Reportez votre numéro d'anonymat dans le cadre ci-dessous et en haut à gauche de chaque page de l'énoncé.

A la fin de l'épreuve glissez ce cahier dans la copie double qui vous a été fournie, qui comporte votre numéro d'anonymat et sur laquelle vous aurez écrit et masqué votre nom en rabattant dessus et en collant la languette triangulaire. Vous ne devez rien écrire d'autre sur cette copie double.

Numéro d'anonymat :

UNIVERSITÉ PARIS SUD 2016-2017 première session d'examen

L3 et Magistère 1ère année de Physique Fondamentale

Examen d'informatique mercredi 8 mars 2017 9h30 à 11h30 bât. 337 salle 3

- *Aucun document n'est autorisé.*
- *Les programmes doivent être écrits en C/C++.* ***Les réponses doivent obligatoirement être écrites sur les feuilles de l'énoncé, dans l'espace réservé.*** *Il est fortement conseillé de rédiger d'abord le plan des réponses au brouillon.*
- *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`.
- *Ne pas commencer à répondre à un exercice avant d'en avoir entièrement lu l'énoncé.*
- *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*[1] *et les fonctions*[2] *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. La note finale sera le nombre de points total divisé par deux.*
- *Le corrigé pourra être consulté sur le site du cours ultérieurement.*

1. S'il y en a un.
2. S'il y en a.

Sphère en mouvement

On considère d'abord un ellipsoïde dont l'équation dans le repère à trois dimensions (O, x, y, z) est :

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = 1, \qquad \text{où } a, b, c \text{ sont les demi-axes de l'ellipsoïde.}$$

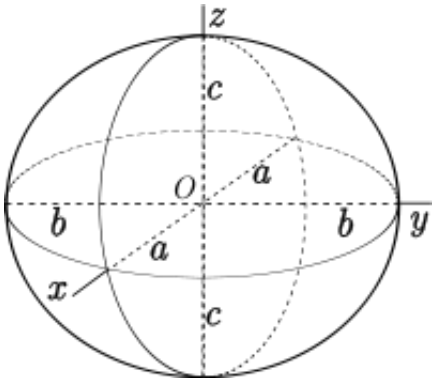


FIGURE 1 – Un ellipsoïde.

Le volume d'un ellipsoïde est donné par la formule $V = \frac{4}{3}\pi abc$.

Question 1 : (2 points)

Écrire une fonction, appelée `volume`, dont les trois arguments sont les trois demi-axes et qui retourne le volume de l'ellipsoïde.
Écrire également 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 a, b, c pour ensuite afficher le volume à l'écran.

Réponse à la question 1 :

```
double volume(double a, double b, double c) {
  return 4. * M_PI * a * b * c / 3.;
}

int main() {
  double a, b, c;
  cout << "Donner les valeurs des trois demi-axes de l'ellipsoide : " << endl;
  cin >> a >> b >> c;
  cout << "Le volume de l'ellipsoide est : " << volume(a,b,c) << endl;
  return 0;
}
```

Dans la relativité restreinte, si un objet a un mouvement rectiligne uniforme avec une vitesse v par rapport à un observateur, cet objet aura l'air déformé pour l'observateur à cause de la contraction de Lorentz. En particulier, la dimension parallèle au mouvement semblera plus courte d'un facteur $1/\gamma$, tandis que les dimensions perpendiculaires au mouvement seront inchangées. Ici γ est défini par

$$\gamma = \frac{1}{\sqrt{1 - \frac{v^2}{C^2}}} \qquad \text{où } C = 2,998 \cdot 10^8 \text{ m/s est la vitesse de la lumière.}$$

On considère maintenant une sphère de rayon R qui se déplace de façon rectiligne uniforme à une vitesse v par rapport à l'observateur.

Question 2 : (2 points)

Écrire une fonction, appelée `vol_deforme`, dont les deux arguments sont le rayon et la vitesse et qui retourne le volume de la sphère comme observé par l'observateur (en faisant appel à la fonction `volume`).

Réponse à la question 2 :

```
const double C = 2.998e8;
```

```
double vol_deforme(double R, double v) {
  double gamma;
  gamma = 1. / sqrt(1. - v*v/(C*C));
  return volume(R, R, R/gamma);
}
```

On prend maintenant $R = 1$ m.

Question 3 : (3 points)

Écrire un programme principal qui crée un tableau à une dimension et le remplit (en faisant appel à la fonction `vol_deforme`) avec les valeurs du volume de la sphère observé pour 1000 valeurs de v régulièrement espacées entre 0 (inclus) et C (pas inclus).

Réponse à la question 3 :

```
int main() {
  int i, n = 1000;
  double R = 1., dv = C / n;      // C variable globale declaree en question 2
  double* tv = (double*)malloc(n * sizeof(double));
  for(i = 0; i < n; i++)
    tv[i] = vol_deforme(R, i*dv);
  // Les lignes de la question 4 sont a rajouter a cet endroit-ci
  free(tv);
  return 0;
}
```

Question 4 : (3 points)

Écrire les compléments à apporter au programme précédent pour calculer (en utilisant les résultats dans le tableau) pour quelle valeur de v (à la précision du tableau près) le volume observé est 1 m^3. Faire afficher cette valeur de v à l'écran.

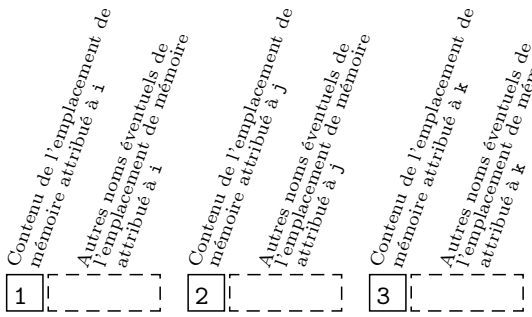
Réponse à la question 4 :

```
  double V = 1.;
  for(i = 0; i < n; i++)
    if(tv[i] < V)
      break;
  cout << "Vitesse pour laquelle le volume observe est " << V << " m^3 : ";
  if(fabs(tv[i-1] - V) < fabs(tv[i] - V))
    cout << (i-1) * dv;
  else
    cout << i * dv;
  cout << " m/s" << endl;
```

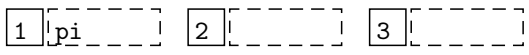
Pointeurs

Soit le programme suivant :

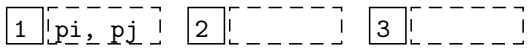
```
#include<iostream>
#include<stdlib.h>
using namespace std;
int main() {
  int i, j, k;
  int *pi, *pj, *pk;
  i = 1; j = 2; k = 3;
```



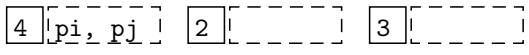

```
  pi = &i;
```



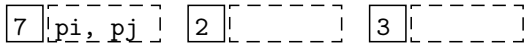

```
  pj = pi;
```




```
  *pi += *pi * i + *pj * j;
```



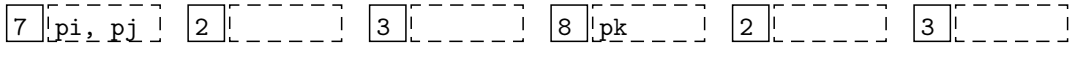

```
  *pj += k;
```



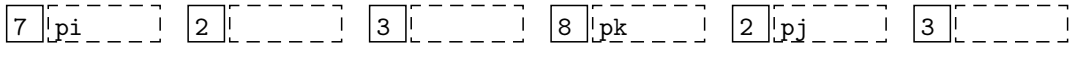

```
  pk = (int*)malloc(3 * sizeof(int));
  pk[0] = i; pk[1] = j; pk[2] = k;
```



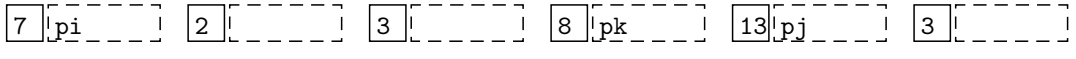
```
  pk[0]++;
```



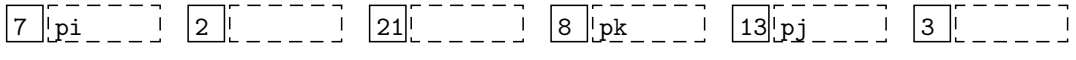

```
  pj = &(pk[1]);
```




```
  *pj = pk[0] + pk[1] + pk[2];
```




```
  k = pk[1] + *pk;
```




```
  cout << i << " " << j << " " << k << endl;
  free(pk);
  return 0;
}
```

Question : (6 points)

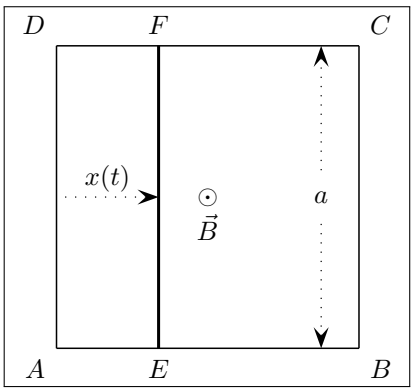
Pour chaque instruction suivie d'une ligne de cases indiquer le résultat de cette instruction dans les cases. Quelles seront les valeurs de `i`, `j` et `k` affichées par le `cout` ?

Réponse à la question :

`7  2  21`

Mouvement d'une barre sur un cadre dans un champ magnétique uniforme

Sur un cadre conducteur carré $ABCD$ de côté a et horizontal est posée une barre EF de masse m qui peut se déplacer sans frottement en restant parallèle au côté AD du cadre. L'ensemble est plongé dans un champ magnétique $\vec{B}$ uniforme, constant et orthogonal au plan du cadre.



Le cadre et la barre ont la même résistance par unité de longueur et la résistance de la barre vaut R.

La position de la barre est repérée par la quantité sans dimension $x(t) = \dfrac{\overline{AE}}{a}$ qui obéit à l'équation différentielle suivante :

$$\ddot{x} = \frac{k\dot{x}}{\dfrac{7}{4} + x - x^2} \quad \text{avec} \quad k = -\frac{B^2a^2}{mR}.$$

La barre est supposée toujours sur le cadre donc : $0 \leq x \leq 1$. La quantité $\dfrac{7}{4} + x - x^2$ s'annule pour $x = \dfrac{1}{2} - \sqrt{2}$, valeur strictement négative, et $x = \dfrac{1}{2} + \sqrt{2}$, valeur strictement supérieure à 1, donc elle ne s'annule pas si la barre est sur le cadre.

Question : (8 points)

Écrire un programme qui, pour $k = -1$ (unités SI), calcule $x(t)$ en $n_p = 1000$ valeurs de t équi-réparties de $t = 0$ à $t = t_{\text{fin}} = 20$ secondes incluses, avec les conditions initiales $x(0) = 0$ et $\dot{x}(0) = 0.5$ s^{-1} par la méthode d'Euler ou celle de Runge-Kutta d'ordre 4 (en utilisant dans ce dernier cas la fonction `rk4` de la bibliothèque des fonctions du Magistère).
Le programme doit s'arrêter si la barre sort du cadre avant l'instant final $t = t_{\text{fin}}$ et afficher alors un avertissement à l'écran.
Les valeurs de $x(t)$ seront écrites dans un fichier nommé *barre.res* en mettant un couple t, $x(t)$ par ligne.

Réponse à la question :

```
void sd(double* q, double t, double* qp, int ne) {
  double k = -1.;
  qp[0] = q[1];
  qp[1] = k*q[1] / (7./4.+q[0]-q[0]*q[0]);
}

int main() {
  int i, ni = 1000, ne = 2;
  double t, dt, tfin = 20.;
  double* q = (double*)malloc(ne * sizeof(double));
  fstream res;
  res.open("barre.res", ios::out);
  dt = tfin / (ni-1);
  q[0] = 0.; q[1] = 0.5;
  for(i = 0; i < ni; i++) {
    if(q[0] < 0 || q[0] > 1) {
      cout << "La barre sort du cadre" << endl;
      break;
    }
    t = i*dt;
```

```
      res << t << " " << q[0] << endl;
      rk4(sd,q,t,dt,ne);
    }
    res.close();
    free(q);
    return 0;
}
```

Le fond diffus cosmologique

Le fond diffus cosmologique (ou CMB, de l'anglais *cosmic microwave background*) est le rayonnement de photons créé au moment où la température de l'univers avait baissée assez pour que les protons et les électrons se combinent en des atomes d'hydrogène neutres. À ce moment-là, quand il avait 380 000 ans, l'univers est devenu transparent pour les photons. À cause de l'expansion de l'univers, ce rayonnement s'est refroidi depuis et a actuellement une température de $T_0 = 2,725$ K (des micro-ondes).
Bien que presque isotrope, il y a des petites fluctuations dans le CMB, avec une amplitude relative de $(T - T_0)/T_0 \sim 10^{-5}$, voir figure. On pense que ces fluctuations ont été générées pendant une période d'inflation dans l'univers primordial. Elles sont les germes de la formation des grandes structures dans l'univers (comme les amas de galaxies) par effondrement gravitationnel.

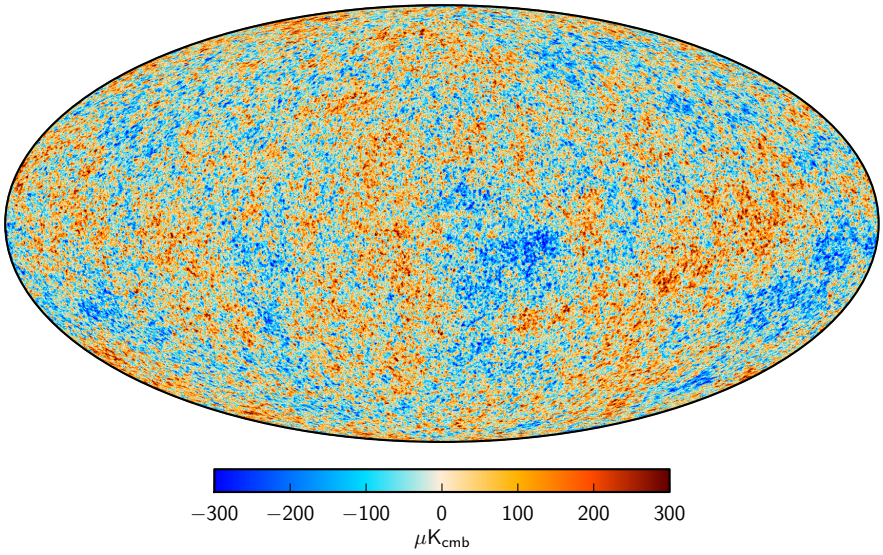


FIGURE 2 – La carte des fluctuations du fond diffus cosmologique comme mesurées par le satellite Planck.

Vu qu'on observe le CMB comme venant d'une surface sphérique qui nous entoure, il est pratique de faire un développement en harmoniques sphériques $Y_{\ell m}$:

$$\frac{T(\theta,\varphi)}{T_0} = \sum_{\ell=2}^{\ell_{\max}} \sum_{m=-\ell}^{+\ell} a_{\ell m} Y_{\ell m}(\theta,\varphi),$$

où les coefficients $a_{\ell m}$ décrivent donc la carte du fond diffus cosmologique dans l'espace harmonique. On commence la somme à $\ell = 2$ et non pas à $\ell = 0$ pour enlever de $T(\theta,\varphi)$ le monopôle (qui est juste la valeur moyenne T_0) et le dipôle (qui est trop contaminé par le mouvement du détecteur par rapport au référentiel du CMB, à cause de la rotation de la Terre autour du Soleil et du Soleil autour du centre de notre galaxie, ainsi que de la vitesse particulière de notre galaxie dans l'univers).

Les fluctuations sont à peu près gaussiennes, ce qui veut dire que leur propriétés statistiques sont bien décrites par le spectre de puissance C_ℓ seul. Sous l'hypothèse d'isotropie statistique (qui fait disparaître la dépendance de m dans cette expression) on peut définir : $C_\ell = \langle |a_{\ell m}|^2 \rangle$. Le spectre de puissance C_ℓ est une quantité réelle qui est representée dans la figure sur la page suivante.
Par contre, les quantités $a_{\ell m}$ sont complexes. Cependant, parce que $T(\theta,\varphi)$ est réel, elles satisfont à la relation

$$a_{\ell,-m} = (-1)^m a^*_{\ell m}.$$

Il suffit donc de calculer les $a_{\ell m}$ avec $m \geq 0$. On en déduit également que les $a_{\ell m}$ avec $m = 0$ sont réels.

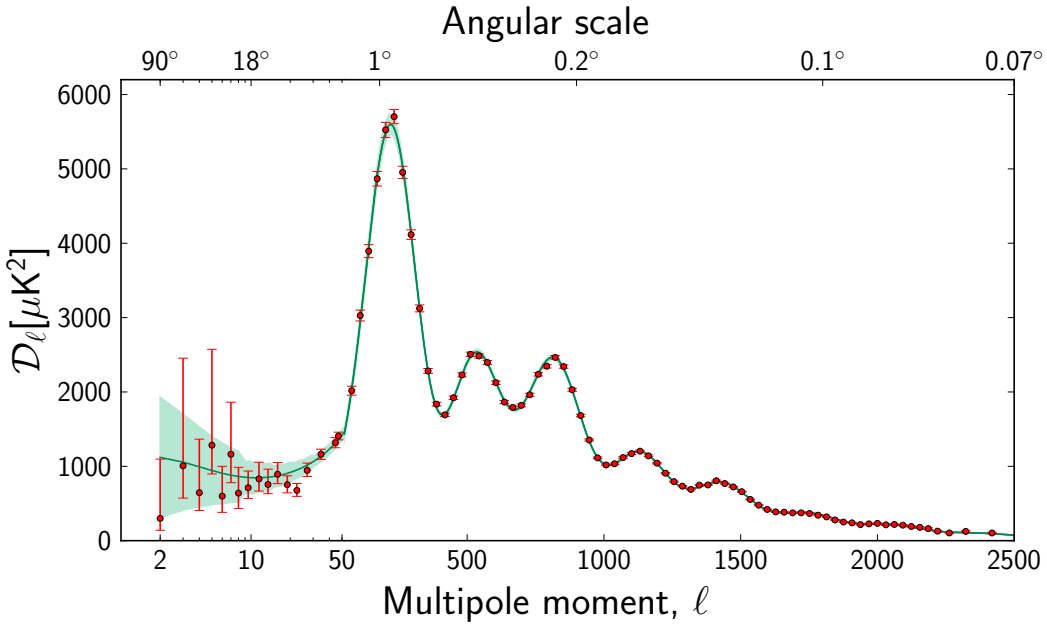


FIGURE 3 – Mesures du spectre de puissance du CMB par le satellite Planck. La quantité D_ℓ est liée à C_ℓ comme suivant : $D_\ell = T_0^2 \ell(\ell+1)C_\ell/(2\pi)$.

Le but de cet exercice est de créer une simulation de la carte du CMB avec le bon spectre de puissance. On commence par la lecture du fichier avec les C_ℓ.

On suppose que le spectre de puissance se trouve dans un fichier du nom *Cl.dat* avec deux colonnes, la première contenant les valeurs de ℓ et la deuxième les valeurs de C_ℓ associées. Les lignes sont ordonnées par ordre croissant de ℓ, commençant par $\ell = 0$ et allant jusqu'à $\ell = \ell_{\max}$ inclus.[3] Toutes les valeurs de ℓ sont présentes, ce qui veut dire que le fichier contient $\ell_{\max}+1$ lignes. Par contre, la valeur de $\ell_{\max}$ (et donc le nombre de lignes) n'est pas connue d'avance.

Question 1 : (4 points)

Écrire une fonction `lire_Cl` qui lit les valeurs de C_ℓ dans le fichier *Cl.dat* et qui retourne, d'une façon de votre choix, deux résultats : la valeur de $\ell_{\max}$ et un tableau à une dimension de $\ell_{\max}+1$ éléments contenant les valeurs des C_ℓ avec $0 \leq \ell \leq \ell_{\max}$.

Réponse à la question 1 :

```
int lire_Cl(double** Cl) {
  int l, lmax;
  double c;
  fstream fich;
  fich.open("Cl.dat", ios::in);
  while(fich >> lmax >> c);
  fich.close();
  *Cl = (double*)malloc((lmax+1) * sizeof(double));
  fich.open("Cl.dat", ios::in);
  while(fich >> l >> c)
    (*Cl)[l] = c;
  fich.close();
  return lmax;
}
/* On pourrait egalement ecrire une fonction double* lire_Cl(int* lmax) ou
   void lire_Cl(int* lmax, double** Cl) ou utiliser une variable globale ou une struct. */
```

Vu que les fluctuations sont gaussiennes, on aura besoin d'un générateur de nombres aléatoires distribués avec une densité de probabilité gaussienne,

$$g(x) \propto \exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right),$$

3. Pour les raisons expliquées ci-dessus, les valeurs de C_ℓ pour $\ell = 0$ et $\ell = 1$ ne sont pas physiques, mais on les inclut dans le fichier et dans le tableau pour simplifier la programmation (valeur de l'indice du tableau égale à la valeur de ℓ).

où μ est la moyenne et σ^2 la variance.

Question 2 : (3 points)

En utilisant la méthode de rejet, écrire une fonction nommée `gaussienne` dont les deux arguments sont la moyenne et la variance et qui retourne une valeur aléatoire tirée au hasard de la distribution gaussienne. Pour cela on supposera que la fonction gaussienne est nulle en dehors de l'intervalle $[\mu - 5\sigma, \mu + 5\sigma]$.

On rappelle que la méthode de rejet pour générer des valeurs aléatoires x de densité de probabilité $g(x)$ dans un intervalle $[a, b]$ marche comme suivant : on tire une valeur aléatoire x de densité uniforme sur $[a, b]$ et une valeur y de densité uniforme sur $[0, M]$ (avec M la valeur supérieure de $g(x)$ sur $[a, b]$), et on accepte x seulement si le point (x, y) est au-dessous de la courbe $g(x)$ (dont la normalisation n'est pas importante pour cette méthode).

Réponse à la question 2 :

```
double gaussienne(double moy, double var) {
  double x, y, sigma;
  sigma = sqrt(var);
  do {
    x = 10.*sigma * drand48() - 5.*sigma;
    y = drand48();
  } while(y > exp(-x*x/(2.*var)));
  return x + moy;
}
/* Remarque : le srand48(time(NULL)); est a inclure au debut de la fonction main
   en question 4, pas ici. */
```

Parce que les $a_{\ell m}$ sont complexes, on les stocke dans deux tableaux à deux dimensions, `alm_reel` pour la partie réelle et `alm_imag` pour la partie imaginaire. Les deux tableaux auront $\ell_{\max} + 1$ lignes (pour la même raison qu'avant on inclut des lignes pour $\ell = 0$ et $\ell = 1$, mais on garde les valeurs des $a_{\ell m}$ correspondants égales à zéro). Parce que les $a_{\ell 0}$ sont réels on pourrait réduire la taille du tableau `alm_imag` d'une colonne, mais il sera plus pratique d'avoir les mêmes dimensions pour les deux tableaux, donc on ne le fera pas (c'est une perte de mémoire négligeable $\propto \ell_{\max}$).

Par contre, ce serait un gaspillage de mémoire important ($\propto \ell^2_{\max}$) de créer des tableaux carrés avec également $\ell_{\max} + 1$ colonnes. Vu qu'on calculera seulement les valeurs des $a_{\ell m}$ avec $0 \leq m \leq \ell$, il sera plus pratique de définir un tableau dont les lignes ont toutes des longueurs différentes, pour exclure du tableau les éléments où m serait supérieur à ℓ.

Question 3 : (3 points)

Écrire une fonction dont le prototype est `double** def_alm(int lmax)` et qui déclare et réserve la mémoire pour un tableau du type comme expliqué ci-dessus et qui retourne un pointeur sur ce tableau.
Écrire également une fonction `free_alm` pour libérer la mémoire réservée.

Réponse à la question 3 :

```
double** def_alm(int lmax) {
  int l;
  double** alm = (double**)malloc((lmax+1) * sizeof(double*));
  for(l = 0; l <= lmax; l++)
    alm[l] = (double*)malloc((l+1) * sizeof(double));
  return alm;
}

void free_alm(double** alm, int lmax) {
  int l;
  for(l = 0; l <= lmax; l++)
    free(alm[l]);
  free(alm);
}
```

Pour simuler une carte du CMB il faut générer des valeurs aléatoires pour tous les $a_{\ell m}$ selon une distribution gaussienne avec moyenne zéro et variance égale à C_ℓ. Attention : pour ces $a_{\ell m}$ qui sont complexes il faut générer la partie réelle et la partie imaginaire indépendamment, avec chacune une variance égale à $C_\ell/2$.

Question 4 : (3 points)

Écrire le programme principal complet qui, en utilisant les fonctions des questions précédentes, génère une simulation du CMB (dans l'espace harmonique).

Réponse à la question 4 :

```
int main() {
  int l, m, lmax;
  double* Cl;
  double** alm_reel, **alm_imag;
  srand48(time(NULL));
  lmax = lire_Cl(&Cl);
  alm_reel = def_alm(lmax);
  alm_imag = def_alm(lmax);
  for(l = 0; l <= lmax; l++)
    for(m = 0; m <= l; m++)
      if (l == 0 || l == 1) {
        alm_reel[l][m] = 0.;
        alm_imag[l][m] = 0.;
      }
      else if(m == 0) {
        alm_reel[l][m] = gaussienne(0., Cl[l]);
        alm_imag[l][m] = 0.;
      }
      else {
        alm_reel[l][m] = gaussienne(0., Cl[l]/2.);
        alm_imag[l][m] = gaussienne(0., Cl[l]/2.);
      }
  // Les lignes de la question 5 sont a rajouter a cet endroit-ci
  free_alm(alm_reel, lmax); free_alm(alm_imag, lmax);
  free(Cl);
  return 0;
}
```

Enfin, on veut vérifier que la carte a bien le bon spectre de puissance.

Question 5 : (3 points)

Écrire les compléments à apporter au programme précédent pour calculer le spectre de puissance de la carte selon la formule

$$C_\ell = \frac{1}{2\ell+1} \sum_{m=-\ell}^{+\ell} |a_{\ell m}|^2$$

et pour afficher à l'écran la fraction des ℓ pour lesquels la différence relative de ce C_ℓ par rapport au C_ℓ d'entrée est supérieure à $2\sqrt{\frac{2}{2\ell+1}}$.

(C'est deux fois la barre d'erreur relative attendue et la fraction devrait donc être autour de 4,55%.)

Réponse à la question 5 :

```
double cl;
int nl = 0;
for(l = 2; l <= lmax; l++) {
  for(m = 0; m <= l; m++)
    if(m == 0)
      cl = pow(alm_reel[l][m], 2);
    else
      cl += 2. * (pow(alm_reel[l][m], 2) + pow(alm_imag[l][m], 2));
      // Remarque bien le facteur 2, pour prendre en compte les m<0 dans la somme
  cl /= 2*l + 1;
  if(fabs((cl-Cl[l])/Cl[l]) > 2.*sqrt(2./(2*l+1)))
    nl++;
}
cout << (double)nl/(lmax-1) << endl;
```