Laboratoire 9

Diffusion et agrégation

9.1 Objectifs:

1. Explorer le lien entre la marche aléatoire sur réseau et la diffusion.
2. Approfondir vos habiletés à manipuler et utiliser des tableaux multidimensionnels en C
3. Bâtir une structure fractale par agrégation limitée par la diffusion

9.2 Rapport de Laboratoire

La rapport doit contenir des réponses, codes et résultats numériques à l'appui, aux questions posées aux §9.3 et 9.4, et doit être dans une semaine, avant le début du prochain lab.

9.3 Marche aléatoire 2D sur réseau

Nous avons vu au chapitre 7 des notes de cours que la marche aléatoire est la représentation à l'échelle microscopique des processus qu'on nomme diffusion du point de vue de l'échelle macroscopique. Du point de vue pratique la situation se corse si l'on désire simuler des systèmes diffusifs où des interactions entre les constituants microscopiques doivent être prises en considération. De telles situations incluent les problèmes d'agrégation couverts à la §7.5 des notes, mais aussi les réactions chimiques en milieux fluides (entre autres). Dans de tels systèmes, les interactions dépendent souvent des distances interparticules, et le calcul de ces distances pour un ensemble de marcheurs aléatoires "classiques" peut devenir très couteux en temps de calcul, quand le nombre de marcheurs devient grand. La **marche aléatoire sur réseau** (voir §7.4 des notes) contourne ce problème en restreignant la position des particules à un réseau cartésien, et une interaction ne se produit que si deux particules se trouvent sur des sites voisins, ce qui est très facile et rapide à calculer.

Ce Labo vise (entre autre) à vous faire explorer les limites de l'équivalence entre la marche aléatoire sur réseau et la diffusion. Vous accomplirez ceci en simulant de telles marches, et en examinant les propriétés statistiques des déplacements parcourus, un peu comme nous avions fait pour la marche alétoire en 1D à la §7.2 des notes. Le résultat-clef était l'augmentation du déplacement quadratique moyen d'un groupe de marcheurs selon la relation

$$\left\langle D_n^2 \right\rangle = n\, s^2 \ , \tag{9.1}$$

où n est le nombre de pas effectués. Il s'agit donc de simuler la marche aléatoire sur réseau 2D. Le canevas global de votre code de simulation est semblable à celui déjà introduit à la §7.2 des notes: une boucle extérieure sur le nombre (`NM`) de marcheurs, dans laquelle est imbriquée une boucle sur le nombre (`NPAS`) de pas effectués par chaque marcheur:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define NM   100
#define NPAS 1000
/* Marche aleatoire sur reseau 2D: NM marcheurs font NPAS pas     */
int main(void)
{
/* Declarations -------------------------------------------------- */
  int pas, x, y, j, k, dfin[NM] ;
/* Executable ---------------------------------------------------- */
  for (j=0 ; j<NM ; j++)              /* boucle sur les marcheurs */
    {
      x=0 ; y=0 ;           /* Tous les marcheurs partent a x=y=0 */
      for (k=0 ; k<NPAS ; k++)              /* boucle sur les pas */
        {
          ...       /* choix d'un pas vers un des 4 sites voisins */
          ...                          /* deplacement du marcheur */
        }
      dfin[j] = x*x+y*y ;    /* calcul du deplacement quadratique */
    }
}
```

Comme on lance ici un marcheur à la fois, on peut considérer que les sites voisins sont toujours libres, donc pas besoin de tester si le déplacement est valide (cf. la Fig. 7.13 des notes), il le sera toujours. La seule difficulté pratique ici est de choisir aléatoirement un de 4 sites voisins; l'approche la plus simple est probablement d'utiliser un test probabiliste (genre `if (1.*rand()/RAND_MAX <= 0.5) {...}`) pour décider si le déplacement se fera en x ou en y, et une fois ce choix fait, utiliser le truc décrit à la §7.2 des notes pour choisir aléatoirement -1 ou $+1$ de manière équiprobable, et effectuer ce pas dans la direction déterminée précédemment. Remarquez que le canevas ci-dessus inclut la définition d'un tableau `dfin` qui calcule le déplacement quadratique à la fin de la marche, pour chacun des marcheurs. Ceci sera utile pour les analyses que vous devrez faire sous peu. Votre protocole de simulation est le suivant:

1. Lancez 1000 marcheurs faisant chacun 1000 pas sur un réseau dans le plan $[x, y]$, et débutant tous à $(x, y) = (0, 0)$. Tracez, avec des appels appropriés aux fonctions PLPLOT, la position de vos 1000 marcheurs aux pas 10, 30, 100, 300 et 1000 (genre, un point-symbole par marcheur, une couleur différente pour chacun des 5 pas de temps).

2. Votre nuage de points a-t-il une forme et une évolution temporelle "raisonnable" pour un processus diffusif? Justifiez votre réponse en quelques lignes.

3. Si la marche aléatoire sur réseau 2D est équivalente à la diffusion, alors le déplacement quadratique moyen $\langle D_n^2 \rangle$ de l'ensemble de vos marcheurs devrait varier comme n. Modifiez votre code pour calculer, à chaque pas de temps de la marche, le $\langle D_n^2 \rangle$, et portez-moi ça en graphique versus n ($1 \leq n \leq 1000$). Est-ce-que ça ressemble bien à la Figure équivalente au chapitre 7 des Notes de cours? Discutez des similarités et différences.

4. Calculez maintenant le déplacement moyen dans la direction x (i.e., la moyenne des positions en x de vos marcheurs), $\langle x_n \rangle$, et portez ceci en graphique en fonction de n, comme précédemment. De toute évidence, $\langle x_n \rangle$ a une gueule tout à fait différente de $\langle D_n^2 \rangle$; comment expliquez-vous ceci?

Finalement, sur la base de tous vos résultats, discutez (un paragraphe d'au plus une douzaine de lignes) dans quelles circonstances la marche 2D sur réseau est ou n'est pas une représentation adéquate de la diffusion, au même niveau que le serait la marche aléatoire classique.

9.4 Agrégation

Dans la seconde partie de ce labo, vous vous bâtirez des dendrites fractales par marche aléatoire sur réseau couplée à un processus d'agrégation. L'idée est la même que celle introduite à la §7.5 des notes: un (ou plusieurs) marcheurs se voient assigner un statut "collant-et-immobile"; lorsqu'un autre marcheur arrive sur un noeud situé au voisinage immédiat (± 1 dans chaque direction), il devient collant et immobile son tour.

Imaginez maintenant que vos marcheurs sont maintenant contenus dans une "boite" de dimension 256×256. Donc, chaque fois qu'un marcheur tente de faire un pas à $x < 1$ ou $x > 256$, il doit "rebondir" dans la direction inverse; et tintin dans la direction y. De plus, on devra maintenant être en mesure de vérifier quand un marcheur situé à un noeud (j, k) a ou non de la compagnie sur un site voisin $(j \pm 1, k \pm 1)$. Pour ce faire il sera utile de définir, en plus de deux tableaux contenant les coordonnées discrètes (j, k) de chaque marcheur, un tableau 2D de taille égale à celle du réseau, dont les éléments auront une valeur '0" pour un site vide et "1" pour un site occupé par un marcheur. Lorsqu'un marcheur se déplace, il faudra donc ajuster ce tableau en conséquence. On assignera aussi une valeur "2" à un marcheur 'collant-et-immobile". Le code C listé à la page suivante effectue ceci. Étudiez bien ce code car vous aurez à le modifier par la suite. Notez en particulier les points suivants:

1. Le canevas global du code inclut encore une fois deux boucles imbriquées, mais cette fois la boucle sur les marcheur est intérieure à la boucle temporelle;

2. L'initialisation consiste à distribuer aléatoirement les marcheurs sur le réseau, et à insérer une valeur "1" à la position correspondante dans le tableau 2D `grille`, pour indiquer que ce site est occupé, les autres éléments de `grille` ayant été préalablement initialisés à zéro. Remarquez comment des élément de tableaux de type `int`, ici `x[j]` et `y[j]`, sont utilisés pour identifier un élément dans le tableau 2D `grille`, opération tout à fait légale en C mais pas dans tous les langages de programmation.

3. Le calcul débute en assignant le statut "collant" à la rangée de N sites situé à à $y = 1$; vous aurez à modifier ceci dans ce qui suit.

4. Un tableau `statut` assigne un code à chaque marcheur, valant 0 si le marcheur est mobile et 1 si il est "collé". Dans la boucle sur les marcheurs, le calcul du pas et le test "voisin" n'est effectué que pour les marcheurs encore mobiles (`status[j]=0`).

5. La boucle temporelle de votre code tourne jusqu'à ce que tous les marcheurs soient "collés" (i.e., `while( ncolle < M ...)`, ou qu'un maximum préétabli d'itérations (`NITERMAX`) soit atteint.

6. Le test voisin utilise deux tableaux `dx`, `dy` de dimension 1 et longueur 8, identifiant la position relative en x et y respectivement des 8 voisins immédiats (haut-bas-gauche-droite et diagonales correspondantes). Ceci permet d'accéder aisément aux huit voisins du marcheur testé avec un seul `if` dans une boucle sur les 8 voisins. Remarquez comment des opérations arithmétiques sur les éléments de `x`, `dx`, etc, servent directement à identifier les éléments du tableau `grille`.

7. Bien que le réseau soit de taille $N \times N$, le tableau 2D `grille` est de taille $(N+2) \times (N+2)$; les rangées et colonnes 0 et $N - 1$ sont des couches "fantômes" requises pour éviter les débordement de tableaux lors du test "voisin" sans avoir à ajouter un tas d'intructions `if`.

```
#include <stdio.h>
#include <stdlib.h>
#define N 256                /* taille du reseau */
#define M 5000               /* nombre de marcheurs */
#define NITERMAX 250000      /* nombre maximal d'iterations temporelles */
int main(void)
{
/* Declarations ============================================================ */
  int x[M], y[M], statut[M], grille[N+2][N+2] ;
  int ncolle, iter, ifound, i, j, k, jj, xnew, ynew ;
  int dx[8]={-1,0,1,1,1,0,-1,-1} ;             /* Stencil des voisins */
  int dy[8]={-1,-1,-1,0,1,1,1,0} ;
/* Executable ============================================================== */
  for (i=0 ; i<N+2 ; i++) { for (j=0 ; j<N+2 ; j++) { grille[i][j]=0 ; }}
  for (j=0 ; j<M ; j++) {                       /* M marcheurs sur le reseau */
    x[j]     =1+floor(1.*N*rand()/RAND_MAX) ;
    y[j]     =1+floor(1.*N*rand()/RAND_MAX) ;
    statut[j]=0 ; grille[x[j]][y[j]]=1 ;        /* initialisation grille */
  }
  for (j=0 ; j<N ; j++) { grille[j][1]=2 ; }    /* sites collants a y=0 */
  ncolle=0 ; iter=0 ;                           /* variables compteur */
  while ( ncolle < M && iter < NITERMAX ) {     /* Boucle temporelle */
    for (j=0 ; j<M ; j++ ) {                    /* boucle sur les marcheurs */
      if ( statut[j] < 1 ) {                    /* les non-colles bougent */
        if ( 1.*rand()/RAND_MAX < 0.5 ) {       /* on bouge en x */
          xnew= x[j] + 2* (floor( 2.*rand()/RAND_MAX ))-1 ;
          if ( xnew < 1 ) { xnew=1 ; }          /* On ne quitte pas le reseau */
          if ( xnew > N ) { xnew=N ; }
          grille[xnew][y[j]]=1 ;
          grille[x[j]][y[j]]=0 ;
          x[j]=xnew ;                           /* deplacement du marcheur */
        } else {                                /* sinon on bouge en y */
          ynew= y[j] + 2* (floor( 2.*rand()/RAND_MAX ))-1 ;
          if ( ynew < 1 ) { ynew=1 ; }          /* on ne quitte pas le reseau */
          if ( ynew > N ) { ynew=N ; }
          grille[x[j]][ynew]=1 ;
          grille[x[j]][y[j]]=0 ;
          y[j]=ynew ;                           /* deplacement du marcheur */
        }
        ifound=0 ;
        for (k=0 ; k<8 ; k++) {                 /* On regarde les 8 voisins */
          if (grille[x[j]+dx[k]][y[j]+dy[k]] == 2 && ifound == 0) {
            grille[x[j]][y[j]]=2 ;              /* ce marcheur colle...  */
            statut[j]=1 ;
            ncolle+=1 ; ifound=1 ;              /* un colle de plus */
          }
        }
      }
    }
    printf ("iteration, nombre de colles %d %d\n",iter,ncolle) ; iter+=1 ;
  }
}
```

Figure 9.1: Code C de base pour l'agrégation limitée par la diffusion.

8. Afin d'accélérer le calcul, deux marcheurs peuvent ici occuper le même site, ce qui n'est pas habituellement le cas avec la marche aléatoire sur réseau.

Je vous laisse comprendre le pourquoi et le comment de la variable `ifound`... et m'expliquer dans votre rapport pourquoi son rôle est essentiel ici au bon fonctionnement du code! Votre première tâche consiste à ajouter les commandes PLPLOT appropriées pour porter en graphique les positions finales de vos marcheurs, et voir la jolie fractale ainsi produite. Ensuite:

1. Ajoutez au code un tableau 1D comptabilisant le nombre total de marcheurs collés, à mesure qu'avance l'itération temporelle. Portez ceci en graphique en fonction de l'itération temporelle; c'est la **courbe de croissance** de votre agrégat.
2. Répétez le calcul pour $N = 2500$, $N = 10000$ et $N = 20000$ marcheurs, toujours sur un réseau $N \times N = 256 \times 256$. Quelles sont les similarités et différences entre les agrégats ainsi produits? Pouvez-vous intuiter lequel parmi ces quatre agrégat a l'indice fractal le plus élevé? le plus petit? Comment les courbes de croissance se comparent-elles?
3. Revenez à $M = 5000$ marcheurs, et changez maintenant la condition initiale du code de manière à ce que le processus d'agrégation ne débute qu'à partir d'un seul site collant situé au centre du réseau; bâtissez ainsi une nouvelle fractale, et calculez de nouveau sa courbe de croissance; bien que la dynamique d'agrégation-diffusion sous-jacente soit la même, cette courbe a une allure passablement différente de la précédente, n'est-ce-pas? Expliquez pourquoi!
4. Finalement, inventez-vous une condition initiale personnelle (distribution spatiale de sites collants), et voyez quelle fractale résulte du processus d'agrégation. N'hésitez pas à travailler sur un plus grand réseau, ou un réseau non-carré, ou un nombre plus ou moins élevé de marcheurs, etc., si l'envie vous prend. Certains de vos collègues de l'an passé ont obtenu des résultats spectaculaires en utilisant une distribution initiale de forme gaussienne centrée au milieu du réseau, avec le bas du réseau "collant", essayez ça en premier si vous manquez d'inspiration...
5. BONUS I: La plus jolie fractale produite à l'étape précédente sera utilisée l'an prochain en arrière-plan aux pages titres des notes de cours et de labo. Si vous voulez participer au concours, glissez une copie sous ma porte de bureau, avec votre nom écrit au verso.
6. BONUS II: Les vraiment enthousiastes peuvent coder la méthode du décompte des boites et calculer ainsi l'indice fractal de leur agrégat personnel.

Et voilà pour le Labo 9. Il ne reste plus qu'un labo... ah, comme la neige a neigé...

Lectures supplémentaires:

Le chapitre 7 des notes de cours devrait être relu attentivement avant de commencer ce labo.

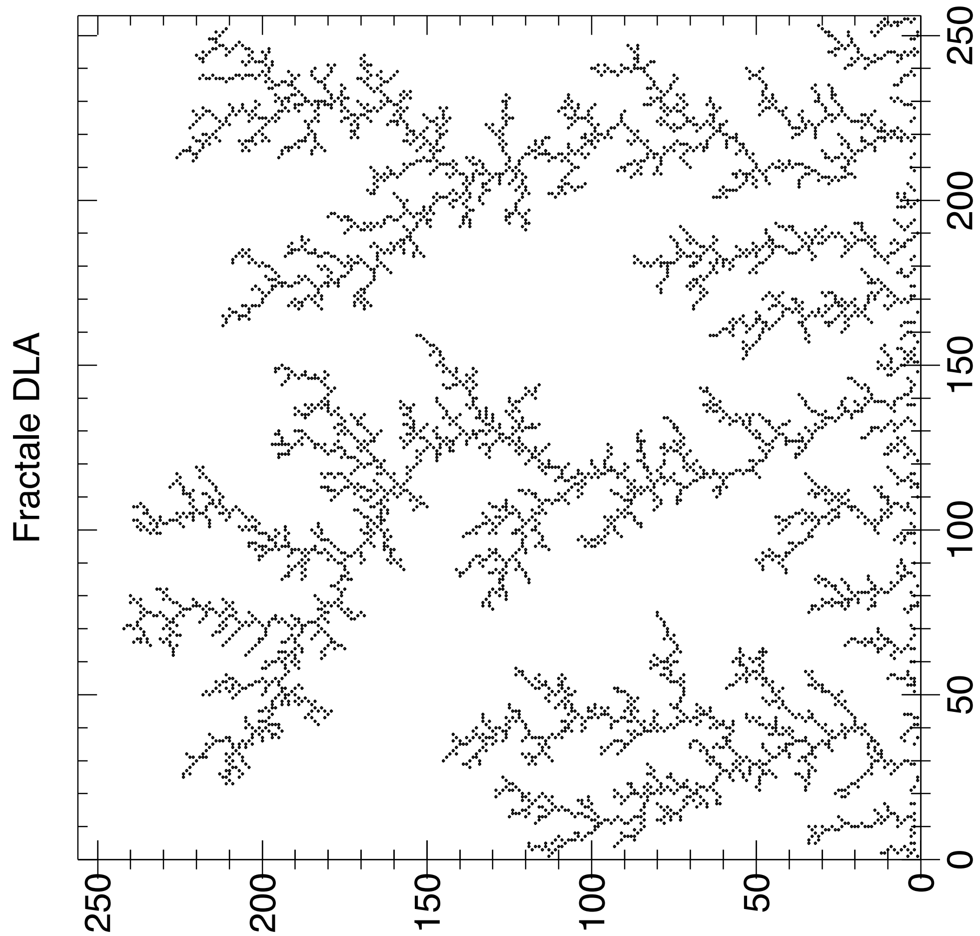

Figure 9.2: Structures dendritiques produites par agrégation sur une surface horizontale "collante" située à $y = 1$. Agrégation de $M = 5000$ marcheurs sur un réseau $N \times N = 256 \times 256$, avec le code C de la Figure 9.1.