]-Eana)/Eana);
  if (diff > max){
    max = diff;
    b = (i+1)/(10.*hom);
  }
}
cout << "Difference relative maximale = " << max << " pour beta = " << b << endl;
```

2. Pointeurs

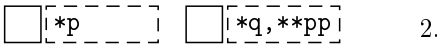
Soit le programme suivant :

```
#include<iostream>
using namespace std;
int main() {
  int i, j;
  int *p, *q;
  int** pp;
  p = &i; q = &j;
```



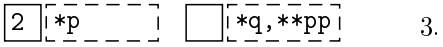
1.

```
  pp = &q;
```



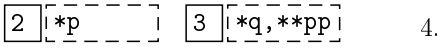
2.

```
  i = 2;
```



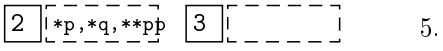
3.

```
  *q = 3;
```



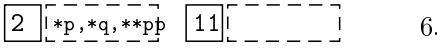
4.

```
  q = p;
```



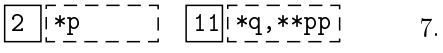
5.

```
  j += **pp + *p + *q + i;
```



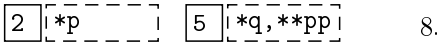
6.

```
  *pp = &j;
```



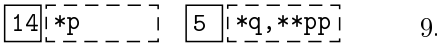
7.

```
  *q /= i;
```



8.

```
  *p = i * (pp[0][0] + p[0]);
```



9.

```
  cout << i << " " << j << endl;
  return 0;
}
```

Question : (7 points)

Pour chaque instruction suivie d'une ligne de cases indiquer le résultat de cette instruction dans les cases. Pour cela il faut donc que vous reproduisiez les quatre cases par ligne (numérotée après les cases) sur votre feuille et que vous les remplissiez (quand la valeur d'une variable n'est pas déterminée, on laisse la case vide).
Quelles seront les valeurs de `i` et `j` affichées par le `cout` ?

Réponse à la question :

```
14   5
```

3. Inflation à deux champs

L'inflation est une brève période d'expansion exponentielle dans l'univers primordial. Cette expansion énorme est engendrée par l'énergie potentielle d'un ou plusieurs champs scalaires. On considère ici un modèle d'inflation à deux champs avec un

potentiel quadratique pour les deux. Les équations du mouvement des deux champs $\phi_1(t)$ et $\phi_2(t)$ sont données par

$$\begin{cases} \ddot{\phi}_1 + 3H\dot{\phi}_1 + m_1^2\phi_1 = 0 \\ \ddot{\phi}_2 + 3H\dot{\phi}_2 + m_2^2\phi_2 = 0 \end{cases}$$

où m_1 et m_2 sont deux constantes (les masses des deux champs, c'est-à-dire les masses des particules de spin-0 associées aux champs) et les points indiquent des dérivées par rapport au temps. Le paramètre de Hubble H est une fonction qui couple les deux équations :

$$H^2 = \frac{1}{6}(\dot{\phi}_1^2 + \dot{\phi}_2^2 + m_1^2\phi_1^2 + m_2^2\phi_2^2).$$

Question 1 : (3 points)

Écrire une fonction avec le prototype `void systeme(double* q, double t, double* qp, int n)` qui décrit le système d'équations différentielles ci-dessus dans une forme qui peut être utilisée par la méthode d'Euler ou la méthode de RK4.

Réponse à la question 1 :

```
double m1 = 1.e-5, m2 = 2.5e-5; // Valeurs explicites donnees en question 3

double H2(double* q){
  return (q[2]*q[2]+q[3]*q[3]+m1*m1*q[0]*q[0]+m2*m2*q[1]*q[1])/6.;
}
// L'utilisation d'une fonction pour H2 n'est pas obligatoire

void systeme(double* q, double t, double* qp, int n){
  double H = sqrt(H2(q));
  qp[0] = q[2];
  qp[1] = q[3];
  qp[2] = -3.*H*q[2] - m1*m1*q[0];
  qp[3] = -3.*H*q[3] - m2*m2*q[1];
}
```

Question 2 : (2 points)

Écrire une fonction avec le prototype `void euler(void (*syst)(double*,double,double*,int), double* q, double t, double dt, int n)` qui fait une itération de la méthode d'Euler.

Réponse à la question 2 :

```
void euler(void (*syst)(double*,double,double*,int), double* q, double t, double dt, int n){
  int i;
  double* qp = (double*)malloc(n * sizeof(double));
  syst(q,t,qp,n);
  for(i = 0; i < n; i++)
    q[i] = q[i] + dt * qp[i];
  free(qp);
}
```

On prend $m_1 = 1 \cdot 10^{-5}$, $m_2 = 2.5 \cdot 10^{-5}$ et $\phi_0 = 30$ comme condition initiale commune pour les deux champs, les deux vitesses initiales étant nulles.

Question 3 : (3 points)

Écrire le reste du programme pour résoudre les équations du mouvement en utilisant la méthode d'Euler pour 1000 points entre $t = 0$ et $t = 3\phi_0/(2m_2)$ (bords inclus) et qui écrit les valeurs de t, ϕ_1 et ϕ_2 dans un fichier nommé `inflation.res`.

Réponse à la question 3 :

```
int main(){
  int i, n = 4, Nt = 1000;
  double f0 = 30., t = 0., tfin = 1.5*f0/m2, dt = (tfin-t)/(Nt-1);
  double* q = (double*)malloc(n * sizeof(double));
  fstream fich;

  q[0] = f0; q[1] = f0; q[2] = 0.; q[3] = 0.;
```

```
  fich.open("inflation.res", ios::out);
  for(i = 0; i < Nt; i++){
    fich << t << " " << q[0] << " " << q[1] << endl;
    euler(systeme,q,t,dt,n);
    // <-- Les lignes de la question 4 sont a rajouter ici
    t += dt;
  }
  fich.close();
  free(q);
  return 0;
}
```

On suppose que l'inflation s'arrête quand le paramètre ϵ de roulement lent devient égal à 1. (À ce moment la période de réchauffement commence et il faudrait prendre en compte le couplage avec d'autres champs dans les équations du mouvement.) Ce paramètre est défini par

$$\epsilon = \frac{\dot{\phi}_1^2 + \dot{\phi}_2^2}{2H^2}.$$

Question 4 : (2 points)

Écrire les compléments à apporter au programme précédent pour que le calcul s'arrête quand soit ϵ devient supérieur ou égal à 1, soit (comme à la question 3) t devient supérieur à $3\phi_0/(2m_2)$. Faire afficher à l'écran la dernière valeur de t avant que le calcul s'arrête.

Réponse à la question 4 :

```
if((q[2]*q[2]+q[3]*q[3])/(2*H2(q)) >= 1){
  cout << t << endl;
  break;
}
```

4. Traversement d'une grille

On considère une grille carrée de points en deux dimensions, de taille (et donc nombre de points) $(2N+1) \times (2N+1)$ où N est un nombre entier positif. On commence au point au milieu et on choisit de façon aléatoire vers lequel des 4 points voisins on va aller (en direction horizontale ou verticale, on ne peut pas se déplacer en direction diagonale). On répète la même procédure pour se déplacer de point à point, avec la contrainte qu'il est interdit d'aller vers un point qui a déjà été visité auparavant. On s'arrête si (1) il ne reste plus de déplacement valide, les quatre voisins étant déjà visités, ou (2) on arrive à un bord de la grille.

Question 1 : (10 points)

Écrire un programme pour faire l'expérience décrite ci-dessus M fois et qui calcule et affiche à l'écran la fraction des cas où l'on est arrivé à un bord de la grille ainsi que la valeur moyenne du nombre de points visités dans ce cas. Vous devez utiliser un tableau d'entiers à deux dimensions pour représenter la grille.

Réponse à la question 1 :

```
int main(){
  int i, j, k, N = 20, dim = 2*N+1, x, y, xx, yy, s, b, sn, M = 100;
  int** t;
  double p;

  srand48(time(NULL));
  t = (int**)malloc(dim * sizeof(int*));
  for(i = 0; i < dim; i++)
    t[i] = (int*)malloc(dim * sizeof(int));
  s = 0; sn = 0;
  for(k = 0; k < M; k++){
    x = N; y = N;
    for(i = 0; i < dim; i++)
      for(j = 0; j < dim; j++)
        t[i][j] = 0;
    t[x][y] = 1;
    while(1){
```

```
      if(t[x+1][y] == 1 && t[x-1][y] == 1 && t[x][y+1] == 1 && t[x][y-1] == 1){
        b = 0;
        break;
      }
      do{
        p = drand48();
        if(p < 0.25)
          {xx = 0; yy = 1;}
        else if(p < 0.5)
          {xx = 1; yy = 0;}
        else if(p < 0.75)
          {xx = 0; yy = -1;}
        else
          {xx = -1; yy = 0;}
      }while(t[x+xx][y+yy] == 1);
      x += xx; y += yy;
      t[x][y] = 1;
      if(x == 0 || x == dim-1 || y == 0 || y == dim-1){
        b = 1;
        break;
      }
    }
    s += b;
    if(b == 1)
      for(i = 0; i < dim; i++)
        for(j = 0; j < dim; j++)
          sn += t[i][j];
  }
  cout << "Fraction : " << (double)s/M << endl;
  if(s > 0)
    cout << "Points visites en moyenne : " << (double)sn/s << endl;
  for(i = 0; i < dim; i++)
    free(t[i]);
  free(t);
  return 0;
}
```

Maintenant on veut refaire la même expérience mais avec une grille circulaire de diamètre $2N + 1$ points au lieu d'une grille carrée. Pour ne pas gaspiller de la mémoire (en particuler quand N est grand) on veut travailler avec un tableau à deux dimensions où l'on n'a pas réservé de la mémoire pour des points qui se trouvent en dehors du cercle. C'est-à-dire que chaque ligne du tableau doit avoir un nombre d'éléments qui correspond au nombre de points dans le cercle à cette ordonnée.

Question 2 : (2 points)

Écrire les commandes pour déclarer et réserver la mémoire pour un tel tableau. Écrire également les commandes pour libérer la mémoire de ce tableau. (On ne demande donc pas les autres changements du programme.)

Réponse à la question 2 :

```
int i, j, N = 20, dim = 2*N+1;
int** t;
t = (int**)malloc(dim * sizeof(int*));
for(i = 0; i < dim; i++){
  j = sqrt(N*N - (i-N)*(i-N));
  t[i] = (int*)malloc((2*j+1) * sizeof(int));
}
for(i = 0; i < dim; i++)
  free(t[i]);
free(t);
```