Chapitre 9

Complexité

9.1 La complexité

La **complexité** est un concept qui a été et continue d'être apprêté à absolument toutes les sauces, autant dans les milieux scientifiques "sérieux" que dans l'imaginaire populaire; depuis une vingtaine d'années c'est un sujet très "in" chez les intellos de toute allégiances, tout comme le chaos l'a été durant les années 1980. Malgré toute cette agitation intellectuelle, il demeure très difficile de cerner une définition de la complexité qui fasse l'unanimité, au delà de sa triviale redéfinition:

$$\text{complexe} \quad = \quad \text{pas simple}$$

ce qui ramène évidemment le problème à définir ce qui est "simple". Néanmoins, et tout comme moi, vous avez probablement en tête une définition plus ou moins vague et intuitive de ce qui est complexe et de ce qui ne l'est pas. Si l'on choisit de se limiter aux systèmes physiques et/ou naturels, on pourrait commencer par dresser une liste de systèmes "simples" et "complexes". Le tableau 9.1 en donne deux courtes listes, et je compte sur vos suggestions pour qu'il s'allonge au fil des années (envoyez-moi ça par courriel).

Quelles seraient les caractéristiques de ces systèmes complexes en apparence si divers? Si l'on reprend les deux courtes listes du tableau 9.1, on peut en extraire certains points communs, tabulés au Tableau 9.2 ci-dessous. Un des plus évident est probablement le très grand nombre de degrés de liberté (ou composantes en interaction), mais la plus fondamentale est plus subtile à détecter, et touche au comportement *dynamique* des systèmes. Prenons un exemple simple. Si vous savez calculer la force nécessaire pour faire tourner une poulie à laquelle est suspendu un poids, vous savez également comment calculer la force nécessaire pour faire tourner un systèmes de 20 poulies démultipliées et/ou interconnectées pour soulever le même poids; ça pourrait bien être un calcul fastidieux, mais ce n'est vraiment qu'une question de patience, de calme et de concentration. La dynamique du système de 20 poulies se réduit effectivement à la dynamique d'une poulie. C'est une **dynamique réductioniste**. Par contre, tout connaitre sur la molécule de H_2O ne vous permet pas d'anticiper que quelques zillions de molécules de H_2O produiront de la turbulence, tout savoir sur l'électrochimie d'un neurone (ce qui serait déjà extraordinaire) ne vous permet certainement pas d'anticiper qu'un assemblage de plusieurs neurones puisse pondre le *Ich Will* de Rammstein ou *l'Odyssée* d'Homère, pas plus que le calcul du coefficient de friction entre deux grains de sable vous permet de comprendre pourquoi un tas de sable (sec) formera toujours un cône présentant une pente de $\simeq 35$ degrés. Dans ces derniers systèmes, la dynamique à l'échelle globale (fluide, cerveau, tas de sable) est qualitativement différente de la dynamique à l'échelle locale (un H_2O, un neurone, un grain de sable). De plus, cette dynamique globale (et souvent "complexe") **émerge** des interactions locales (et habituellement "simples") entre un très grand nombre d'éléments composant le système. C'est là la caractéristique *sine qua non* des systèmes complexes, tout comme la sensibilité aux conditions initiales définit la nature chaotique d'un système.

Table 9.1: Quelques exemples de systèmes simples et complexes

Systèmes "Simples"	Systèmes "Complexes"
Le pendule	La turbulence dans les fluides
Le thermomètre	Le climat
L'atome d'Hydrogène	Un flocon de neige
Le circuit RLC	Le réseau d'Hydro-Québec
Les orbites planétaires	Les écosystèmes
Une balance	Un tas de sable (sec)

Table 9.2: Quelques caractéristiques des systèmes simples et complexes

Systèmes "Simples"	Systèmes "Complexes"
Relativement peu de degrés de liberté	Énormément de degrés de liberté
Causalité unidirectionnelle	Boucles de rétroaction
Une ou quelques échelles caractéristiques	Multi-échelle ou invariance d'échelle
Prévisibles	Imprévisibles
Dynamique réductioniste	Dynamique émergente

Dans ce chapitre nous allons étudier trois modèles numériques simples de systèmes complexes, soit le modèle dit "Tas-de-Sable" (§9.2; "sandpile model" dans la littérature anglophone), le modèle "Feux-de-Forêt" (§9.4; "Forest Fire model"), et le modèle embouteillage (§9.6, "traffic jam model"). Ces trois modèles nous servirons à illustrer divers types de comportements génériques associés aux systèmes complexes, et nous feront aussi faire connaissance avec le concept de la criticalité auto-régulé (§9.3; "Self-Organized Criticality"), sans nul doute une des grandes percées conceptuelles des dernières quelques décennies en physique statistique; le tout assaisonné d'un petit retour sur l'invariance d'échelle (§9.5).

9.2 Le modèle Tas-De-Sable

Le modèle Tas-de-Sable (ci-après TdS) est un modèle sur réseau évoluant selon des règles simples et locales dans l'espace et le temps. On considère un réseau unidimensionnel de J noeuds sur lesquel on définit une variable S_j^n, où l'indice j numérote le noeud sur le réseau et n mesure l'itération temporelle. Initialement ($n = 0$), on a

$$S_j^0 = 0 \ , \qquad j = 1, ..., J \ . \tag{9.1}$$

Cette variable nodale est est sujette à un mécanisme de forçage selon lequel une petite quantité de sable est ajoutée sur le tas à un noeud choisi au hasard à chaque itération temporelle:

$$S_r^{n+1} = S_r^n + \epsilon \ , \qquad r \in [0, J] \ , \qquad \epsilon \in [0, E] \ , \tag{9.2}$$

où r et ϵ sont extraits de distribution aléatoires uniformes dans les intervalles correspondants, et l'incrément maximal E est un paramètre du modèle. L'analogie habituellement introduite est celle du tas de sable, où S_j^n correspond alors à la hauteur du tas à la position j sur le réseau au "temps" n, et le mécanisme de forçage revient à laisser tomber des grains de sable au hasard quelquepart sur le tas. La conséquence en sera que la hauteur du tas va croitre inexorablement... du moins au début.

Maintenant pour la dynamique du système; à mesure que le tas croit en hauteur, à chaque itération temporelle on calcule la **pente** associée à chaque paire de noeuds $(j, j+1)$:

$$z_j^n = |S_{j+1}^n - S_j^n| \ , \qquad j = 1, ..., J = 1 \ . \tag{9.3}$$

Si cette pente dépasse une valeur critique Z_c prédéfinie, alors la paire de noeud $(j, j+1)$ est déclarée instable, et un processus de redistribution entre en jeu pour ramener le système à une configuration stable à l'itération suivante. Nous utiliserons ici la règle:

$$S_j^{n+1} = S_j^n + (\bar{S} - S_j^n)/2 \ , \qquad S_{j+1}^{n+1} = S_{j+1}^n + (\bar{S} - S_{j+1}^n)/2 \ , \tag{9.4}$$

où

$$\bar{S} = (S_{j+1}^n + S_j^n)/2 \ . \tag{9.5}$$

Je vous laisse vérifier que cette règle est **conservative**, dans le sens que

$$S_j^{n+1} + S_{j+1}^{n+1} = S_j^n + S_{j+1}^n \ , \tag{9.6}$$

et que la quantité δS_j^n de sable déplacée par cette procédure de redistribution est donnée par

$$\delta S_j^n = \frac{z_j^n}{2} \ . \tag{9.7}$$

Mais voilà, le fait d'avoir redistribué du sable entre les noeuds j et $j+1$ change la pente associée aux paires de noeuds $(j-1, j)$ et $(j+1, j+2)$, et peut fort bien pousser l'une ou l'autre de ces pentes au dessus du seuil critique Z_c. Si c'est le cas, la règle de redistribution est appliquée à cette nouvelle paire instable, et on doit conséquemment ensuite vérifier la stabilité —et appliquer la règle de redistribution le cas échéant— des paires voisines, et ainsi de suite. L'effet de tout ça est une **avalanche** de redistributions, déplaçant le sable vers le bas du tas jusqu'à ce que la stabilité soit restaurée sur l'ensemble du réseau. C'est ici que les conditions limites entrent en jeu. On impose au modèle:

$$S_1^n = S_J^n = 0 \ . \tag{9.8}$$

Ceci est équivalent à laisser tomber le sable atteignant le bout du réseau, un peu comme si notre tas de sable se trouvait sur une table de largeur égale à la base du tas. Les conditions limites jouent ici un rôle clef dans la dynamique du système; puisque la règle de redistribution est conservative, et vu l'addition lente mais continuelle de sable sur le tas par le mécanisme de forçage, il est essentiel que du sable puisse être évacué du système si on veut avoir une chance d'atteindre un état stationnaire.

Avant même de simuler quoique ce soit, si on songe un peu à l'effet de tout ça, on en déduirait qu'après un certain temps, le tas prendra une forme triangulaire, avec le sommet au centre du réseau, les pentes de chaque coté égales à la pente critique Z_c, et pour un réseau de taille J on prédirait alors une hauteur maximale égale à $Z_c \times J/2$. Voyons si ces attentes sont réalisées.

Le code C listé à la Figure 9.1 offre une implémentation minimale du modèle TdS. Le tableau `sable[N]` contient ici la quantité de sable à chacun des `N` noeuds du réseau. Il y a deux choses très importantes à remarquer ici:

1. L'itération temporelle se fait en deux sous-étapes séquentielles: on teste d'abord la stabilité à chaque paire de noeud, et on accumule dans un tableau (ici `move`) les quantités de sable qui doivent être transférées pour rétablir la stabilité *sans toucher au tableau* `sable` *à ce stade*. Ce n'est qu'une fois tous les noeuds testés qu'on ajuste le tableau `sable` en conséquence, en y ajoutant le contenu de `move`. Cet ajustement synchrone de la variable nodale est essentiel afin de ne pas introduire de biais directionnel dans le déclenchement ou la propagation des avalanches;

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#define N 101        /* taille du reseau */
#define PENTECRIT 5. /* pente critique */
int main(void)
{
/* Declarations ========================================================= */
  float sable[N], move[N] ;
  float masse, dmasse, av, pente ;
  int j, jj, iter, niter=100000 ;
/* Executable =========================================================== */
  for ( j=0 ; j<N ; j++) { sable[j]=0. ; }       /* Condition initiale */

  for (iter=0 ; iter<niter ; iter++) {         /* iteration temporelle */

     for ( j=0 ; j<N ; j++) { move[j]=0. ; }
     dmasse=0 ;

     for (j=0 ; j<N-1 ; j++ ) {
        pente=fabs(sable[j+1]-sable[j]) ;    /* pente associee a j,j+1 */
        if ( pente >= PENTECRIT ) {     /* la paire j,j+1 est instable */
            av=0.5*(sable[j+1]+sable[j]) ;
            move[j]  =move[j]  +0.5*(av-sable[j]) ;
            move[j+1]=move[j+1]+0.5*(av-sable[j+1]) ;
            dmasse+=pente/2. ;           /* cumul de la masse deplacee */
        }
     }

     if ( dmasse > 0. ) {  /* Il y a eu avalanche; on ajuste le reseau */
        for (j=0 ; j<N ; j++ ) { sable[j]=sable[j]+move[j] ; }
     }
     else {                      /* Il n'y a pas eu avalanche; on force */
        jj=floor(1.*N*rand()/RAND_MAX) ; /* choix aleatoire d'un noeud */
        sable[jj]=sable[jj]+0.5*rand()/RAND_MAX ;    /* ajout de sable */
     }

     sable[0]  =0. ;              /* Imposition des conditions limites */
     sable[N-1]=0. ;

     masse=0. ;                         /* calcul de la masse du reseau */
     for (j=0 ; j<N ; j++ ) { masse+=sable[j] ; }

     printf ("masses totale et deplacee: %f, %f\n",masse,dmasse) ;
   }
}
```

Figure 9.1: Code C pour le modèle TdS sur un réseau en une dimension. Il s'agit ici d'une implémentation minimale, dans le sens qu'on a sacrifié la vitesse d'exécution à la clarté et longueur du code.

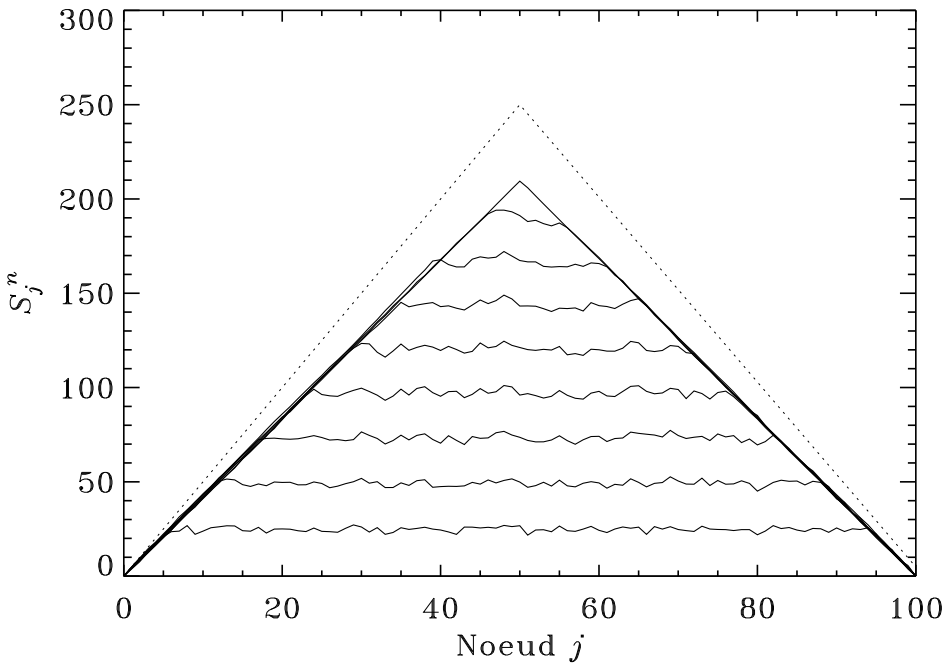

Figure 9.2: Croissance du tas de sable produit par le code C de la Figure 9.1, ici avec $J = 101$, $Z_c = 5$ et $E = 0.1$. Le trait pointillé indique la forme attendue d'un tas parfaitement triangulaire, avec les pentes droites et gauche égales à $\pm Z_c$. Chaque courbe est séparée de la précédente par 50000 itérations.

2. L'ajout de sable ne se produit que si le système est stable partout à cette itération ("stop-and-go sandpile"). Ceci vise à refléter le fait que le temps caractéristique associé au forçage se veut beaucoup plus long que celui caractérisant la propagation des avalanches.

Allons-y pour une petite simulation. La Figure 9.2 illustre la croissance du tas durant les premières 5×10^5 itérations de l'algorithme ci-dessus, sur un réseau de taille $J = 101$ avec $Z_c = 5$ et $E = 0.1$ En raison des conditions limites imposées aux bouts du réseau, la forme conique (attendue!) du tas s'établit d'abord près des bords. Cependant, dans sa configuration finale quasi stationnaire, les deux pentes du tas atteignent une valeur *inférieure* à la pente critique (indiquée en pointillé), et le tas s'en retrouve proportionnellement moins haut qu'anticipé. Ceci est une conséquence de la stochasticité du processus de forçage, qui conduit à des redistributions pour certaines paires de noeuds avant que toutes les paires aient atteint la pente critique; le système stabilise donc à une valeur de pente moyenne plus petite que Z_c, s'approchant de Z_c seulement dans la limite $E \to 0$.

Il sera utile de se définir quelques quantités globales afin de caractériser l'évolution temporelle du réseau. Commençons par la masse, soit la quantité totale de sable contenue dans le tas à l'itération n:

$$M^n = \sum_{j=1}^{J} S_j^n \ . \tag{9.9}$$

La Figure 9.3A montre l'évolution temporelle de cette quantité à partir du début de la simulation. La masse croit initialement de manière linéaire, mais sature éventuellement à une valeur statistiquement stationnaire, mais sujette à fluctuations. La forme que prennent ces fluctuations est particulière, comme on peut le constater sur examen de l'encadré, montrant un zoom sur une petite partie de la séquence dans le régime stationnaire. On y voit la masse croitre

graduellement et approximativement linéairement, mais cette croissance est épisodiquement interrompue par des baisses très rapides, manifestations d'une avalanche ayant fait chuter une certaine quantité de sable du tas. Le pattern en dents-de-scie est une réflexion du fait que le processus de chargement du tas opère d'une manière lente et graduelle, tandis que le processus de vidage est lui rapide et très intermittent. La courbe décrivant la variation de M^n est self-similaire, et on peut lui associer une dimension fractale plus grande que un, un peu comme on l'avait fait pour la fractale de Koch (§7.7).

Une autre quantité d'intérêt est la masse déplacée à l'itération n lors d'une avalanche:

$$\Delta M^n = \sum_{j=1}^{J-1} \delta S_j^n \ , \tag{9.10}$$

où δS_j^n est défini via l'éq. (9.7). Notez que cette dernière quantité n'est *pas* nécessairement égale à $M^{n+1} - M^n$, puisqu'une avalanche n'atteignant pas le bord du réseau ne changera pas la masse du tas, bien qu'elle déplace du sable.

La Figure 9.3B montre la portion de la séquence temporelle de ΔM^n correspondant à l'intervalle couvert par l'encadré de la partie (A). Cette séquence est très **intermittente**, dans le sens que $\Delta M^n = 0$ sauf durant de courts épisodes d'activité correspondant évidemment aux avalanches. Celles-ci se déclenchent de manière irrégulière dans le temps, et ont des tailles pouvant varier grandement d'une avalanche à l'autre. On remarque cependant que les plus grandes avalanches ont une amplitude maximale $\Delta M^n \simeq 75$, ce qui correspond à une avalanche déplaçant vers le bas une quantité $\sim Z_c/2$ de sable à chaque noeud entre le sommet du tas et une de ses extrémités basales. Vu la taille finie du réseau ($J = 101$ ici), il est tout simplement impossible de développer une avalanche plus intense. On peut vérifier que les masses déplacées par les avalanches sont distribuées en loi de puissance.

9.3 La criticalité auto-régulée

Ce qui est remarquable dans la dynamique globale du système TdS est qu'il accumule de "l'énergie" (i.e., sable) de manière lente et graduelle, mais la libère de façon intense mais intermittente. Le fait que plusieurs systèmes naturels se comportent ainsi a donc généré beaucoup d'intérêt pour le modèle TdS (et ses variantes) comme une représentation idéalisée de ces systèmes. Le tas de sable (réel ou modélisé numériquement, comme ci-dessus) est en fait devenu l'icône du concept de la **criticalité autorégulée** (ma traduction de "self-organized criticality", souvent abbrévié à "SOC"), notion théorique qui fait beaucoup de bruit en physique statistique depuis une vingtaine d'années.

On a vu au chapitre précédent, dans le contexte de la percolation, qu'un système est dit critique quand une petite perturbation locale —l'ajout d'un site occupé quelquepart dans le réseau— se fait sentir globalement —un amas apparait qui traverse maintenant tout le réseau, ce qui rend le matériau électriquement conducteur, ou perméable à l'écoulement d'un fluide, etc. Cependant ce phénomène n'était possible qu'au seuil de percolation, et la manifestation de la criticalité ne pouvait se produire que si ce paramètre de contrôle était soumis à un ajustement —habituellement extérieur— très fin.

Dans le cas du tas de sable, l'équivalent du seuil de percolation est l'angle critique de la pente. Si le tas est sous l'angle critique, l'ajout d'un peu de sable ne fera probablement pas grand chose; si la pente du tas est au dessus de l'angle critique, elle avalanche déjà; mais si on est très près de l'angle critique, l'ajout d'un seul grain de sable quelquepart sur le tas peut: (1) ne rien faire, (2) faire bouger un seul grain, (3) déclencher une avalanche balayant toute la pente, ou (4) n'importe quoi entre (2) et (3). Cependant, et contrairement au seuil de percolation du chapitre précédent, ici cet état critique est un attracteur de la dynamique du système, dans le sens qu'il résulte naturellement de l'interaction entre un très grand nombre de grains de sable, sans ajustement fin d'un paramètre de contrôle par un agent extérieur au système. C'est pourquoi on dit du tas de sable qu'il est dans un état de **criticalité autorégulée**.

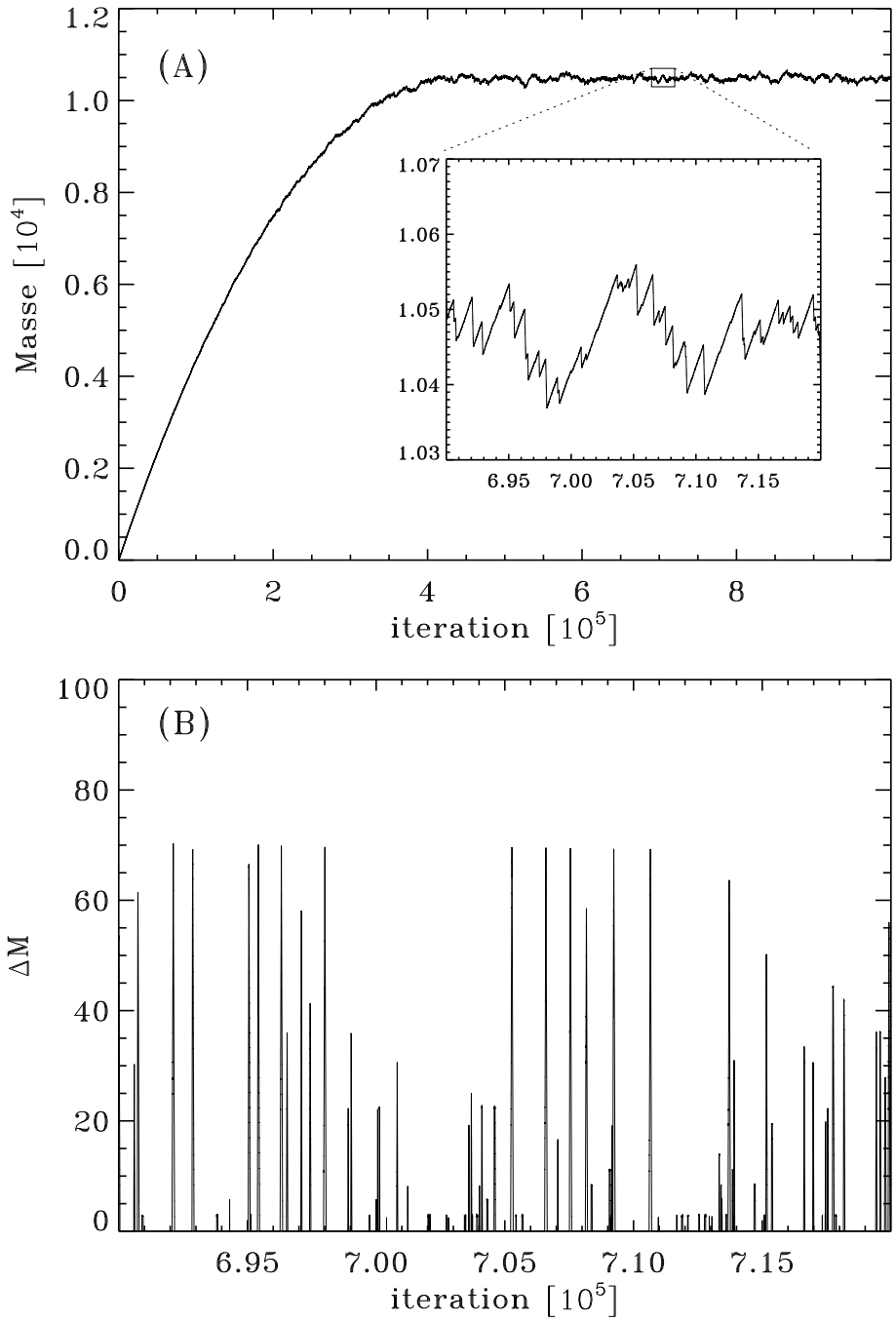

Figure 9.3: Évolution temporelle du modèle TdS en une dimension spatiale, à partir d'une condition initiale $S_j^0 = 0$. Les paramètres de cette simulations sont $J = 101$, $Z_c = 5$, et $E = 0.1$. La partie (A) montre l'évolution de la masse M du tas, l'encadré offrant un zoom sur une petite portion de la séquence, et montrant la forme self-similaire de la séquence temporelle. La partie (B) montre la séquence temporelle de la masse déplacée ΔM, pour le même petit intervalle temporel que l'encadré de la partie (A).

Si on distille tout ça pour en extraire les conditions essentielles pour qu'un système puisse évoluer vers un état critique auto-régulé, on en arrive à la courte liste suivante. Le système doit être:

1. ouvert,
2. soumis à un forçage lent,
3. sujet à une instabilité locale...
4. ...dont l'activation requiert le dépassement d'un seuil d'amplitude finie,
5. et où la restabilisation s'effectue par redistribution locale de la variable dynamique.

Le nombre de système naturels pouvant satisfaire à ces conditions est substantiel: en plus des avalanches et autres formes de glissement de terrain, on peut y inclure les sous-orages géomagnétiques, les tremblements de terre, les éruptions solaires, et les feux de forêts, sujet de la section qui suit.

9.4 Le modèle Feu-de-Forêt

Le modèle "Feux-de-Forêt" (ci-après FdF) appartient la classe dite des **automates cellulaires**. Le modèle assigne à chaque noeud (i, j) d'un réseau cartésien 2D une variable d'état S_{ij}^n pouvant avoir l'une de trois valeurs: 0 pour un site vide, 1 pour un site occupé mais inerte, et 2 pour un site occupé et actif. À partir d'une condition initiale où $S_{ij}^0 = 0$ sur tout le réseau, la variable nodale évolue dans le temps ($S_{ij}^n \rightarrow S_{ij}^{n+1}$) selon une série de règles discrètes dont certaines incorporent une composante stochastique:

1. Règle 1: Si un site est inoccupé, il peut devenir occupé avec une probabilité p_g;
2. Règle 2: Si un site est occupé mais inactif, il peut devenir actif spontanément avec une probabilité p_f
3. Règle 3: Si un site est occupé et inactif, il devient actif si l'un de ses 8 voisins immédiats est actif.
4. Règle 4: Si un site est occupé et actif, il devient vide à l'itération suivante.

L'inspiration initiale du modèle vient de l'écologie: un site occupé et inactif correspond à un arbre; et un site occupé et actif à un arbre en train de brûler; la règle 1, c'est un arbre qui pousse; la Règle 2, c'est un arbre frappé par la foudre; la Règle 3, c'est un arbre en feu enflammant les arbres voisins; et la Règle 4 c'est la destruction d'un arbre par le feu.

Le fait qu'un arbre en feu puisse en enflammer un autre peut conduire à des effets d'**avalanche** sur le réseau, dans le sens que l'activation d'un seul site peut conduire à une reconfiguration majeure du système. Ceci est illustré à la Figure 9.4, qui montre une séquence d'instantanés chacun séparé par 10 itérations[1], d'une simulation ayant $p_g = 10^{-3}$ et $p_f = 10^{-5}$. Cette simulation roulait déjà depuis plusieurs milliers d'itérations, et avait donc déjà atteint un état statistiquement stationnaire. Quelques itérations avant le second instantané la foudre a frappé un peu en haut et à gauche du centre du réseau. L'activité se propage de site en site, sous la forme d'un front de combustion approximativment circulaire initialement, mais développant une forme de plus en plus convoluée à mesure qu'il se propage dans le réseau, en raison du fait que la densité d'arbres dans la configuration pré-activité varie substantiellement à travers le réseau. Cette hétérogénéité est elle-même une conséquence de feux ayant balayé le réseau dans un passé plus ou moins rapproché (voir, e.g., le dernier instantané au bas de la Figure 9.4). Bien que le déclencheur soit ici un processus stochastique (cf. Règle 2), l'activité passée affecte donc fortement l'activité présente.

[1] Des animations en format mpeg de cette simulation, et d'autres pour différentes valeurs des paramètres p_g et p_f sont disponible sur la page web du cours.

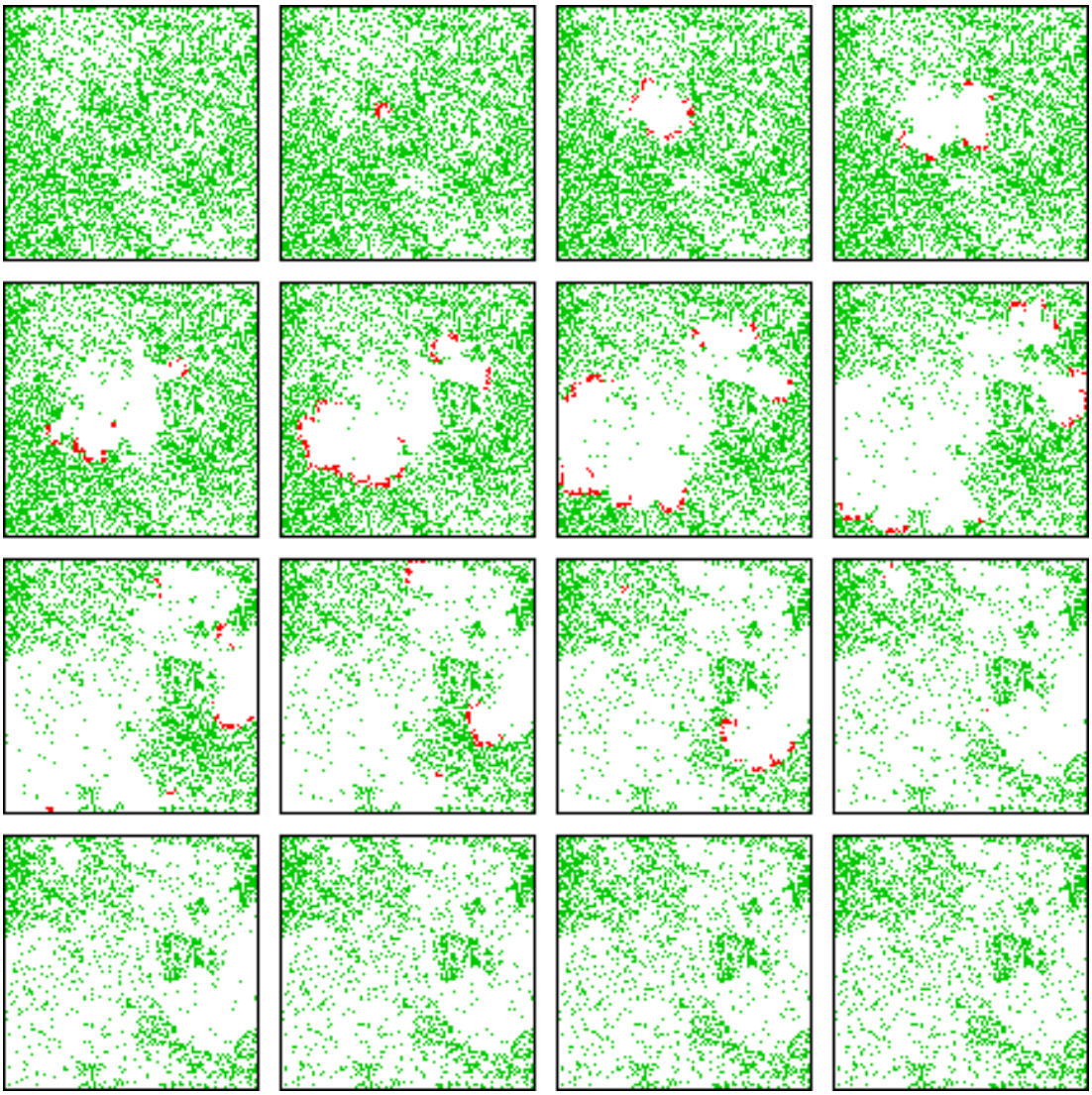

Figure 9.4: Séquence d'instantanés de l'état du réseau pour le modèle FdF avec $p_g = 10^{-3}$ et $p_f = 10^{-5}$, séparés chacun par 10 itérations. Les sites vides sont en blanc, les sites actifs en rouge, et les sites occupés mais inactifs en vert. Au second instantané la foudre vient de frapper un arbre près du centre du réseau; un "front" de combustion balaye par la suite une grande partie du réseau. Simulation sur un réseau de 100×100.

La Figure 9.4 illustre bien la disparité des échelles temporelles implicite au modèle FdF. La plus courte échelle de temps est l'échelle "dynamique" de la propagation de l'activité d'un site occupé à un autre. L'échelle suivante est celle associée à l'occupation des sites vides. La dernière rangée d'instantanés au bas de la Figure 9.4 montre bien la lenteur de ce processus; il faut vraiment y regarder de près pour réaliser que de nouveaux arbres sont apparus ici et là durant les 40 itérations couvertes par ces images. La probabilité d'activation spontanée définit habituellement la plus longue échelle de temps dans le modèle, l'intervalle moyen entre deux activations étant donnée par $(p \times N^2 \times p_f)^{-1}$, où N est la grandeur du réseau dans chaque dimension et p la probabilité moyenne d'occupation. Ici, avec $N = 100$, $p \simeq 0.3$ et $p_f = 10^{-5}$, on s'attend à une activation tous les 33 itérations en moyenne.

Le code C listé à la Figure 9.5 représente une implémentation minimale du modèle FdF, minimale encore une fois dans le sens qu'on aurait pu écrire une version plus efficace du point de vue numérique, mais moins lisible. La structure globale du code est semblable à celle du modèle TdS (Fig. 9.1). Toute l'action est incluse à l'intérieur d'une boucle inconditionnelle contrôlant l'itération temporelle. Remarquez encore une fois qu'on identifie d'abord quels sites doivent s'enflammer, se vider ou se remplir, via le premier bloc de doubles boucles `for`, et que la mise à jour du réseau se fait ensuite de manière synchrone, via le second bloc de doubles boucles `for` à la fin de chaque itération temporelle. Les modifications au réseau sont accumulées dans un tableau (`evol`) de taille identique à celle du réseau. Remarquez aussi qu'on a utilisé la valeur 10 pour identifier un arbre en flammes; ceci permet un truc efficace pour vérifier si un arbre a un voisin enflammé: on additionne les valeurs de `grid` aux 8 sites voisins, et ce n'est que si l'un d'eux est en flamme que cette somme (`q`) ne pourra être plus grande que 10 (huit sites occupés par des arbres ne brûlant pas donnerait une somme `q=8`). Notez finalement que le tableau `grid` inclue un "tampon" de noeuds fantômes sur sa périphérie, qui demeurent toujours vides mais permettent de ne pas avoir à traiter les bords véritables du réseau de manière différente afin d'éviter les débordements de tableaux. C'est pourquoi, à l'intérieur de l'itération temporelle, les boucles sur le réseau commencent à `1` et se terminent à `N`.

Les règles contrôlant l'évolution du modèle sont très simples, et la seule règle de couplage (la 3) est purement locale, dans le sens qu'elle n'implique que les voisins immédiats sur le réseau. De plus, le modèle ne compte que deux "paramètres" ajustables, soit la probabilité (à chaque itération temporelle) de croissance d'un arbre (p_g), et la probabilité (p_f) d'être frappé par la foudre. Malgré tout, ce modèle peut produire une vaste gamme de comportements qui sont fort difficiles à anticiper sur la base de ses règles d'opération. Ceci est illustré aux Figures 9.6 et 9.7, montrant l'évolution du nombre de sites occupés inactifs (N_a, trait noir) et actifs (N_f, traits rouge) pour différentes combinaisons de valeurs de p_g et p_f.

Si $p_f \sim p_g$ (Fig. 9.6, en haut, avec $p_g = p_f = 10^{-4}$), les arbres sont frappés par la foudre à une fréquence comparable à celle à laquelle ils poussent. Le nombre d'arbres sur le réseau demeure à peu près constant, et de petits feux brûlent toujours quelquepart, sans jamais cependant prendre trop d'ampleur car la densité d'arbres est trop faible, ce qui implique que très peu d'arbres ont un de leur 8 sites voisins occupés par un autre arbre. Si on augmente p_g à 10^{-2} (Fig. 9.6, en bas), la densité d'arbre est plus élevée, et les arbres repoussent très rapidement; si rapidement en fait qu'une fois un feu déclenché, il ne s'éteint jamais, étant continuellement alimenté par le "reboisement" rapide s'effectuant derrière le front de combustion. Ceci conduit à des oscillations passablement régulières dans le nombre de site actifs et inactifs, les deux en antiphase, avec la période d'oscillation déterminée par le temps requis pour balayer le réseau, ici de l'ordre de 100 itérations.

Si on laisse maintenant chuter la probabilité d'activation p_f à une très faible valeur ($p_f = 10^{-6}$ pour les deux solutions de la Figure 9.7), alors le réseau a la chance de se remplir beaucoup plus avant qu'un site ne devienne actif. Mais quand celà se produit, presque chaque arbre a au moins un voisin, et on déclenche alors un feu majeur qui brûle presque tout le réseau. Ceci se traduit en un cycle quasi-périodique dit de "charge-décharge", où à intervalles semi-régulier un feu de grande amplitude rase la plus grande partie de la la forêt. (Fig. 9.7, en haut). Avec une probabilité d'occupation ici de $p \simeq 0.5$ la quasi-totalité des arbres ont un autre arbre à au moins un de leurs sites voisins, et donc le feu se propage rapidement. Le fait que le nombre d'arbres

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#define N 100    /* taille du reseau */
#define PG 1.e-3 /* probabilite de croissance */
#define PF 1.e-5 /* probabilite d'activation */
int main(void)
{
/* Declarations ========================================================= */
  float grid[N+2][N+2], evol[N+2][N+2], float tree, burn, q ;
  int i, j, k, niter=100000 ;
/* Executable =========================================================== */
   tree=0. ;       /* Variable compteur: nombre d'arbres sur le reseau */
   burn=0. ;       /* Variable compteur: nombre d'arbres enflammes     */
   for (i=0 ; i<N+2 ; i++) {               /* Initialisation du reseau */
      for (j=0 ; j<N+2 ; j++) {
         grid[i][j]=0 ; evol[i][j]=0. ;  /* aucun arbres sur le reseau */
         evol[i][j]=0 ;
      }
   }
   for ( k=1 ; k<niter ; k++ ) {               /* iteration temporelle */
      burn=0. ;
/* on balaye le reseau pour identifier quels arbres doivent s'enflammer,
   lesquels doivent disparaitre, et lesquels doivent pousser           */
      for (i=1 ; i<N+1 ; i++) {
         for (j=1 ; j<N+1 ; j++) {
            if ( grid[i][j] == 1 ) {            /* le site est occuppe */
/* La foudre frappe */
               if (1.*rand()/RAND_MAX < PF ) { evol[i][j]=9 ; burn+=1. ; }
/* Enflamme par un voisin brulant deja */
               q=grid[i-1][j-1]+grid[i-1][j]+grid[i-1][j+1]+grid[i][j-1]
                +grid[i][j+1]+grid[i+1][j-1]+grid[i+1][j]+grid[i+1][j+1] ;
               if ( q > 10 ) { evol[i][j]=9 ; burn+=+1. ; }
            }
/* Elimination des arbres brulant a l'iteration precedente */
            if ( grid[i][j] == 10 ) { evol[i][j]=-10 ; tree=tree-1. ; }
/* Croissance des arbres aux sites vides */
            if ( grid[i][j] == 0 ) {
               if ( 1.*rand()/RAND_MAX <= PG ) { evol[i][j]=1 ; tree+=1. ; }
            }
         }
      }
/* Maintenant on mets a jour le reseau */
      for (i=1 ; i<N+1 ; i++) {
         for (j=1 ; j<N+1 ; j++) {
/* On enflamme, elimine ou ajoute les arbres identifies prealablement */
            grid[i][j]+=evol[i][j] ; evol[i][j]=0 ;
         }
      }
   printf("iteration, tree, burn: %d  %f  %f\n",k,tree,burn) ;
   }
}
```

Figure 9.5: Code C pour le modèle FdF ayant servi à produire la simulation de la Figure 9.4.

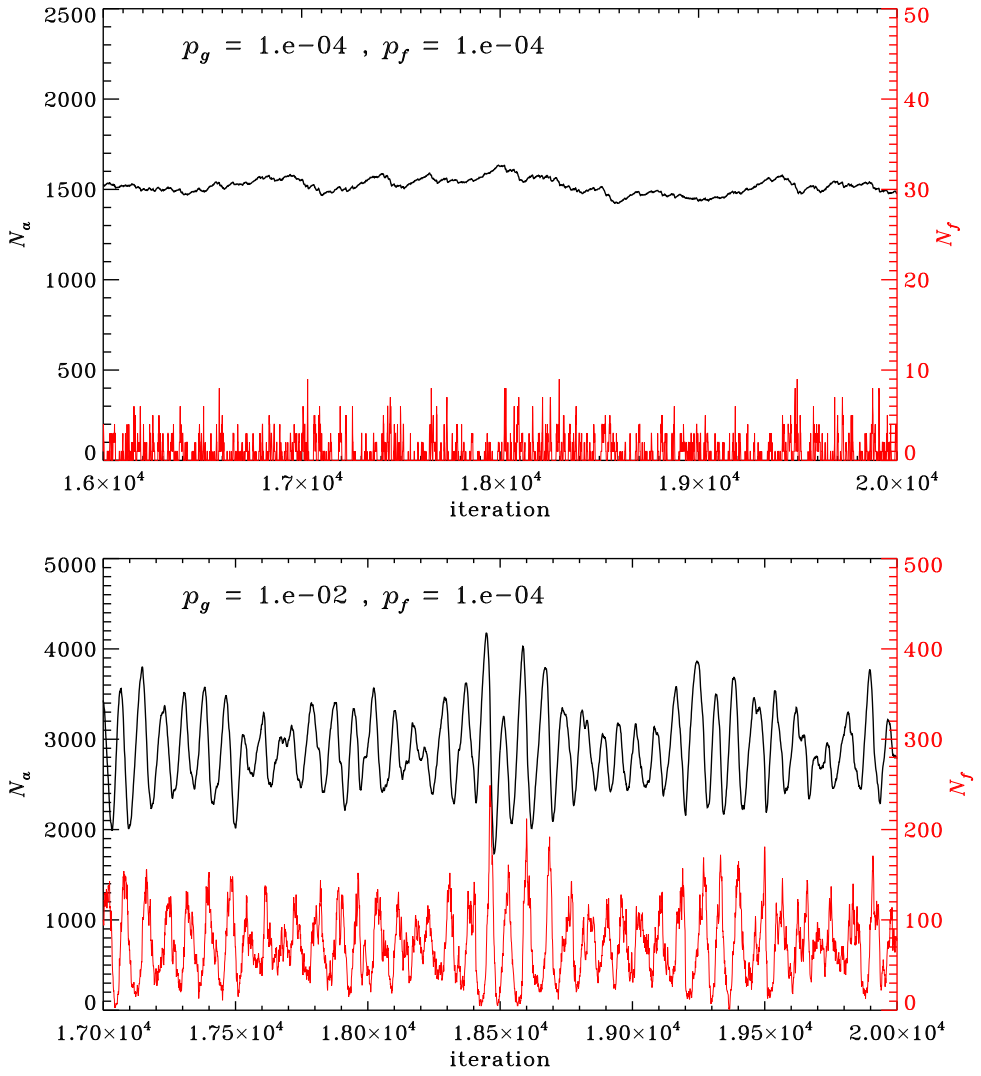

Figure 9.6: Évolution temporelle du nombre de sites occupés inactifs (traits noir) et actifs (traits rouge) dans le modèle FdF avec diverses combinaisons de valeurs de p_g et p_f, dans le régime où la probabilité d'activation est relativement élevée. Toutes les simulations sont effectuées sur un réseau de taille 100×100.

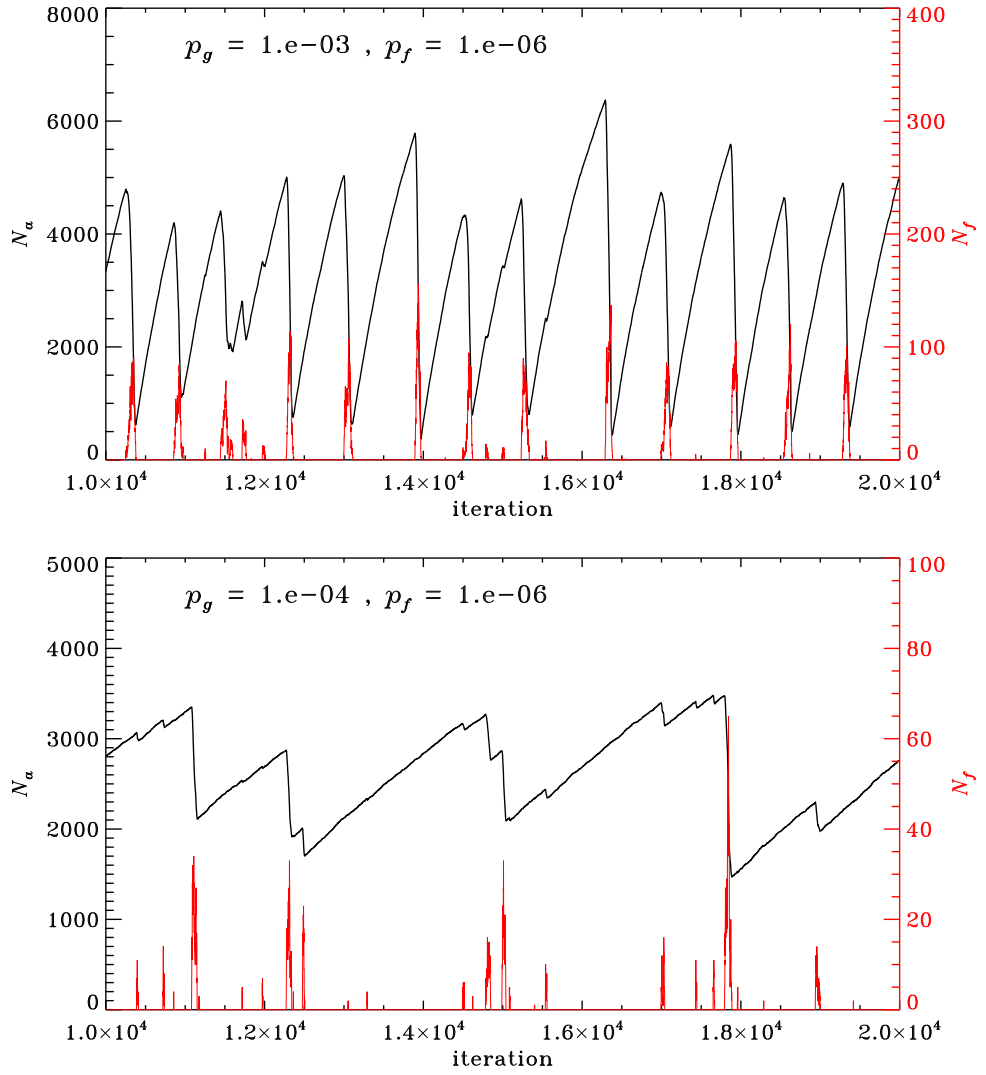

Figure 9.7: Suite de la précédente, maintenant dans le régime où la probabilité d'activation p_f est très faible. Notez bien que les échelles horizontale et verticale des graphiques ne sont pas toujours les mêmes d'un graphique à l'autre.

ne tombe pas à zéro ici est en partie une conséquence des effets de bord, les arbres étant situés sur les bords du réseau ayant moins de voisins se retrouvant plus difficile à enflammer. À bas p_f, ce cycle ne peut être brisé que si la croissance des arbres est elle-même suffisamment lente, de manière à ne pas toujours remplir le réseau entre deux activations par la foudre. Ceci est illustré au bas de la Fig. 9.7, pour une solution toujours avec $p_f = 10^{-6}$ mais où l'on a baissé à $p_g = 10^{-4}$. Bien que des feux de grande amplitude rasent une bonne partie du réseau de temps en temps, ceci se produit maintenant de manière beaucoup plus irrégulière, et les tailles des feux couvrent maintenant un intervalle beaucoup plus large en terme du nombre d'arbres détruits par feu. Notons la forme en dents-de-scie-fractale de la séquence temporelle pour le nombre de sites occupés N, qui n'est pas sans rappeler la séquence temporelle de la masse dans le modèle Tas-de-Sable (cf. Fig. 9.3A, encadré).

Ces différences deviennent frappantes si l'on calcule, à partir des résultats des simulations, les fonctions de densité de probabilité de la taille des feux, en d'autre mots la fonction histogramme $f(N_f)$ mesurant la probabilité d'occurrence d'un feu brûlant entre N_f et $N_f + \mathrm{d}N_f$ arbres. Ces distributions sont portées en graphique à la Figure 9.8, pour les deux simulations de la Figure 9.7. On y voit que dans le régime $p_f \ll 1$, baisser de $p_g = 10^{-3}$ à $p_g = 10^{-4}$ produit une transition d'une distribution de type Gaussienne, caractérisée par une valeur moyenne bien définie, à une distribution en loi de puissance de la forme:

$$f(N) = f_0 N^{-\alpha} \ , \qquad \alpha > 0 \ , \tag{9.11}$$

ici avec $\alpha = 1.07$. Ce qui est remarquable est que dans le régime $p_g \ll 1$, $p_f \ll p_g$, la valeur de α est universelle, i.e., ne dépend essentiellement pas des valeurs exactes des paramètres du modèle. Ça devrait commencer à vous rappeler quelquechose...

Quelle que soit la forme de la distribution $f(N)$, la quantité $f(N)\mathrm{d}N$ mesure la probabilité qu'un feu détruise entre N et $N + \mathrm{d}N$ arbres. La quantité totale (S) d'arbres détruits par l'ensemble de tous les feux est alors donnée par une intégrale du genre:

$$S = \int_{N_{\min}}^{N_{\max}} f(N) N \mathrm{d}N, \tag{9.12}$$

où $N_{\min}$ et $N_{\max}$ représentent les quantités minimale et maximale d'arbres pouvant être détruits, respectivement ici 1 et 10^4, soit le nombre de sites occupables sur le réseau. C'est à toutes fins pratiques la même intégrale que rencontrée au chapitre précédent dans notre étude de la percolation. Substituant l'éq. (9.11) dans l'éq. (9.12), on obtient:

$$S = \frac{f_0}{2-\alpha} \left[N_{\max}^{2-\alpha} - N_{\min}^{2-\alpha} \right] \ . \tag{9.13}$$

Si $N_{\min} \ll N_{\max}$ (et ce sera habituellement le cas pour des systèmes impliquant un très grand nombre de degrés de liberté), on se retrouve encore une fois dans la situation suivante:

1. $\alpha < 2$; l'exposant $2-\alpha$ est alors positif, et donc la quantité d'arbres détruits est dominée par les rares grands feux, via la contribution de $N_{\max}$ à l'évaluation de l'intégrale:

$$S \simeq \frac{f_0 N_{\max}^{2-\alpha}}{2-\alpha} \ , \qquad [\alpha < 2] \ . \tag{9.14}$$

2. $\alpha > 2$; l'exposant $2-\alpha$ est alors négatif, et donc la quantité d'arbres détruits est dominée par les nombreux petits feux, via la contribution de $N_{\min}$ à l'évaluation de l'intégrale:

$$S \simeq \frac{f_0}{(\alpha-2) N_{\min}^{\alpha-2}} \ , \qquad [\alpha > 2] \ . \tag{9.15}$$

Dans le cas du modèle FdF en régime $p_a \ll 1$, $p_f \ll p_a$, on est dans la première de ces situations, et ce sont donc les très rares plus grands feux qui dominent l'évolution de l'écosystème. On

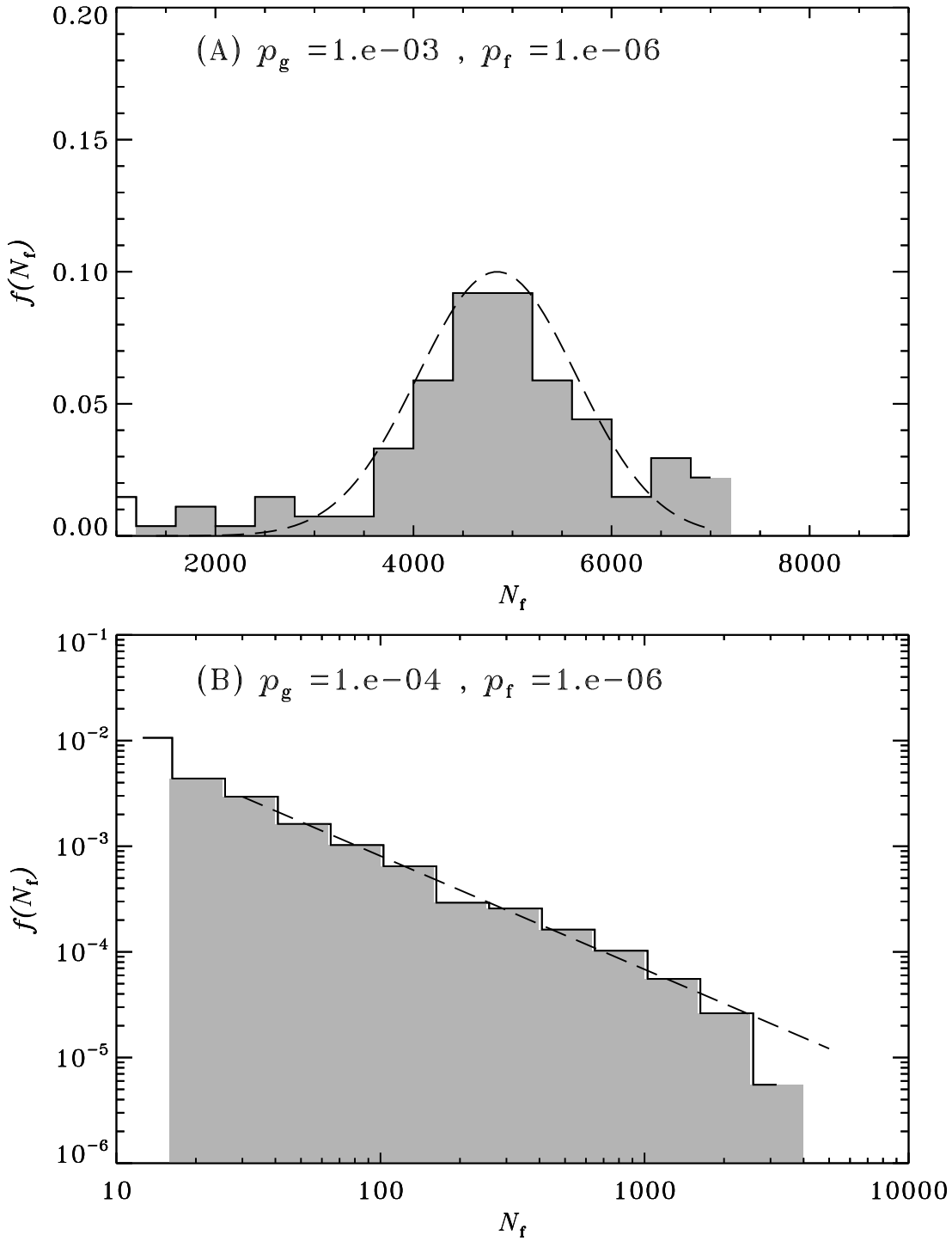

Figure 9.8: Fonctions de densité de probabilité des tailles des feux balayant le réseau, pour les deux simulations de la Figure 9.7. En (A), la distribution est tolérablement bien représentée par une gaussienne, tandis qu'en (B) on observe une loi de puissance, ici avec pente logarithmique -1.07.

observe le même comportement avec, par exemple, les tremblements de terre, où la relaxation des stress tectoniques est dominée par les plus intenses séismes.

Il s'avère que dans la double limite

$$p_f \ll p_g \ , \qquad p_g \ll 1 \ , \tag{9.16}$$

le modèle FdF opère en régime critique auto-régulé; la forçage, c'est la croissance des arbres; l'instabilité, c'est qu'il y ait une densité d'arbre suffisante pour que chaque arbre ait en moyenne un arbre voisin; et la redistribution, c'est la destruction des arbres par le feu.

Chose promise, chose due: le modèle FdF offre un algorithme particulièrement performant pour l'identification des amas dans la percolation sur réseau. Les sites occupés, c'est la fôret; une fois brûlés, les arbres ne repoussent pas; et plutôt que d'allummer de manière aléatoire, on balaye systématiquement les sites du réseau, enflammant successivement chaque site occupé; l'ensemble des sites brulés par chaque allumage correspond à un amas, et le plus grand feu à l'amas le plus grand du réseau.

9.5 Retour sur l'invariance d'échelle

On a vu que les deux systèmes complexe étudiés ci-dessus peuvent produire des "événéments" (avalanches, feux) dont les tailles se distribuent selon une loi de puissance, dont on peut écrire la forme générale comme:

$$f(x) = f_0 x^{-\alpha} \ , \qquad \alpha > 0 \ . \tag{9.17}$$

Ceci s'avère caractéristique des systèmes en état d'autorégulation critique. Considérons ce qui arrive si l'on décide soudainement de changer l'échelle de mesure de la variable x (par exemple, on décide de mesurer la taille d'une avalanche de sable en tonnes plutôt qu'en kilos). Ceci revient à appliquer un changement d'échelle à la variable x, dont on peut représenter le résultat par la définition d'une nouvelle variable y telle que:

$$y = x/a \ , \tag{9.18}$$

où a est une simple constante numérique (e.g., $a = 10^3$ si on décide de mesurer x en tonnes métriques plutôt qu'en kilos). L'éq. (9.17) devient alors:

$$f(x) = f(ay) = f_0 (ay)^{-\alpha} = (f_0 a^{-\alpha}) y^{-\alpha} \tag{9.19}$$

On voit que la distribution $f(y)$ demeure une loi de puissance, *avec la même valeur de l'exposant.* Donc, que la variable x soit mesurée en centimètres ou en années-lumière, sa distribution est toujours une loi de puissance avec pente logarithmique égale à $-\alpha$. On dit donc que la distribution est **invariante** par rapport à un changement d'échelle. Contrastons ceci avec ce qui se passe, sous la même transformation d'échelle, avec la distribution exponentielle:

$$f(x) = f(ay) = \lambda^{-1} \exp(-(ay)/\lambda) = a \left(\frac{\lambda}{a}\right)^{-1} \exp\left(-y)/(\lambda/a)\right) \ ; \tag{9.20}$$

ici, la forme même de la distribution change, dans le sens que la valeur numérique du paramètre d'échelle λ se retrouve effectivement divisée par un facteur a. La distribution exponentielle n'est donc pas invariante par rapport à un changement d'échelle. Il en irait de même avec la distribution gaussienne, pour laquelle et la valeur moyenne et la variance se retrouve à être affectées par un changement d'échelle.

L'invariance d'échelle caractérisant les systèmes en autorégulation critique vient du fait que la dynamique du système n'introduit aucune échelle caractéristique entre la distance internodale et la taille globale du système. Une avalanche, qu'elle n'implique qu'un seul site ou toute la pente, opère toujours via l'interaction d'un site avec ses voisins immédiats; un feu, qu'il soit petit ou grand, se propage toujours via l'allumage d'un arbre par un voisin déjà en feu. Une des conséquences les plus intriguantes de cette invariance d'échelle apparait quand on examine les forme géométriques des avalanches dans le modèle TdS, des fronts de combustion ou des clairières qu'ils forment dans le modèle FdF: ces structures sont toutes des fractales!

9.6 Le modèle embouteillage

Des millions d'autos, "...des millions d'êtres humains qui s'battent pour un pouce d'autoroute, sans trop se demander c'kyé au boutte..."; pas évident à prime abord, mais le trafic automobile partage bon nombre de similarités avec nos deux premier systèmes complexes: un grand nombre de composantes (les autos) n'interagissant qu'avec les quelques composantes voisines (l'auto devant et celle derrière) selon des règles simples (ralentir si l'auto d'en avant ralentit, accélérer si elle accèlère, etc.). Mais cette audacieuse analogie tient-elle la route? Nous examinerons ici cette question à l'aide du modèle embouteillage (ci-après A15), qui offre une représentation très idéalisée du trafic définie par les règles suivantes:

1. Les voitures se déplacent sur une route à une seule voie, à sens unique; donc, pas de dépassement.
2. Les positions et vitesse de la k-ième voiture au temps t_n sont dénotée par x_k^n et v_k^n
3. à chaque pas de temps (n), chaque chauffeur (k) ajuste sa vitesse en fonction de la distance le séparant de l'auto le précédant:
$$\delta = x_{k+1}^n - x_k^n$$
4. Si $\delta < 5$, on ralentit: $v_k^n \to v_k^n - 3$
5. Si $\delta > 5$, on accéllère: $v_k^n \to v_k^n + 1$
6. Vitesse minimale: zéro (on ne recule PAS dans un sens unique, n'en déplaise aux chauffeurs de taxi...);
7. Vitesse maximale: 10 (sinon la SQ s'en mêle vite, avec son enthousiasme habituel pour la chose)
8. Les voitures se déplacent suivant la prescription habituelle reliant le déplacement à la vitesse (ici considérée constante durant une itération temporelle):
$$x_k^{n+1} = x_k^n + v_k^n \times \Delta t \ .$$

Dans tout ce qui suit, on posera $\Delta t = 1$ sans aucune perte de généralité. Le code C de la Figure 9.9 ci-dessous implémente cet "algorithme" de déplacement du trafic, avec une importante addition discutée plus bas. Notez bien, et comprenez, les aspects suivants:

1. La simulation est globalement structurée selon des boucles imbriquées, la boucle extérieure étant l'itération temporelle et une séquence de trois boucles inconditionnelles intérieures sur les `N` voitures;
2. L'initialisation des positions se fait selon des *incréments* de taille aléatoire mais toujours positifs; ici $3 \leq x_{k+1} - x_k \leq 17$, pour un intervalle moyen de 10 unités; cette façon de faire assure que $x_1 < x_2 < x_3 < x_4 < ... < x_{\rm N}$;
3. Le changements de la vitesse des voitures est tout d'abord calculé pour toutes les voitures, et ensuite une seconde boucle modifie le tableau des positions de manière synchrone, en une étape distincte du calcul des changements de vitesse.
4. Une fonction incluant un test assure que la vitesse ne peut chuter sous zéro;
5. Une fonction incluant un test assure que la vitesse ne peut dépasser 10;
6. Un test assure que chaque voiture ne peut pas s'approcher à moins d'une unité de la voiture la précédant;

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define NT    2000                              /* Nombre de pas de temps */
#define N     300                               /* Nombre d'autos */
#define PBOZO 0.1                               /* Probabilite freinage aleatoire */
int main(void) {
/* Declarations ============================================================ */
  int tmin(int, int) ;
  int tmax(int, int) ;
  int x[N], v[N], del, k, iter ;
/* Executable ============================================================== */
  x[0]=1 ; v[0]=0 ;
  for (k=1 ; k<N ; k++) {                       /* repartitions initiale des autos */
    x[k]=x[k-1]+floor(3.+14.*rand()/RAND_MAX) ;
    v[k]=0 ;                                    /* Vitesse initiale nulle */
  }
  for ( iter=0 ; iter<NT ; iter++ ) {           /* boucle temporelle */
    for ( k=0 ; k<N-1 ; k++ ) {                 /* boucle 1 sur voitures: vitesse */
      del = x[k+1]-x[k] ;                       /* calcul distance */
      if ( del < 5 ) { v[k]=tmax(0 ,v[k]-3) ; } /* trop pres: on ralentit... */
      if ( del > 5 ) { v[k]=tmin(10,v[k]+1) ; } /* assez loin: on accelere... */
    }
    if (x[N-1]-x[N-2] <= 10 ) {                 /* Cas special: la voiture de tete */
      v[N-1]=tmin(10,v[N-1]+1) ; }
    for ( k=0 ; k<N ; k++ ) {                   /* boucle 2 sur voitures: freinage */
      if ( 1.*rand()/RAND_MAX <= PBOZO ) {      /* un bozo ralentit parfois pour rien */
        v[k]=tmax(0,v[k]-3) ; }
    }
    for ( k=0 ; k<N-1 ; k++ ) {                 /* boucle 3 sur voitures: on roule */
      x[k]=tmin(x[k]+v[k],x[k+1]-1) ;           /* Deplacement (borne)  */
    }
    x[N-1]=x[N-1]+v[N-1] ;                     /* Cas special: la voiture de tete */
  }                                             /* fin boucle temporelle */
}
int tmin ( int a, int b )                       /* Trouve le plus petit de a,b */
{
  int ff ;
  if ( a < b ) { ff = a ; }
  else         { ff = b ; }
  return  ff ;
}
int tmax ( int a, int b )                       /* Trouve le plus grand de a,b */
{
  int ff ;
  if ( a < b ) { ff = b ; }
  else         { ff = a ; }
  return  ff ;
}
```

Figure 9.9: Code C de base pour le modèle du trafic automobile décrit dans le texte.

7. La voiture de tête, qui n'a pas de voiture la précédant, ajuste sa vitesse de manière particulière, en fonction de la distance de la voiture la suivant;

8. Et voici la clef de la simulation: occasionnellement, sans raison particulière autre que l'arrivée d'un texto, le changement d'un CD, le cellulaire qui sonne, un écureuil traversant la chaussée, ou même vraiment pour faire chier le peuple, un chauffeur aléatoire freine... Ici cet ajustement est effectué après l'ajustement déterministe contrôlé par les distances inter-voitures.

Les Figures 9.10 et 9.11 illustrent différents aspects d'une simulation typique, ici pour un groupe de 300 voitures initialement réparties aléatoirement avec une distance inter-voiture moyenne de 10; il s'agit en fait des valeurs de paramètres utilisées dans le code de la Figure 9.9. La figure 9.10 montre les trajectoires de chacune des 300 voitures, i.e., 300 tracés de x versus t. Une déviation horizontale des tracés, impliquant que x demeure constant quand t augmente, indique une vitesse zéro, soit un embouteillage! La figure 9.11 pousse trois fois plus loin dans le temps, et se borne à indiquer les positions où les voitures sont au repos ou presque (vitesse ≤ 1). Remarquons que même une fois la simulation poussée très loin dans le temps, des embouteillages pouvant impliquer un nombre très variable de voitures se développent de manière en toute apparence erratique. La plupart de ces embouteillages sont causés par le freinage aléatoire d'une voiture, mais ce qui est remarquable est qu'un tel freinage individuel puisse produire (parfois) un embouteillage impliquant une substantielle fraction du groupe de voitures en mouvement. L'encadré sur la Fig. 9.10 illustre la trajectoire d'une voiture particulière s'étant tout juste dépêtrée d'un embouteillage majeur ayant affecté tout le système, et rejoignant, puis quittant, deux embouteillages secondaires en tentant d'accélérer de nouveau. Plusieurs caractéristiques de cette simulation méritent d'être notées:

1. Le trafic se met en branle en deux phases plus ou moins distinctes: une phase initiale "solide" farcie d'embouteillages majeurs, suivie d'une seconde phase "fluide" où toutes les voitures se déplacent à une vitesse moyenne plus ou moins constante et légèrement inférieure à la vitesse maximale permise. Ici la transition semble se produire à $t \simeq 1300$ (cf. Fig. 9.11), mais on verra plus loin que le système atteint un état véritablement statistiquement stationnaire passablement plus tard, soit vers $t \simeq 2000$.

2. Même durant la seconde phase, de nombreux embouteillages se développent et disparaissent, pouvant n'impliquer que quelques voitures, ou une grande fraction du peloton (e.g., l'embouteillage débutant à $(x, t) \simeq (7000, 500)$ sur la Fig. 9.10).

3. Durant la seconde phase, une voiture quelconque tend soit à se déplacer presque à vitesse maximale, soit à être coincée dans un embouteillage (voir trajectoire en orange).

4. L'étendue temporelle de l'embouteillage est substantiellement plus longue que le temps passé par une voiture s'y empêtrant (trajectoire orange); les voitures à la tête de l'embouteillage peuvent accélérer et s'en dépêtrer graduellement, une voiture à la fois, tandis que les voitures atteignant la queue de l'embouteillage s'y empilent. C'est pourquoi l'embouteillage, une fois déclenché, "recule" en x en fonction du temps (voie encadré sur Fig. 9.11).

Deux quantité intéressantes à suivre sont la vitesse moyenne des voitures:

$$\langle v \rangle = \frac{1}{N} \sum_{k=1}^{N} v_k \ , \tag{9.21}$$

et la distance moyenne entre voitures:

$$\langle \delta \rangle = \frac{1}{N-1} \sum_{k=1}^{N-1} (x_{k+1} - x_k) = \frac{x_N - x_1}{N-1} \tag{9.22}$$

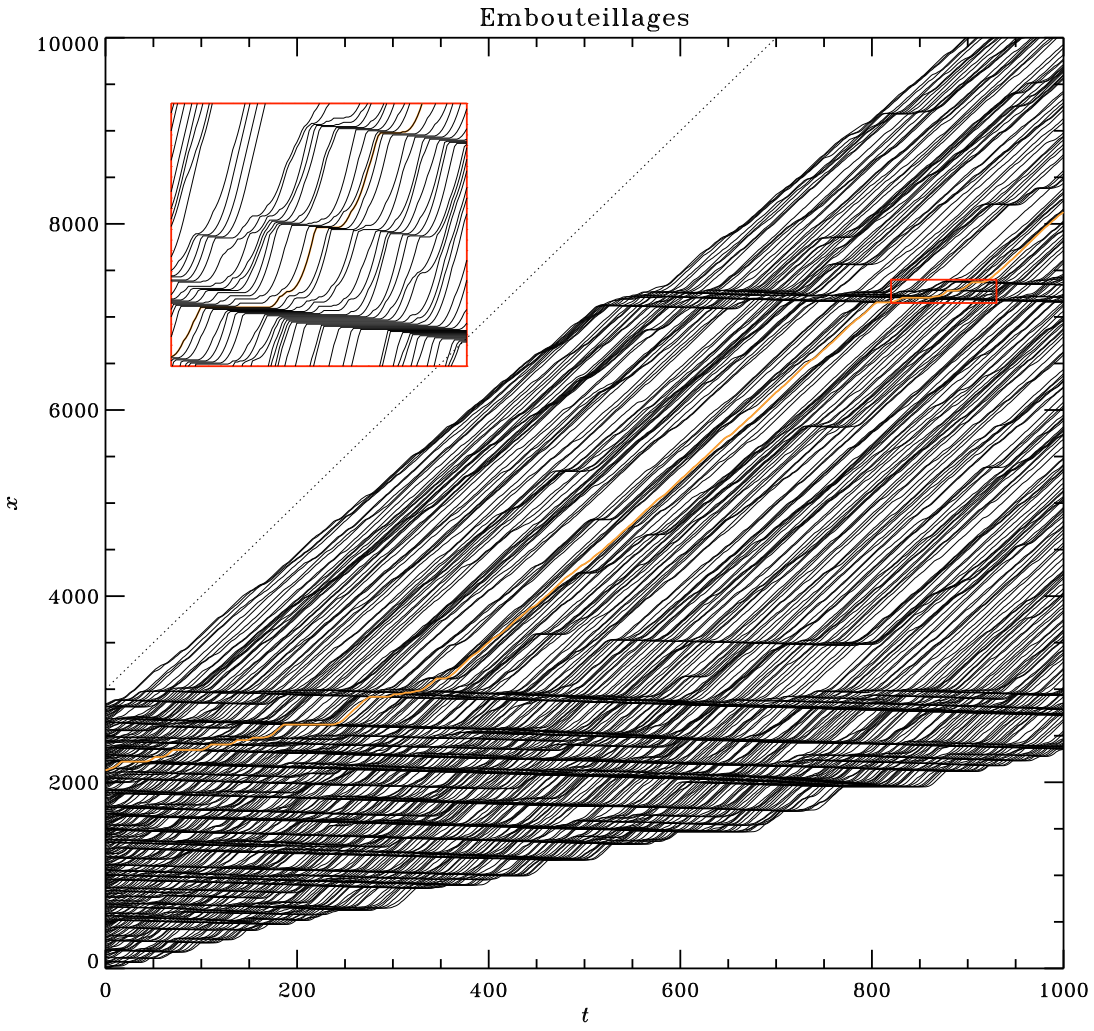

Figure 9.10: Évolution de la position des voitures (axe vertical) en fonction du temps (axe horizontal). On note deux phases distinctes, la première durant laquelle la condition initiale relaxe, par une série d'embouteillages majeurs, vers un état où la vitesse de toutes les voitures est approximativement constante, mais où des embouteillages peuvent néanmoins se produire. Le trait orange indique la trajectoire d'une voiture spécifique, située au trois quart du peloton. Les lignes pointillées indiquent la pente associée à une voiture se déplaçant à la vitesse maximale $v = 10$. L'encadré montre un zoom sur un embouteillage; on y constate que les voitures ne se croisent jamais (comme il se doit puisque les dépassements sont interdits!). Cette simulation, impliquant 300 voitures, a été effectuée avec la condition initiale et les valeurs de paramètres tels que spécifiés dans le code de la Figure 9.9.

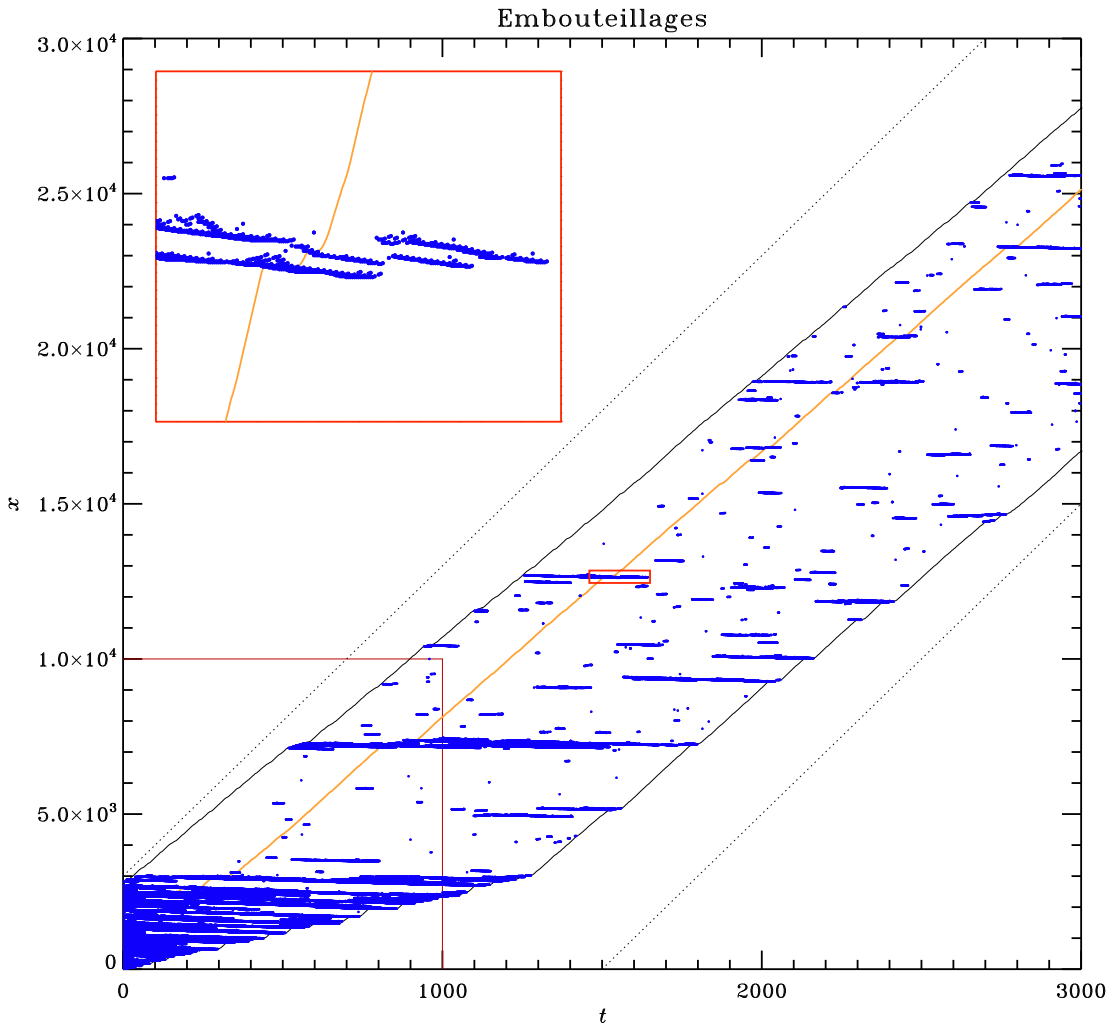

Figure 9.11: La même simulation que sur la Figure 9.10, mais cette fois couvrant un plus grand intervalle temporel et montrant la distribution spatiotemporelle des embouteillages. Chaque point bleu correspond à une voiture au repos ou presque ($v \leq 1$). Le carré inférieur gauche définit l'intervalle couvert sur la Figure 9.10, et les lignes pointillées indiquent encore une fois la pente d'une trajectoire à vitesse maximale $v = 10$. Les trajectoires des première et dernière voitures sont tracée en noir, et la voiture-test en orange.

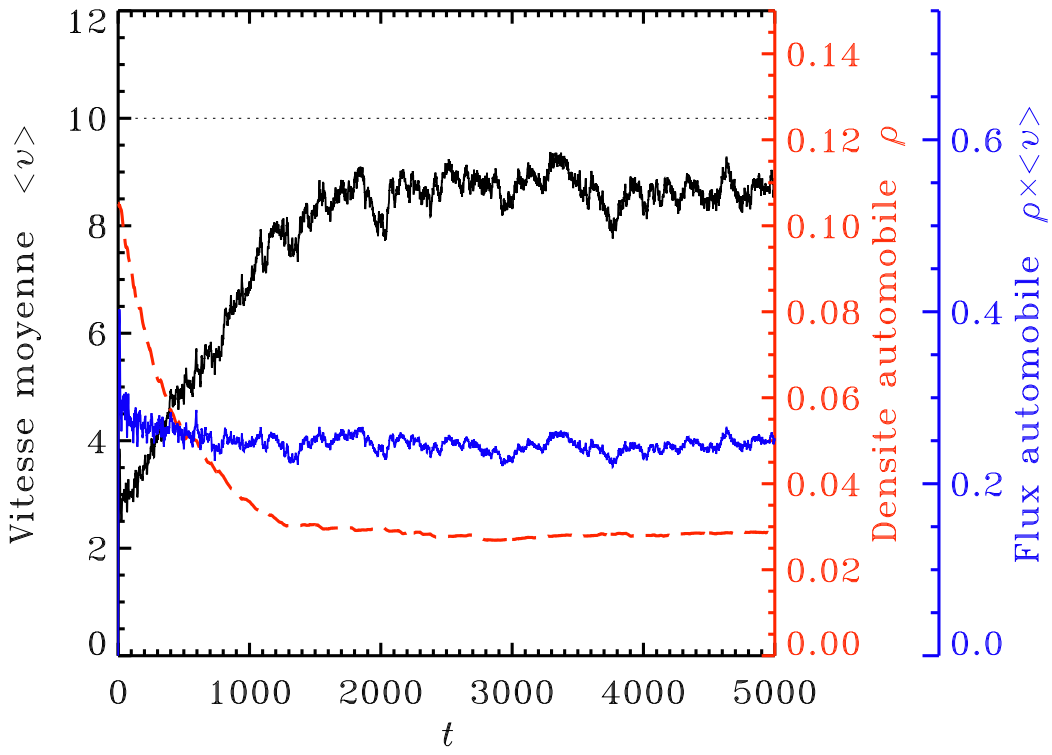

Figure 9.12: Variation temporelle de la vitesse moyenne $\langle v \rangle$, de la densité de voitures ρ, et du flux Φ, pour la simulation des Figures 9.10 et 9.11, maintenant poussée jusqu'à 5000 pas de temps.

(je vous laisse le soin de démontrer la seconde égalité dans cette dernière expression). La densité moyenne de voitures (ρ; nombre de voitures par unité de distance) est simplement l'inverse de cette expression:

$$\rho = \frac{N-1}{x_N - x_1} \tag{9.23}$$

Connaissant ces deux quantités, le *flux* (Φ) de voitures, soit le nombre moyen de voiture traversant une position x^* quelconque par unité de temps, se calcule facilement:

$$\Phi = \rho \times \langle v \rangle \ ; \tag{9.24}$$

La Figure 9.12 montre l'évolution temporelle de $\langle v \rangle$, ρ et Φ, pour la simulation de la Figure 9.10. On y remarque que les valeurs numériques de ces deux quantités varient rapidement jusqu'à $t \sim 1300$, ce qui correspond au changement marqué dans la structure des embouteillages visibles sur la Fig. 9.10, cependant la densité de voitures —et donc aussi le flux— ne se stabilise vraiment que vers $t \simeq 2000$.

La série d'embouteillages monstres caractérisant la phase "solide" en début de simulation suggère que la condition initiale choisie ici n'est peut-être pas la meilleure, du point de vue de la fluidité du trafic. Jusqu'à quel point cette condition initiale influence-t-elle l'évolution de la simulation? Est-elle "oubliée" rapidement ou lentement? En fait, concoctez la condition initiale que vous voulez, ou changez le nombre de voitures (tant que N demeure grand), le système stabilisera *toujours* aux valeurs statistiquement stationnaires de $\langle v \rangle$, ρ et Φ de la Figure 9.12! Cette forme de trafic, avec ses embouteillages intermittent, s'avère être un attracteur de la dynamique du système.

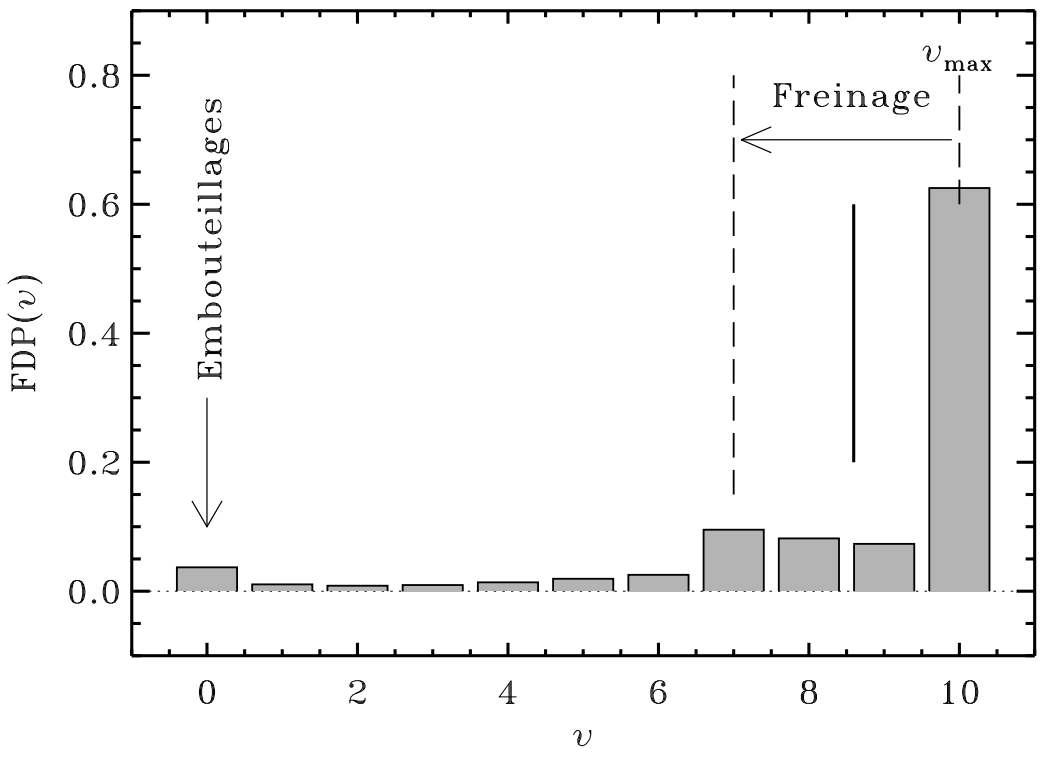

Figure 9.13: Fonction de distribution de probabilité des vitesses, construite à partir des mesures de vitesses de toutes les voitures à chaque pas de temps pour la simulation des Figs. 9.10, 9.11, et 9.12, dans la phase "fluide" $t > 3500$. Le trait vertical noir indique la vitesse moyenne, et le pic secondaire à $v = 7$ est produit par le freinage aléatoire ($v \to v - 3$) de voitures se déplaçant à vitesse maximale (voit texte).

L'évolution vers une vitesse moyenne "robuste" pourrait laisser croire que toutes les voitures se déplacent à une vitesse ne différant pas trop de cette valeur moyenne, e.g. la distribution des vitesses a l'air d'une gaussienne centrée sur $\langle v \rangle$. Ce n'est en fait pas du tout le cas, comme on peut le constater sur examen de la Figure 9.13. La fonction de densité de probabilité est ici calculée par échantillonnage des vitesses de toutes les voitures à tous les pas de temps dans la phase fluide ($t > 3500$). La distribution n'est en rien symmétrique par rapport à la vitesse moyenne (tiret vertical à $v \simeq 8.6$), mais couvre l'intervalle complet $[0, 10]$ avec un maximum à $v = 10$, un pic secondaire à $v = 0$ correspondant aux embouteillages, et un autre à $v = 7$. Ce second pic secondaire est une conséquence directe de la procédure de freinage aléatoire, qui réduit la vitesse de 3 unités, agissant sur le pic de la distribution à $v = 10$. On constate également que les voitures passent un peu plus de 60% du temps à rouler à la vitesse maximale $v = 10$ et environ 4% du temps coincées dans un embouteillage quelconque, ce qui n'est pas si mal finalement (même si le niveau de stress généré serait évidemment hors de proportion par rapport à cette réalité). On voit qu'ici la vitesse moyenne n'est pas une quantité particulièrement utile pour caractériser les vitesses des voitures, même si elle a un sens mathématique incontestable au niveau du flux global de l'ensemble des voitures.

Vous aurez probablement déjà saisi que la formation d'un embouteillage représente une forme d'avalanche de freinages successifs. Si l'analogie tient, on pourrait s'attendre à ce que ces "avalanches" soient caractérisées par une invariance d'échelle. La *taille* d'un embouteillage devrait normalement être définie comme le nombre de voitures bloquée ($v_k^n \leq 1$) sommé sur la durée d'arrêt de chaque voiture impliquée. Autrement dit, le nombre de points bleus dans

chaque "amas" distinct sur la Figure 9.11. Une alternativer plus facile consiste à choisir quelques voitures-test (genre, une dizaine) suffisamment séparées l'une de l'autre, et de comptabiliser les *durées* des phases où chaque voiture est arrêtée ou presque (dans le sens $v_k^n \leq 1$). On sait déjà (voir Figures 9.10 et 9.11) que ces phases sont plus courtes que la durée de l'embouteillage dans son ensemble, mais elles devraient tout de même montrer les même distributions statistiques. Dans les deux cas, on obtient, devinez quoi, une loi de puissance, indiquant une invariance d'échelle au niveau des embouteillages.

Globalement, ce qui compte au niveau du déplacement efficace du traffic, c'est de maximiser le flux de voitures, soit le produit de la densité moyenne de la vitesse moyenne. On pourrait imaginer imposer un grand espacement entre chaque voiture, de manière telle à ce qu'un freineur aléatoire ait assez de temps pour ré-accélérer à la vitesse maximale avant de forcer la voiture le suivant à ralentir. Cependant, une telle configuration serait caractérisée par une densité moyenne de voiture très faible, donc un flux très faible même si toutes les voitures roulent à vitesse maximale. Inversement, forcer toutes les voitures à rouler très près les unes des autres, de manière à avoir une très haute densité et donc un très haut flux, garantit que le moindre freinage aléatoire causera un embouteillage monstre qui forcera l'arrêt complet d'un très grand nombre de voitures, diminuant ainsi drastiquement le flux. Mais voici le truc vraiment subtil: l'état bordellique du trafic simulé ici, avec ses embouteillages spatiotemporellement intermittents et de toutes tailles, représente l'état qui *maximise* le flux de voitures, en présence de freineurs aléatoires. Et cet état est auto-régulé, dans le sens qu'il émerge naturellement des interactions locales entre chaque voiture et ses deux voisines. On pourrait même déclarer que le système est dans un état critique, dans les sens qu'une petite perturbation spatialement localisée —un freinage aléatoire quelquepart— a une probabilitée définitivement non-nulle de causer un changement global dans tout le système —un embouteillage stoppant la quasi-totalité des voitures. On pourrait donc dire qu'on a encore ici affaire à un système en état d'autorégulation critique.

9.7 Complexité $\neq$ Stochasticité $\neq$ chaos

Les trois systèmes considérés ici ont été déclarés complexes *a priori* (en commençant avec le titre du chapitre!). Cependant, les chapitres 4, 7, et 8 nous ont également mis face à divers systèmes se comportant de manière "complexe", du moins dans le sens vernaculaire du terme. On peut donc légitimement s'interroger sur la présence ou absence de différences fondamentales permettant de distinguer et/ou catégoriser les sytèmes stochastiques, chaotiques et complexes (maintenant dans le sens mathématico-physique du terme).

Si on se limite aux système dont l'extrant peut être quantifié mathématiquement, alors l'évolution du système peut habituellement se décrire comme une séquence de nombres ou de symboles plus abstraits. Il s'avère que dans de tels cas, il existe une mesure de la complexité qui peut se définir rigoureusement: c'est la **complexité algorithmique**.

Considérons les six mystérieuses séquences de 20 entiers tabulées dans chaque colonne du Tableau 9.3. Comme chaque entier est composé ici de 5 chiffres (base 10), chaque colonne contient donc à strictement parler 100 caractères, qui occuperait conséquemment 100 octets de mémoire sur votre ordinateur; et si les colonnes avaient 2×10^6 nombres plutôt que 20, ceci demanderait 10 MégaOctet (MO) de mémoire. La quantité de mémoire requise augmente tout simplement linéairement avec la longueur de la séquence, et est donc la même pour chacune des séquences du Tableau 9.3.

Mais examinons de plus près la première colonne (séquence s_1); il s'agit ici d'une alternance régulière de `00000` et `00001`. Donc, plutôt que d'emmagasiner en mémoire 5MO de `00000` et 5MO de `00001`, il pourrait être judicieux d'emmagasiner un *programme* générant `00000` et `00001` en alternance, 20 millions de fois. De même, la seconde séquence pourrait être emmagasinée dans un programme qui débute à `00001`, et ajoute 1 à ceci, 2 millions de fois. La séquence s_3 est plus difficile à déchiffrer; elle croit de manière monotone mais définitivement pas linéaire ou quadratique, mais il semble avoir un pattern ici également. La quatrième séquence semble

Table 9.3: Six séquences numériques plus ou moins mystérieuses...

s_1	s_2	s_3	s_4	s_5	s_6
00000	00001	00001	00001	06464	13153
00001	00002	00002	00004	23945	45865
00000	00003	00003	00005	72118	21895
00001	00004	00005	00008	79626	67886
00000	00005	00008	00009	64242	93469
00001	00006	00013	00014	90966	51941
00000	00007	00021	00003	32539	03457
00001	00008	00034	00020	86927	52970
00000	00009	00055	00025	45000	00769
00001	00010	00089	00007	98010	06684
00000	00011	00144	00003	07723	68677
00001	00012	00233	00024	28221	93043
00000	00013	00377	00025	80218	52692
00001	00014	00610	00004	62839	65391
00000	00015	00987	00017	92472	70119
00001	00016	01597	00046	27565	76219
00000	00017	02584	00021	79069	04746
00001	00018	04181	00025	65535	32823
00000	00019	06765	00049	89442	75641
00001	00020	10946	00023	37395	36533

encore plus irrégulière, dans le sens que globalement elle semble croitre comme s_2 et s_3, mais avec des baisses occasionnelles. Les séquences s_5 et s_6, quant à elles, ne présentent rien qui ressemble à un pattern pouvant s'encoder dans une instruction simple. "À l'oeil", on pourrait donc être tenté de classer les six séquence par ordre de complexité croissante, de la manière suivante (schématiquement):

$$[s_1] < [s_2] < [s_3] < [s_4] < [s_5] \simeq [s_6] \ .$$

Donc, même si toutes les séquences sont définies par le même nombre total d'entiers, on serait tenté de dire que les deux premières sont les plus "simples" et les deux dernières sont les plus "complexes", parce que les premières peuvent clairement être "compressées" en quelques instruction de production, mais (apparamment) pas les autres.

Cette idée est formalisée sous l'appellation de **complexité algorithmique**, qui est définie comme la *longueur* (mesurée en octets, par exemple) du programme le plus court pouvant produire une séquence donnée. Si il n'y a vraiment pas de pattern dans la séquence (comme apparamment ici avec s_5 et s_6), un tel programme ne pourra qu'énumérer les membres de la séquences, dans lequel cas le programme aura essentiellement la même longueur que la séquence même. Mais parfois, comme dans le cas des séquences s_1 er s_2, le programme sera *beaucoup* plus court que la séquence.

En fait, les six séquences du Tableau 9.3 ont été produite par le petit bout de code C reproduit à la Figure 9.14 ci-dessous. Les séquences s_1, s_2, s_3 (séquence de Fibonnacci) et s_5 (séquence chaotique produite par carte logistique) s'avèrent à avoir une complexité algorithmique comparable, dans le sens qu'elles sont toutes produites par une instruction séquentiellement répétée, soit la ligne de code à l'intérieur de la boucle. Ici ces instructions n'impliquent que des opérations arithmétiques simples, comme l'addition ou la multiplication. La séquence s_4 est également produite par une seule ligne de code, mais utilise l'opérateur "%", qui calcule le reste de la division du chiffre le précédant par celui le suivant; opération à prime abord plus compliquée que l'addition ou la multiplication, mais qui du point de vue de

la manière dont elle est calculée par l'ordi, ne l'est pas vraiment. La séquence s_6, quant à elle, résulte d'une manipulation simple d'une séquence de nombres pseudo-aléatoires, de complexité algorithmique plus élevée parce qu'il pas mal d'instructions cachées dans la fonction C `rand()` (incluant l'opérateur `%`!). Les trois systèmes complexes étudiés dans ce chapitre peuvent tous être définis par un code C long d'une page et qui utilise `rand()`), donc on devra assigner à leur extrant (e.g., séquence temporelle de sable déplacé; d'arbres brulés; vitesse d'une voiture) une complexité algorithmique encore plus élevée.

Donc, du point de vue de la complexité algorithmique, on devrait donc reclasser nos six séquences, ainsi que les trois systèmes étudiés dans ce chapitre, de la manière suivante (toujours schématiquement):

$$[s_1] \simeq [s_2] \simeq [s_3] \simeq [s_5] \simeq [s_4] \qquad < \qquad [s_6] \qquad < \qquad [\mathrm{TdS}] \simeq [\mathrm{FdF}] \simeq [\mathrm{A15}] \ .$$

À moins d'avoir l'esprit vraiment tordu, ceci n'est définitivement pas conforme à l'impression "intuitive" produite par un premier examen du tableau 9.3, telle que reflétée dans notre premier classement préliminaire.

9.8 Complexité et émergence

On a vu que les processus stochastiques classiques n'ont aucune mémoire; il est donc impossible, par exemple, de prédire la trajectoire d'un marcheur aléatoire. Donc, vu la nature stochastique du mécanisme déclencheur dans les modèles TdS, FdF, et A15, on s'attendrait à juste titre qu'il soit impossible de prédire leur comportement dans le détail (e.g., étant donné l'état du système à l'itération n, où et quand se déclenchera la prochaine avalanche ou le prochain feu). Cependant il ne faut surtout pas en conclure que ces systèmes se comportent de manière véritablement stochastique, puisque l'état du système à l'itération n a une très forte influence sur l'état du système à l'itération $n+1$. Il est certainement vrai que les systèmes stochastiques et complexes impliquent habituellement un grand nombre de degrés de liberté. Mais, même si la trajectoire d'un seul marcheur aléatoire peut paraitre très "complexe", comme on l'a vu l'évolution de la distribution d'un très grand nombre de marcheurs est, elle, très simple, et exprimable sous la forme d'une équation différentielle pouvant même parfois être solutionnée analytiquement. Ce n'est certainement pas le cas des systèmes complexes en état d'autorégulation critique, qui ne se portent absolument pas à ce genre de réduction statistique. À mesure que l'on augmente le nombre de degrés de liberté dans le système, la possibilité de produire une grande "fluctuation" par rapport à l'état moyen (i.e., une avalanche de taille comparable à l'échelle globale du système) reste la même! Tout le contraire de la convergence statistique des systèmes stochastiques, où dans la limite $N \to \infty$ les fluctuations diminuent comme $N^{-1/2}$.

La relation avec les systèmes chaotiques est encore plus difficile à établir. Ces derniers impliquent habituellement un nombre relativement bas de degrés de liberté, et leur évolution est (habituellement) régie par des règles complètement déterministes qui font que l'état du système au temps $t + \Delta t$ est complètement fixé par l'état du système au temps t. L'antithèse même des processus stochastiques, mais il peut sembler y avoir un lien ici avec les systèmes complexes, puisque ceux-ci évoluent également de manière purement déterministe, sauf pour le déclenchement des avalanches. Pourrait-on imaginer un système complexe comme un système chaotique ayant un grand nombre de degrés de liberté, et sujet à un faible forçage stochastique? La réponse s'avère être non, et, ultimement, est associée à la manière dont les décorrélations s'effectuent dans ces systèmes. Dans les systèmes chaotiques, la décorrélation de deux solutions différant très peu se produit lors de l'approche de points critiques dans l'espace de phase du système (retournez voir la Fig. 4.10). Ces points critiques peuvent être identifiés *ab initio* si la dynamique de chaque degré de liberté est connue. Dans un système complexe en régime SOC, deux noeuds du réseau décorrèlent au passage d'une avalanche, dont les caractéristiques ne peuvent absolument pas être anticipée sur la base des règles d'évolution locales.

L'émergence d'une dynamique globale qualitativement distincte de la dynamique locale semble donc être la condition *sine qua non* requise pour déclarer qu'un système est complexe.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define NN 21
/* Calcul six sequences plus ou moins complexes de nombres    */
int main(void)
{
/* Declarations ------------------------------------------------- */
  float x1[NN], x2[NN], x3[NN], x4[NN], x5[NN], x6[NN], r ;
  int x1p, x2p, x3p, x4p, x5p, x6p ;
  int k ;
  FILE *fd ;
  fd=fopen("tabdata2.dat","w") ;
/* Executable --------------------------------------------------- */

/* Sequence 1: bitflip */
      for (k=0 ; k<NN ; k++) { x1[k]=(k+1) % 2 ; }

/* Sequence 2: entiers positifs */
      x2[0]=1.0 ;                          /* Initialisation */
      for (k=1 ; k<NN ; k++) { x2[k]=x2[k-1]+1 ; }

/* Sequence 3: Fibonnacci */
      x3[0]=1.0 ; x3[1]=1. ;               /* Initialisation */
      for (k=2 ; k<NN ; k++) { x3[k]=x3[k-1]+x3[k-2] ; }

/* Sequence 4: a mod k */
      for (k=0 ; k<NN ; k++) { x4[k]=12345%(3*k+1) ; }

/* Sequence 5: Carte logistique, regime chaotique */
      r=0.99 ; x5[0]=0.983399 ;            /* Initialisation */
      for (k=1 ; k<NN ; k++) { x5[k]=4.*r*x5[k-1]*(1.-x5[k-1]) ; }

/* Sequence 6: Aleatoires uniforme dans [0,1] */
      for (k=0 ; k<NN ; k++) { x6[k]=1.*rand()/RAND_MAX ; }

/* Sortie sur disque */
      for (k=1 ; k<NN ; k++) {             /* boucle sur les pas */
          x1p=floor(x1[k]) ;
          x2p=floor(x2[k]) ;
          x3p=floor(x3[k]) ;
          x4p=x4[k] ;
          x6p=floor(1.e5*x5[k]) ;
          x5p=floor(1.e5*x6[k]) ;
          fprintf (fd,"%5i   %5i   %5i   %5i   %5i   %5i\n",x1p,x2p,x3p,x4p,x5p,x6p) ;
        }
  fclose(fd) ;
}
```

Figure 9.14: Code C pour calculer les six séquences du Tableau 9.14. La sortie sur disque transforme artificiellement ces séquences en entiers de 5 chiffres.

Exercices:

1. Vérifiez que l'éq. (9.7) résulte bien de l'application de la règle de redistribution définie par l'éq. (9.4).

2. Élaborez et codez une version 2D du modèle TdS de la §9.2. Mesurez les tailles et durée des avalanches une fois le système dans son état SOC, et montrez que ces deux quantités se distribuent comme des lois de puissance.

3. Modifiez le code C pour le modèle FdF (Fig. 9.5) de manière à opérer en mode "stop-and-go", c'est-à dire qu'aucun arbre ne peut pousser tant qu'un feu brûle quelquepart sur le réseau. Dans quel(s) régions(s) de l'espace des paramètres celà change-t-il le comportement du modèle ?

4. Modifiez le code C pour le modèle FdF (Fig. 9.5) de manière à ce que seul les quatres voisins haut+bas+gauche+droite puissent enflammer un arbre. Celà change-t-il le comportement global du modèle ?

5. L'équation (9.13) se comporte de manière pathologique ($\rightarrow \infty$) pour une loi de puissance ayant $\alpha = 2$. Comment vous y prendriez vous pour calculer le nombre d'arbres détruits par l'ensemble des feux dans cette situation?

Bibliographie:

Ce chapitre est de mon cru pas mal de A à Z, mais a été évidemment très influencé par des années de lectures et de travaux sur le sujet.

Plusieurs bouquins traitant de complexité à saveur grand public ont été écrits, et sont devenus des "best-sellers". Il pourrait être instructif d'en lire au moins un. Je doit bien être la seule personne que je connaisse (!) à ne pas avoir été particulièrement épaté à la lecture du best-seller *The Quark and the Jaguar*, par Murray Gell-Mann (Prix Nobel de Physique 1969!), mais c'est définitivement un incontournable sur le sujet, dans la catégorie non-technique. Plusieurs ouvrages passablement plus techniques sont également disponibles. Ils font cependant souvent dans la superficialité, en partie en raison de la vaste gamme d'applications du sujet, et aussi en raison du fait qu'il n'existe pas à l'heure actuelle de théorie de la complexité. Dans cette catégorie, ma préférence (en date de la mi-novembre 2009) va à

Érdi, P., *Complexity Explained*, Springer (2008).

Une exception notable à cette tendance à la superficialité est ce qui est rapidement devenu le Manifeste du (maintenant défunt) Santa Fe Institute, soit l'ouvrage:

Kauffman, S.A., *The Origin of Order*, Oxford University Press (1993),

mais l'emphase y est nettement mise sur les systèmes biochimiques. Je suggérerais également l'éclectique bouquin

Hofstadter, D.R., *Gödel, Escher, Bach*, Basic Books (1979),

qui, bien que datant de maintenant 30 ans, demeure une lecture des plus stimulantes intellectuellement. Les chapitres 15 à 19 de l'ouvrage de Flake cité au chapitre 7 représentent également une intéressante introduction au sujet. Tout ce que vous voudriez savoir sur les automates cellulaires se trouve fort probablement dans la plus récente brique de Stephen Wolfram (l'inventeur du logiciel *Mathematica*):

Wolfram, S., *A new kind of science*, Wolfram Media Inc. (2002).

Cependant, préparez vous psychologiquement à devoir endurer l'égo cosmologique de l'auteur, suintant abondamment de chaque page souvent au point de rendre la lecture irritante. Sur la criticalité auto-régulée, commencez par

Bak, P., *Quand la nature s'organise* (trad. *How Nature Works*), Flammarion (1999),

et passez ensuite à la vision plus cartésienne du sujet telle que développée dans:

Jensen, H.J., *Self-organized Criticality*, Cambridge (1998).