PHY-1234

INTRODUCTION
À LA
PHYSIQUE NUMÉRIQUE

Notes de cours
par

Paul Charbonneau
Département de Physique
Université de Montréal

Septembre 2013

MINI-PRÉFACE

Ces notes contiennent toute la matière couverte dans le cours PHY-1234, "Introduction à la Physique Numérique". Ce cours a été complètement remanié en 2008: nouvelles notes de cours, nouveaux projets de laboratoires, nouveau langage de programmation, nouveau prof.

À la lumière des deux premières années, le cours a été retravaillé en 2010 , en plaçant moins d'emphase sur les aspects "MAT" et "IFT" du sujet, et plus sur son coté "PHY". Par rapport aux versions antérieures, on y trouvera moins de matériel traitant de la mathématique sous-jacente aux techniques numériques même, et plus sur les processus et phénomènes physiques pouvant être modélisés sur ordinateur. Cette version 2013 des notes de cours reprend à peu de choses près la version 2012, les quelques changements non-triviaux reflétant encore cette année une emphase plus grande sur le coté "PHY" du cours, et l'élimination de quelques sections dont l'utilité n'a pas été démontrée par l'expérience des années. L'objectif, ramener le texte sous 200 pages, n'a pas été atteint, mais presque! Cependant, plusieurs des dix laboratoires ont été encore une fois remaniés pour pallier aux plus flagrants problèmes et difficultés identifiés l'an passé.

Les divers remaniements du cours, des notes et des laboratoires au fil des années ont été effectués en partie suite aux commentaires de vos prédécesseurs; c'est votre responsabilité de perpétuer la tradition. De mon point de vue je considère toujours que le le cours n'est pas encore bien rodé, et nous demeurons donc tous des cobayes; tous commentaires, suggestions, critiques (en particulier constructives) seront fortement appréciés: passages obscurs, diagrammes portant à confusion, manque de détails ici, détails superflus là, fautes de frappw ou d'ortographe, etc. Merci d'avance!

Table des Matières

1 La physique numérique 7
1.1 La modélisation 7
1.2 Représentations numériques 8
1.3 Les algorithmes 9
1.4 La programmation 10
1.5 Pourquoi le C? 13
1.6 L'analyse numérique 13
1.7 La physique numérique 13

2 Dérivées, interpolation et intégration 17
2.1 Le développement en série de Taylor 17
2.2 Dérivées d'une fonction 20
2.2.1 Discrétisations d'une fonction 20
2.2.2 Les différences finies 21
2.2.3 Exemple: dérivée numérique de $\sin\theta$ 23
2.2.4 Précision, troncation et double précision 24
2.3 Interpolation 24
2.3.1 Interpolation linéaire 27
2.3.2 Interpolation d'ordre plus élevé 29
2.3.3 Interpolation versus extrapolation 32
2.4 Intégration 34
2.4.1 La méthode du trapèze 35
2.4.2 Les règles de Simpson 37
2.4.3 Intégration de Romberg 37
2.5 Coda, en plus qu'une variable 38
2.5.1 Discrétisation 38
2.5.2 Dérivées partielles et différences finies 38
2.5.3 Interpolation multidimensionnelle 40
2.5.4 Intégrales multidimensionnelles 40

3 Équations différentielles ordinaires 45
3.1 Repenser $F = ma$ 45
3.2 La méthode d'Euler 46
3.2.1 Euler explicite 46
3.2.2 Euler explicite avec extrapolation linéaire 48
3.2.3 Euler pour équations nonlinéaires 50
3.3 Le pendule linéaire 50
3.3.1 Solution analytique 51
3.3.2 Reformulation en deux équations d'ordre 1 52
3.3.3 Euler explicite, bis 53
3.3.4 Euler explicite avec extrapolation linéaire, bis 56
3.4 Le pendule nonlinéaire 56

3.5 Au delà d'Euler . . . 58

4 Nonlinéarité et chaos 63
4.1 Encore le pendule... . . . 63
4.2 L'espace de phase . . . 63
4.3 Le pendule amorti . . . 65
4.4 Le pendule forcé . . . 66
4.5 Le pendule forcé et amorti . . . 69
4.6 Bifurcations . . . 71
4.7 Le chaos . . . 72
4.8 Chaos $\neq$ aléatoire . . . 77

5 Monte Carlo 79
5.1 Nombres aléatoires . . . 79
5.2 Fonctions de distributions . . . 83
5.3 Distributions non-uniformes . . . 87
5.3.1 Distributions exponentielles . . . 87
5.3.2 Distributions gaussiennes . . . 88
5.4 Évaluation d'intégrales par Monte Carlo . . . 89
5.5 L'approche à l'équilibre . . . 92
5.5.1 Formulation en équation différentielle . . . 93
5.5.2 Formulation Monte Carlo . . . 95
5.6 Réalisme physique et irréversibilité . . . 95
5.6.1 Comment choisir les alternatives en modélisation? . . . 95
5.6.2 L'irréversibilité . . . 98

6 Racines et optimisation 101
6.1 La diffraction . . . 102
6.2 La bissection . . . 104
6.3 La méthode de Newton . . . 107
6.4 Maximisation (et minimisation) . . . 109
6.5 Méthodes de grimpe . . . 110
6.6 La grimpe stochastique . . . 117

7 Marche aléatoire et diffusion 121
7.1 Les processus stochastiques . . . 121
7.2 Marche aléatoire . . . 121
7.3 Diffusion = marche aléatoire . . . 127
7.4 Marche aléatoire sur réseau . . . 134
7.5 Agrégation . . . 135
7.6 L'invariance d'échelle . . . 138
7.7 Les fractales et leur mesure . . . 139
7.8 La dimension fractale . . . 144

8 Criticalité 153
8.1 La percolation . . . 153
8.2 Percolation en 1D . . . 155
8.3 Percolation en 2D . . . 157
8.4 Invariance d'échelle et universalité . . . 164
8.5 Systèmes critiques en physique . . . 171

9 Complexité **173**
9.1 La complexité 173
9.2 Le modèle Tas-De-Sable 174
9.3 La criticalité auto-régulée 178
9.4 Le modèle Feu-de-Forêt 180
9.5 Retour sur l'invariance d'échelle 188
9.6 Le modèle embouteillage 189
9.7 Complexité $\neq$ Stochasticité $\neq$ chaos 196
9.8 Complexité et émergence 198

10 Calcul évolutif **203**
10.1 Retour sur la modélisation 203
10.2 La puissance de la sélection cumulative 203
10.3 Un algorithme évolutif de base 205
10.3.1 L'algorithme 206
10.3.2 Initialisation 207
10.3.3 Évaluation 207
10.3.4 Classement 207
10.3.5 Sélection 208
10.3.6 Reproduction 208
10.3.7 Remplacement 209
10.3.8 Test de convergence 209
10.4 La diffraction 2D, troisième tour... 209
10.5 Les algorithmes génétiques 211

Chapitre 1

La physique numérique

La représentation mathématique des théories physiques, et la formulation numérique des observations et résultats expérimentaux que ces théories cherchent à expliquer, sont des concepts tellement centraux de la physique moderne qu'il nous est difficile d'imaginer qu'il puisse ou qu'il ait pu en être autrement. Cependant, cette union de fait entre les mathématiques, la physique, l'observation et la manipulation expérimentale est un phénomène relativement récent dans l'histoire de ces sujets. Les historiens des sciences la retracent au dix-septième siècle, et résument sa genèse en disant "de Galilée à Newton". Dans son *Il Saggiatore*, un ouvrage polémique publié en 1623, Galilée écrit:

> "La philosophie est écrite dans ce grand livre, l'univers, perpétuellement ouvert à nos yeux. Mais le livre ne peut être lu sans en avoir d'abord compris la langue et appris à lire l'alphabet. Il est écrit dans la langue des mathématiques, et ses caractères sont les triangles, cercles et autres figures géométriques sans lesquels il est humainement impossible d'en comprendre un seul mot."

Ce n'est pas que la mathématique n'ait pas été utilisée pour décrire (ou même prédire) des phénomènes naturels; déjà il y plus de 2000 ans, les Babyloniens avaient développé des recettes arithmériques permettant une prédiction passablement précise des éclipses du soleil. Et au quatrième siècle, Claudius Ptolémé avait perfectionné un modèle mathématique du mouvement des planètes d'une complexité et d'une précision impressionnantes. Mais tous ces modèles mathématiques étaient vus comme des outils purement descriptifs, sans réalité physique. C'est Galilée qui le premier a insisté sur le fait que des modèles mathématiques basés sur l'observation et l'expérimentation peuvent se proclamer physiques, dans le sens moderne du terme. Si ce genre de considérations scientifico-philosophiques vous intéresse, ne manquez pas de vous inscrire au cours PHY-3315...

1.1 La modélisation

Nous devons tout d'abord faire la distinction entre une **théorie** et un **modèle**. Un exemple devrait suffire. La très célèbre troisième loi de Newton,

$$\mathbf{F} = m\mathbf{a} \ , \tag{1.1}$$

nous informe qu'une force agissant sur un mobile produit une accélération dans la direction de la force appliquée, et de grandeur inversement proportionnelle à la masse m du mobile en question. Ceci exprime une **théorie** (dynamique) du mouvement accéléré.

Considérons maintenant l'application de cette théorie à la chute libre d'un caillou vers le sol. Travaillant en coordonnées cartésiennes avec z pointant vers le haut, la force gravitationelle s'écrit

$$\mathbf{F} = -mg\hat{\mathbf{e}}_z \ , \tag{1.2}$$

où $\hat{\mathbf{e}}_z$ est un **vecteur unitaire** pointant dans la direction de l'axe-z, m est la masse du caillou, et $g = 9.8\,\mathrm{m\ s^{-2}}$. On en déduit que $a \equiv |\mathbf{a}| = -g$, et la position d'un corps relaché d'une hauteur z_0 est donnée par la petite équation qu'on vous a fait mémoriser au secondaire:

$$z(t) = z_0 + v_0 t + \frac{a}{2}t^2 \ , \tag{1.3}$$

où v_0 est la vitesse initiale du corps, nulle ici pour la situation considérée. Les équations (1.2)—(1.3) définissent un **modèle** de la chute d'un corps. C'est un modèle parce qu'à partir de l'éq. (1.1) nous avons fait certains choix arbitraires (e.g., le choix des coordonnées cartésiennes), et surtout plusieurs approximations:

1. La gravité pointe exactement vers le bas;

2. La gravité est supposée constante, à une valeur $g = 9.8\,\mathrm{m\ s^{-2}}$;

3. La seule force agissant sur le corps est la gravité.

La première approximation s'avère excellente, et ce même si vous êtes aux environs du Mont Everest ou collé sur le plus sphérique de vos mononcles; la seconde ne tient que si le corps est relâchée d'une hauteur beaucoup plus petite que le rayon terrestre; et la troisième devient rapidement mauvaise dès que le corps atteint une vitesse de quelques dizaines de mètres par secondes (ce qui ne prendra que quelques secondes!), après quoi la résistance de l'air représente une force de grandeur comparable à la gravité. La prise en considération de cette force demanderait, au minimum, la spécification de la forme du corps, et une connaissance de la densité et coefficient de viscosité de l'air.

Un modèle est donc une représentation approximative de la réalité, basée sur des théories qui se veulent exactes (jusqu'à preuve du contraire). Théories et modèles n'ont pas nécessairement besoin d'être exprimés mathématiquement, mais depuis Galilée, en physique les deux habituellement le sont. Les équations mathématiques décrivant le modèle peuvent être solutionnées analytiquement, comme pour le mouvement rectiligne uniformément accéléré considéré ci-dessus, mais pour un modèle le moindrement complexe ces équations doivent le plus souvent être solutionnées numériquement sur ordinateur.

1.2 Représentations numériques

Sans trop entrer dans le détail de la manière dont les ordinateurs encodent l'information dans leur mémoire ou sur disque, il importe d'examiner un peu la représentation numérique des nombres, car celle-ci peut avoir des impacts importants sur la solution de problèmes physiques sur ordinateurs.

Sur la majorité des ordinateurs, la représentation numérique se limite à deux grandes classes de "nombres": les **entiers** et les **réels**. Les premiers sont presque universellement encodés en mode binaire, tel qu'illustré au Tableau 1.1. Chaque position dans la chaine de "0" et "1" est appelée un **bit**, et 8 bits forment on **octet** (l'anglicisme **byte** étant souvent utilisé).

La représentation numérique des entiers est donc **exacte**, cependant l'architecture du processeur de calcul sur votre ordi limite la longueur de la chaine de bits pouvant définir un entier, ce qui se retrouve à poser une limite supérieure à sa grandeur possible. Sur un système dit à 32 bits, l'entier le plus grand est $2^{31} - 1$ (= 2147483647), sur un système configuré en 64 bits c'est $2^{63} - 1$, etc[1].

Les nombres réels sont également encodés en utilisant une chaine de bits, mais les bits en questions ne sont plus une simple expression en base-2. Un nombre réel ("**float**", pour "floating-point number", en anglais) est d'abord exprimé comme

$$S \times 0.M \times b^{E-e} \ , \tag{1.4}$$

[1]Le premier bit, à l'extrême gauche de la chaine, est utilisé pour encoder le signe de l'entier, $0 \equiv -$, $1 \equiv +$.

Table 1.1: Représentation binaire (32-bits) de quelques entiers bien connus

0	00000000000000000000000000000000
1	00000000000000000000000000000001
2	00000000000000000000000000000010
3	00000000000000000000000000000011
4	00000000000000000000000000000100
5	00000000000000000000000000000101
.	.
68	00000000000000000000000000100100
.	.
143	00000000000000000000000001001111
.	.
.	.

où S est un bit encodant le signe $+/-$, M est une chaine de bits encodant la mantisse comme un entier, b une base, et $E-e$ est une chaine de bits encodant l'exposant. Notez ici que la chaine de bits e est une constante sur une architecture donnée (en C $e \equiv 127$ ou 1023 dépendant de la nature du "float", on y reviendra). Par exemple, la charge électrique élémentaire

$$q = 1.602177 \times 10^{-19}\,\mathrm{C}\ , \tag{1.5}$$

pourrait s'encoder comme:

$$S = 1\ , \qquad M = 1602177\ , \qquad b = 10\ , \qquad E = 109\ , \tag{1.6}$$

en mode 32 bits (avec $e = 127$). Chacun de ces entiers est ensuite converti en représentation binaire, et les chaines de bits concaténées en une seule, longue chaine de bits $S:M:E$ qui est alors la représentation interne du nombre réel.

Notez que la quasi-totalité des architectures utilise en fait $b = 2$, et, comme e, ceci est une constante pré-établie et donc qui n'a pas besoin d'être incluse explicitement dans l'encodage de notre nombre réel.

Comme dans le cas des entiers, la longueur de la chaine de bits définissant la mantisse M est en général restreinte (23 en 32-bits, 52 en 64-bits). Contrairement à la représentation des entiers, *la représentation des réels n'est pas exacte*, car elle est sujette à une **erreur d'arrondissement**. Et comme l'encodage de l'exposant E est également sujet à une limite sur la taille de la chaine de bits le représentant (8 en 32-bits, 11 en 64-bits), il existe une grandeur maximale (et minimale) aux nombres réels pouvant être représentés.

Vous ferez très bientôt connaissance (si ce n'est pas déjà fait) avec les nombres dits complexes; ces derniers sont simplement représentés comme deux réels, le second correspondant à la partie imaginaire du nombre complexe, et le premier à sa partie réelle.

Dernier commentaire important: il n'existe pas sur les ordinateurs d'équivalents directs à la représentation fractionnaire. Par exemple, si deux variables $a = 5$ et $b = 2$ ont été définies comme des entiers, alors l'évaluation du quotient a/b donnera un entier égal à 2, et b/a donnera 0, mais si a et b sont définis comme réels, alors l'évaluation de a/b donnera 2.500000..., et b/a 0.400000, comme l'on s'y attendrait normalement.

1.3 Les algorithmes

Dans le contexte du calcul scientifique, un **algorithme** est une séquence d'étapes conduisant au calcul numérique d'une quantité d'intéret. Considérons par exemple l'algorithme suivant, qui sert à calculer un estimé du nombre irrationnel π:

1. Initialiation: $s = 0$;
2. Générer deux nombre aléatoires r_1, r_2 entre zéro et un;
3. Si $r_1^2 + r_2^2 \leq 1$ ajouter 1 à s;
4. Répéter 10^6 fois les étapes 2 et 3;
5. $\pi = 4 \times s/10^6$.

Étudiez bien la structure générale: on initialise les variables du problème (ici $s = 0$), ensuite des groupes d'opérations mathématiques ou logiques sont effectuées séquentiellement (étapes 2 et 3) de manière répétées (ici un million de fois), conduisant à un résultat final (étape 5), ici un estimé de π. Remarquez également que certaines opérations (étape 3) ne sont pas effectuées à toutes les itérations du processus répétitif.

Un algorithme n'est ni une théorie, ni un modèle dans le sens décrit précédemment. C'est une "recette" complètement déterministe consistant en une séquence d'étapes exécutées l'une à la suite de l'autre. Il existe évidemment une théorie derrière cette approche plutôt bizarre au calcul de π, mais elle n'est pas particulièrement évidente ici (elle le deviendra au chapitre 5).

1.4 La programmation

La programmation consiste à exprimer un algorithme sous une forme déchiffrable par l'ordinateur. C'est habituellement un processus en deux étapes: l'algorithme est d'abord écrit en un **langage de programmation** sujet à des règles syntaxiques spécifiques; le **code source** ainsi produit est ensuite **compilé**, c'est-à-dire traduit en une forme spécifique à l'architecture du processeur de calcul de votre ordinateur. Le petit bout de code source suivant correspond à une implémentation en langage C de notre algorithme de calcul de π:

```
#include <stdio.h>
int main(void)
{
  /* Ce programme calcule un estime de la valeur de pi par Monte Carlo */
  /* Declarations ------------------------------------------------- */
  int i, n ;
  float somme, r1, r2 ;
  float estimePi ;                /* Ceci sera notre estime de pi */
  /* Executable --------------------------------------------------- */
  n=1000000;                      /* On fait 1 million d'iterations */
  somme=0. ;
  for ( i=0; i<n ; i=i+1 )        /* Boucle d'iteration */
    {
      r1 = 1.*rand()/2147483647 ;   /* Un nombre aleatoire entre 0 et 1 */
      r2 = 1.*rand()/2147483647 ;   /* ... et un autre */
      if ( r1*r1+r2*r2 <= 1. )
        {
          somme=somme+1. ;
        }
    }
  estimePi=4.*somme/n ;           /* Calcul de l'estime de pi */
  printf ("valeur de pi = %f\n", estimePi) ;
}
```

Le code source est habituellement "lisible" par un humain familier avec le langage de programmation, et même si vous ne connaissez encore rien au C vous devriez être capable d'y reconnaitre les différent éléments de notre algorithme. Remarquez l'utilisation des parenthères curvilignes "{...}" pour délimiter le bloc d'instructions devant êtres répétées (étapes 2 et 3 dans l'algorithme ci-dessus), et à l'intérieur de ce bloc, un second bloc imbriqué également délimité par des "{...}" indiquant l'instruction sujette à une exécution conditionnelle (étape 3 de l'algorithme). Notez également que la fonction `rand` est une fonction prédéfinie du langage C, équivalente à un dé à 2147483648 faces, dont chaque appel produit un entier aléatoire extrait d'une distribution uniforme dans l'intervalle $[0, 2147483647]$; cette dernière valeur n'est en fait applicable qu'aux architectures montées en 64 bits, comme pour la quasi-totalité des ordinateurs que vous serez susceptible d'utiliser. Il existe toute une batterie de ces fonctions prédéfinies, correspondant aux fonctions mathématiques avec lesquelles vous êtes déjà familiers, comme `sin`, `cos`, `exp`, `log`, etc.

Le résultat de la compilation du code source produit un fichier dit **exécutable**. Ce fichier est typiquement illisible même aux programmeurs d'expérience. Le détail de l'exécutable dépend également du compilateur utilisé pour la traduction du code source en exécutable. Sur la plupart des systèmes LINUX, la compilation du code source se ferait comme suit. On suppose ici que le code source est contenu dans un fichier texte nommé `estimePi.c`. Il suffit alors de taper:

```
gcc estimePi.c
```

Ceci créera dans le répertoire courant un fichier appelé `a.out`, qui, sur la plupart des système Linux, est par la suite exécuté en tapant simplement

```
./a.out
```

Pour le code source ci-dessus roulant sur mon vieux Mac, l'instruction `printf` produit alors la sortie suivante:

```
valeur de pi = 3.142452
```

ce qui approxime bien π (= 3.1415926536...) à 0.04% près. Comme vous pouvez certainement l'imaginer, un algorithme donné peut s'exprimer de différentes manières même si écrit dans le même langage de programmation. C'est comme au primaire, quand on vous demandait une production écrite de 10 lignes sur *Le Brontosaure*; votre texte pouvait bien dire essentiellement la même chose que celui d'un(e) de vos petit(e) camarade, mais son style était assurément différent, et certains styles se lisent mieux que d'autres. Il y a également des styles en programmation, qui souvent visent un compromis entre la lisibilité du code source, la modularité, et la vitesse d'exécution du code une fois compilé. Par exemple, le code suivant produit exactement la même sortie que le précédent, aussi rapidement, mais pour un(e) débutant(e) est un peu plus opaque:

```
#include <stdio.h>
int main(void)
{
  long i, x, y, n=10000000, s=0 ;
  for (i=1; i<=n ; i++) {
    x=rand() ; y=rand() ;
    if (x*x+y*y <= 2147483647.*2147483647.) s += 1 ;
  }
  printf ("valeur de pi = %f\n",4.*s/n) ;
}
```

(toujours pour un ordi monté en 64 bits). Personnellement, c'est plutôt comme ça que j'aurais tendance à coder l'algorithme (quoique j'y ajouterais certainement quelques explications en commentaires). Comme en littérature ou en musique, le goût personnel y est également pour quelque chose, mais certains aspects du style de programmation sont généralement considérés incontournables. Revenant à notre première version du code C, notez par exemple que:

1. Les déclarations de variables sont regroupées en bloc au début du code, et la partie exécutable suit; en C-standard, vous n'avez en fait pas le choix de faire ça comme ça, mais d'autres langages vous laissent entremêler déclarations et instructions. Le "`int`" définit les variables qui suivent comme des entiers, tandis que le "`float`" assigne le statut de nombre réel.
2. On assigne aux variables un nom relié à leur signification dans l'algorithme; le code aurait fonctionné tout aussi bien si la variable `estimePi` avait été appellée `tintinEtMilou` ou `montanaWildhack`[2], mais ce genre d'humour subtil est rarement utile dans un code source.
3. Les blocs d'opérations associés aux instructions de type `for` ou `if` sont visuellement démarqués du reste du code, horizontalement en les décalant vers la droite par un nombre fixe de caractères blancs, et verticalement en leur assignant des délimiteurs { }, chacun sur sa propre ligne. Dans le cas du bloc `if`, comme il ne comprend qu'une seule instruction il serait normal de le remplacer par une seule ligne de code ayant la forme

   ```
   if( r1*r2 < 1.) somme=somme+1. ;
   ```

   comme dans la seconde version du code ci-dessus.
4. Plusieurs commentaires sont ajoutés pour décrire ce que font diverses parties du code. Ces commentaires doivent être encadrés par les délimiteurs `/*` et `*/`, et ne sont pas "vus" par le compilateur;
5. Le point-virgule "`;`" est un caractère indiquant une fin d'instruction; son utilisation permet de combiner une une seule ligne du code source des instructions qui ont un lien "naturel" dans l'algorithme, et ainsi réduire la longueur "verticale" du code; ou encore de répartir sur deux lignes de code une instruction très longue. Ce caractère fin-de-ligne est obligatoire en C, sauf après les parenthèses curvilignes délimitant les blocs d'instructions, puisqu'une fin de bloc est inévitablement une fin d'instruction[3].
6. Plusieurs espaces blancs, qui ne sont également pas "vus" par le compilateur, ont été ajouté afin de faciliter la lecture du code par l'usager.

Nous aurons l'occasion de revenir sur ces questions stylistiques au fil des chapitres qui suivent, lorsque nous discuterons les bouts de code C implémentant les divers algorithmes que nous étudierons. Au niveau du "ligne-par-ligne", le calcul scientifique revient souvent à encoder, utilisant les règles syntaxiques du langage C, des équations mathématique exprimant des lois physiques[4]. Tout les opérateurs arithmétiques de base y trouvent leur pendant, parfois direct, comme les symboles "+" et "−" pour les addition et soustraction, parfois via un symbole différent, comme l'utilisation du "$*$" pour la multiplication, et le "/" pour la division[5].

Un symbole mathématique pratique et bien connu, le "=", mérite une brève discussion ici; en programmation C (ainsi que dans bien d'autres langages de programmation), cet opérateur est unidirectionnel et indique une **assignation**, plutôt qu'une égalité: `b=a` veut dire que la valeur de la variable a (ou plus précisément, le contenu du registre de mémoire assigné à a) est copié à la variable (registre mémoire associé à) b; pour faire le contraire if faut écrire `a=b`. C'est pourquoi en C, l'instruction `s=s+1` est tout à fait légale et appropriée, bien qu'elle fasse hurler de rage la plupart des mathématiciens puristes, qui y voient un non-sens arithmétique absolu (effectivement, $0 = 1!$). Pour leur faire plaisir, le langage C inclut un opérateur "`+=`" tel que `s+=1` veut dire `s=s+1`; et similairement pour les opérateurs "`-=`", "`*=`", etc.

[2] Nom de variable inspiré par un personnage secondaire d'un petit roman qui vaut vraiment la peine d'être lu, et que je vous laisse le plaisir de retracer.

[3] L'oubli du caractère de fin de ligne sera votre plus commune erreur de codage durant vos premières semaines au labo. Avant de disjoncter là-dessus, demandez-moi de vous expliquer comment les instructions sont délimitées en FORTRAN, histoire de vous faire mieux apprécier le `;` du C...

[4] Le tout premier langage de programmation scientifique avait d'ailleurs été baptisé FORTRAN, pour "FORmula TRANslation.

[5] Un manque idiot au langage C standard (à mon très humble avis...) est l'absence d'un opérateur "exposant", qui permettrait de pouvoir écrire autre chose que `x*x` pour mettre `x` au carré.

1.5 Pourquoi le C?

Pourquoi avoir choisi le langage C pour ce cours? À l'origine (c.-à-d. 1972), le C ne se voulait pas un langage pour le calcul scientifique, mais avait plutôt été développé pour les *systèmes d'exploitation*, soit la suite de programmes qui contrôlent le fonctionnemennt de l'ordinateur: l'échange d'information entre le CPU et la mémoire, l'accès disque, le petit programme qui fait parfois apparaitre le message "voulez-vous vraiment effacer votre disque dur?", bref, toute la poutine qui interface vos logiciels préférés avec la "quincaillerie" de l'ordi. Les systèmes d'exploitation UNIX et LINUX sont écrits en C, ainsi que bien des langages dits de "haut niveau", incluant la majorité des logiciels graphiques. Le C contient cependant tous les types d'instructions requis pour le calcul scientifique: opérateurs arithmétiques, instructions de répétition, d'exécution conditionnelle, entrée/sorties numériques, librairie de fonctions mathématiques, etc.

Cette dualité en fait le langage de choix pour bien des applications industrielles ou l'on doit contrôler ou faire communiquer entre eux instruments de mesures, ordinateurs, et/ou machinerie servo-actives. D'ailleurs, plusieurs de vos confrères et consoeurs ayant gradué en physique depuis une vingtaine d'années et qui, par goût ou par nécessité se sont retrouvé(e)s à travailler dans l'industrie de haute technologie sont unanimes: ça aurait donc été pratique d'avoir déjà appris le C au Bac... Donc voilà, on s'y met! De plus, une connaissance de base du C sera extrêmenent utiles à ceux/celles voulant continuer de développer leurs habiletés en programmation via les divers cours de offerts au département d'informatique et recherche opérationnelle, pour lesquels PHY-1234 est maintenant considéré comme un prérequis acceptable.

1.6 L'analyse numérique

L'analyse numérique est une branche des mathématiques dédiée à l'analyse mathématique de diverses propriétés importantes des algorithmes: précision, taux de convergence, stabilité, etc. C'est un sujet immense, que nous ferons qu'effleurer occasionnellement dans ce cours. Dans notre exemple du calcul d'un estimé de π, l'analyse numérique nous permettrait (entre autre) de prédire le nombre d'itérations requises du processus de calcul de π pour atteindre un niveau de précision spécifié *a priori* pour notre estimé. Plusieurs cours sont offerts sur le sujet au département de mathématique, et si vous vous mettez sérieusement à la simulation numérique en physique, vous aurez intérêt à approfondir ce sujet.

1.7 La physique numérique

Qu'est-ce donc que la **physique numérique**? On a l'habitude de compartimenter la physique en blocs associés à des théories comme "l'électromagnétisme", "la relativité", "la mécanique quantique", "la thermodynamique", etc. Cette tendance est par ailleurs très fortement reflétée dans la structure de cours de votre Bac en physique! La physique numérique ne cadre pas très bien dans cette structure, puisqu'elle consiste plutôt en une *approche* à la formulation des problèmes physiques, et à leur solution. Tout ceci peut vous paraitre ésotérique à ce stade, mais d'ici la fin du cours vous devriez avoir saisi de quoi il en ressort. Donc, fonçons!

Exercices:

1. Déterminez les valeurs de S, M et E encodant le nombre $\pi = 3.1415926536...$, en mode 32-bits avec $b = 2$ et $e = 127$.

2. Calculez le plus grand et le plus petit réel positif pouvant être encodé selon l'éq. (1.4), en mode 32-bits et 64 bits.

3. Codez le premier code présenté dans ce chapitre pour calculer π, et déterminez le nombre d'itérations requises pour obtenir un estimé de π précis à 10^{-6}. NB: si vous travaillez sur un système monté en 32 bits, remplacez le chiffre `2147483647` par `32767`.

4. Travaillant avec la seconde version "raccourcie" du code C calculant la valeur de π, remplacez `4.*s/n` par `s/n*4.` dans la ligne d'instruction commencant par "`prinf`". Contre toute attente purement mathématique, ceci changera drastiquement la valeur numérique de votre estimé de π. Comment expliquez-vous ceci?

Bibliographie:

Les notes que vous avez en main contiennent toute la matière qui sera couverte en PHY-1234. À moins d'être déjà absolument et totalement convaincu(e) que vous ne toucherez plus au calcul scientifique de votre vie (ce qui risque de fort limiter vos options de carrière, soit dit en passant...), je vous recommande l'achat du bouquin:

Press, W.H., Teukolsky, S.A., Vetterling, W.T., & Flannery, B.P., *Numerical Recipes: the Art of Scientific Computing*, Cambridge University Press (1992–2007)

Ce sera fort probablement le premier et dernier dans lequel vous aurez à investir sur ce sujet. Plusieurs versions du bouquin existent, et ne varient qu'au niveau du langage de programmation utilisé. Dans la troisième édition (2007) les auteurs ont opté pour le langage C++, un langage basé pour le C mais incluant des fonctionalités additionnelles au niveau de la programmation dite "orientée objet". Les exemples de codes inclus dans ce bouquin sont écrits en C++, mais la majorité des codes peuvent être directement compilés en C et utilisés tel quel. La seconde édition, qui utilisait le langage C plutôt que C++, est malheureusement épuisée, mais les habitué(s)s d'*Amazon* peuvent essayer de dénicher une copie. Cet ouvrage demeure également inégalé (à mon avis) dans l'équilibre qu'il atteint entre l'analyse numérique, le design d'algorithmes simples et robustes, et les bonnes habitudes de programmation.

Plusieurs ouvrages, traitant spécifiquement de modélisation numérique en physique ont été écrits et les meilleurs souvent réédités au fil des années. Parmi ces grands classiques, l'ouvrage suivant demeure recommandable:

Gould, H., & Tobochnik, J., *An Introduction to Computer Simulation Methods*, 2^{e}éd., Addison-Wesley (1996).

Même si la seconde édition est relativement récente, à bien des points de vue elle conserve la "saveur" de l'édition originale. À un niveau physique et mathématique plus avancé que celui de ce cours, j'aime bien:

Pang, T., *An Introduction to Computational Physics*, 2^{e}éd., Cambridge University Press (2006).

Bien que l'approche utilisée dans ce cours sera d'apprendre les bases de la programmation "sur le tas", un ouvrage de référence sur le langage C vous sera certainement très utile. Pour ceux/celles n'ayant vraiment aucune expérience en programmation, l'ouvrage suivant est une bonne introduction au langage C, en français de surcroit:

Delannoy, C., *Le livre du C premier langage*, Éditions Eyrolles (2007).

Plus complets, et plus appropriés pour ceux/celles ayant déjà fait un peu de programmation:

Delannoy, C., *Programmer en langage C*, 5^{e} éd., Éditions Eyrolles (2009).
Aitken, P., & Jones, B.L., *Le langage C*, 2^{e} éd., Pearson (2012).

La référence classique, loin d'être idéale pour les débutants cependant, demeure:

Kernighan, B.W., & Ritchie, D.M., *The C programming language*, 2^{e} éd., Prentice Hall (1988).

Les écrits de Galilée valent encore bien la peine d'êtres lus; je recommenderais le suivant:

Drake, S. (éd.), *Discoveries and Opinions of Galileo*, Doubleday (1957).

Chapitre 2

Dérivées, interpolation et intégration

Dans ce chapitre et deux autres à venir, nous allons voir comment effectuer numériquement sur ordinateur la série de grandes manoeuvres mathématiques qu'on vous a appris en calcul intégral et différentiel: calculer des dérivées et intégrales (ce chapitre), solutionner des équations différentielles simples (chapitre suivant), et trouver les zéros et extrema d'une fonction (chapitre 6).

Nous devrons en fait d'abord discuter le contexte mathématique qui est à la base du calcul numérique des dérivées et intégrales, soit le développement en série de Taylor (§2.1). Nous examinerons ensuite le processus de discrétisation d'une fonction, pour en arriver au calcul de formules dites de différences finies pour le calcul numérique des dérivées (§2.2). Nous examinerons ensuite l'interpolation (§2.3), ce qui paraitra être un détour mais nous fournira une base solide pour passer au calcul numérique des intégrales (§2.4). Nous concluerons en généralisant toutes ces grandes manoeuvres aux fonctions de plus d'une variable (§2.5).

2.1 Le développement en série de Taylor

On vous a appris au CEGEP (ou peut-être même avant) que la **dérivée** d'une fonction $f(x)$ évaluée à $x = x_0$ correponds à la pente d'une droite tangente à $f(x)$ au point x_0. Mathématiquement, ceci revient à écrire:

$$\left.\frac{\mathrm{d}f(x)}{\mathrm{d}x}\right|_{x_0} = \lim_{\Delta x\to 0} \frac{f(x_0+\Delta x) - f(x_0)}{\Delta x} \tag{2.1}$$

avec $\Delta x \equiv x - x_0$. Comme l'illustre la Figure 2.1, l'utilisation de l'équation (2.1) avec un Δx de taille finie produira une erreur sur le calcul de la pente, qui cependant s'amenuise à mesure que Δx devient de plus en plus petit.

On ne vous l'a pas appris comme ça à l'époque, mais l'équation (2.1) résulte en fait d'une **troncation** d'un développement plus général, dit en **série de Taylor**, d'une fonction continue $f(x)$ dans les environs de x_0:

$$\begin{aligned} f(x_0+\Delta x) &= f(x_0) + (\Delta x)\left.\frac{\mathrm{d}f}{\mathrm{d}x}\right|_{x_0} + \frac{(\Delta x)^2}{2!}\left.\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}\right|_{x_0} + \frac{(\Delta x)^3}{3!}\left.\frac{\mathrm{d}^3 f}{\mathrm{d}x^3}\right|_{x_0} + \ldots \\ &= f(x_0) + \sum_{n=1}^{\infty}\frac{(\Delta x)^n}{n!}\frac{\mathrm{d}^n f}{\mathrm{d}x^n}\ , \end{aligned} \tag{2.2}$$

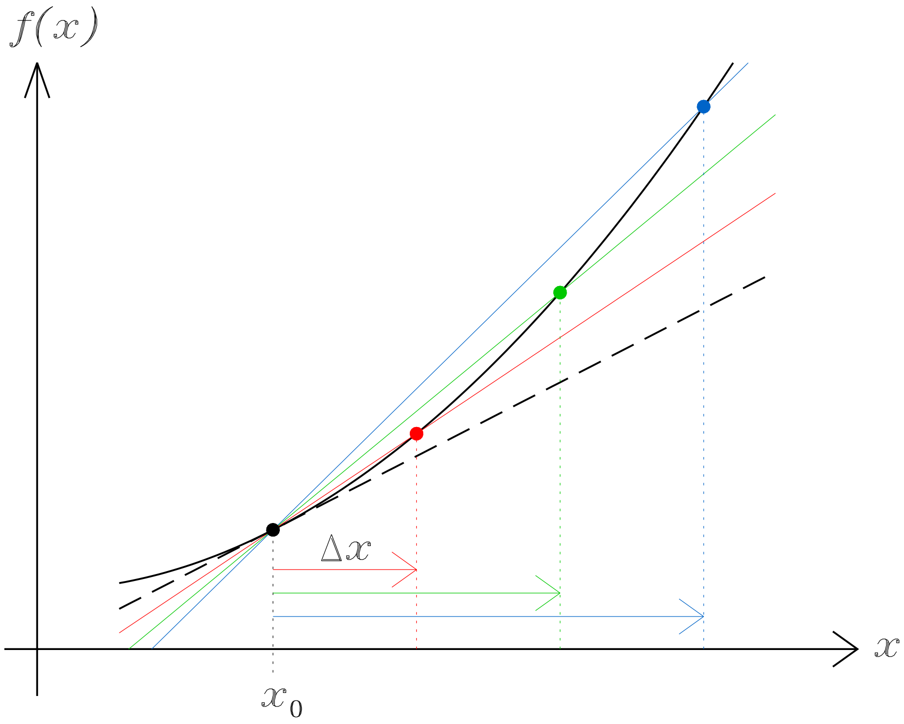


Figure 2.1: La dérivée comme un procesuss de calcul de pente dans la limite $\Delta x \to 0$. Plus Δx est petit, plus la droite passant par x et $x + \Delta x$ a une pente se rapprochant de la tangente à $f(x)$ au point x_0 (droite en tirets).

où $n!$ est la fonction factorielle, définie comme:

$$n! \equiv n \times (n-1) \times (n-2) \times \ldots\ 2 \times 1\ . \tag{2.3}$$

La Figure 2.2 illustre l'idée générale d'un tel développement. La fonction développée est ici un polynôme quartique de la forme:

$$f(x) = 1 + x^4\ . \tag{2.4}$$

Les quatre autres courbes résultent de l'utilisation du développement (2.2), en ne conservant que 1, 2, 3 ou 4 termes, tel qu'indiqué sur la Figure. Ne conserver que le premier terme revient à dire que $f(x)$ assume une valeur constante:

$$f(x_0 + \Delta x) = 1 + x_0^4\ . \tag{2.5}$$

Conserver les deux premier termes conduit à:

$$f(x_0 + \Delta x) = 1 + x_0^4 + 4(\Delta x)x_0^3\ , \tag{2.6}$$

tandis que conserver trois ou quatre termes donne:

$$f(x_0 + \Delta x) = 1 + x_0^4 + 4(\Delta x)x_0^3 + 6(\Delta x)^2 x_0^2\ , \tag{2.7}$$

$$f(x_0 + \Delta x) = 1 + x_0^4 + 4(\Delta x)x_0^3 + 6(\Delta x)^2 x_0^2 + 4(\Delta x)^3 x_0\ . \tag{2.8}$$

Examinez bien la Figure 2.2, pour vous convaincre que le développement devient de plus en plus précis à mesure que le nombre de termes retenus dans le développement augmente,

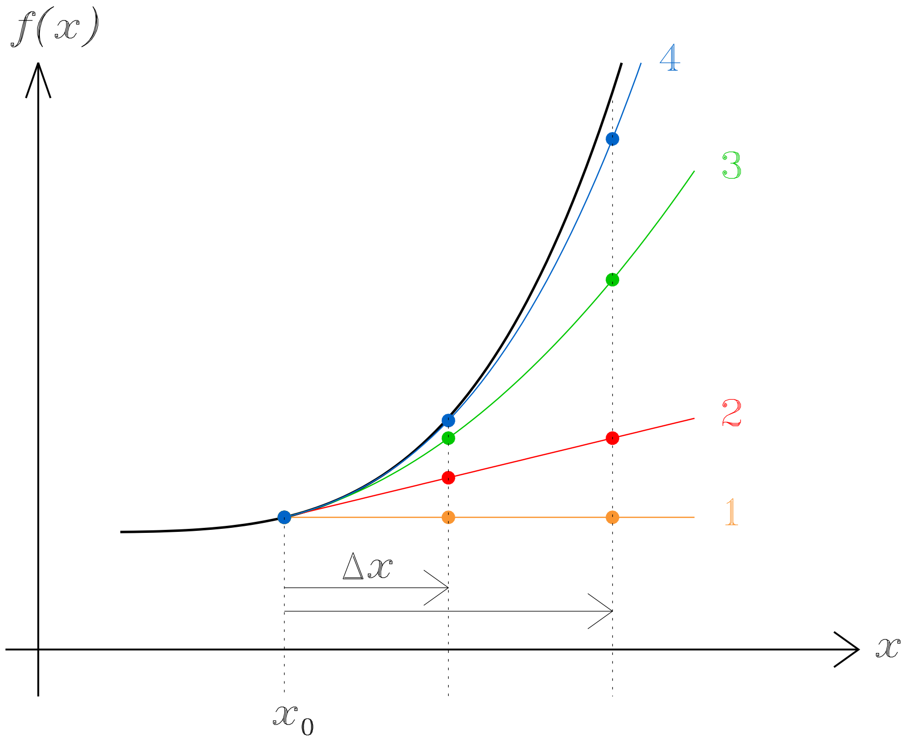


Figure 2.2: Développement en série de Taylor d'un polynôme quartique (trait noir), en ne conservant qu'un, deux, trois ou quatre termes dans le développement, tel qu'indiqué. Le développement devient de plus en plus précis à mesure que le pas Δx diminue, et/ou le nombre de termes conservés dans le développement augmente.

ou que l'intervalle Δx diminue. Remarquez également qu'une fois les dérivées évaluées, le développement se retrouve à prendre la forme d'un polynôme en puissances de Δx.

La précision du développement peut évidemment être améliorée en subdivisant l'intervalle Δx en sous-pas, et en reévaluant les dérivées au membre de droite de (2.2) après chaque sous-pas. Un exemple est présenté à la Figure 2.3, toujours pour notre polynôme quartique. Comparant cette Figure à la Fig. 2.2, notez comment la précision de l'estimé de $f(x_0 + 2\Delta x)$ d'ordre n est comparable à celui de l'estimé d'ordre $n+1$ quand le pas est deux fois plus grand.

Si $\Delta x \to 0$, chaque terme successif au membre de droite de l'éq. (2.2) est plus petit que le précédent par un facteur Δx, donc si $\Delta x \ll 1$ on peut s'attendre à ce que le développement soit dominé par les quelques premiers termes, et je vous laisse vérifier que l'équation (2.1) est obtenue en ne conservant que les deux premiers.

Croyez-le ou non, mais dans vos cours de physique au secondaire vous avez déjà fait connaissance avec un développement en série de Taylor, tronqué au second terme. Dans le mouvement rectiligne uniformément accéléré, la position $x(t)$ d'un mobile au temps t est donné par:

$$x(t) = x_0 + v_0 t + \frac{1}{2} a t^2 \ , \tag{2.9}$$

où x_0 et v_0 sont les position et vitesse au temps $t = 0$. Puisque

$$v = \frac{\mathrm{d}x}{\mathrm{d}t} \ , \qquad a = \frac{\mathrm{d}v}{\mathrm{d}t} = \frac{\mathrm{d}^2 x}{\mathrm{d}t^2} \ , \tag{2.10}$$

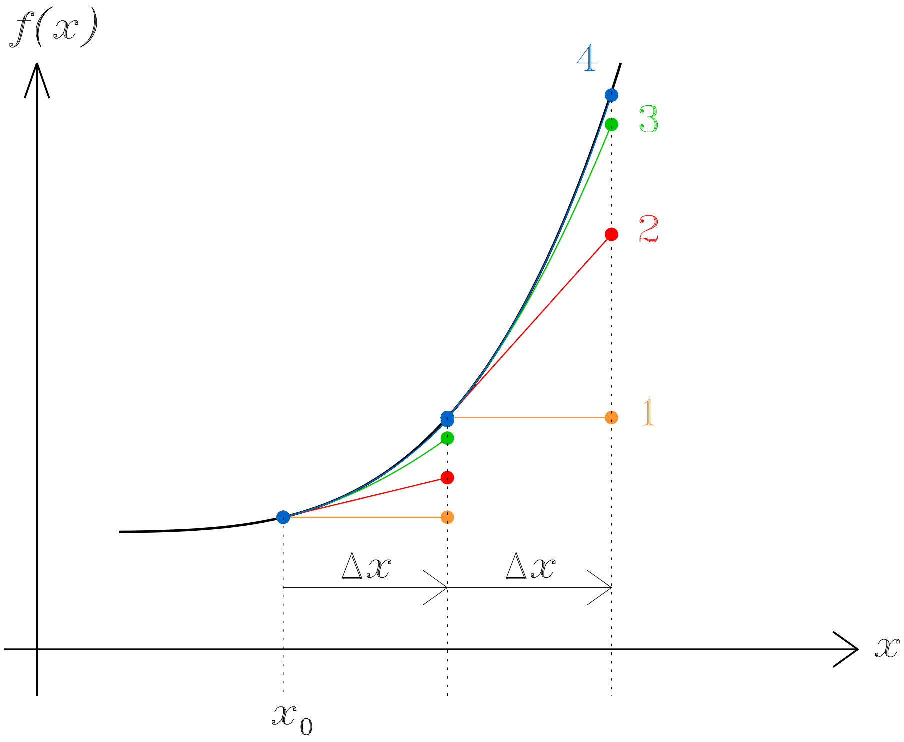


Figure 2.3: Semblable à la Figure 2.2, sauf que le pas Δx est subdivisé en deux, et les dérivées apparaissant dans le développement reévaluées à la position $x_0 + \Delta x$ pour obtenir un estimé final à $x_0 + 2\Delta x$.

l'éq. (2.9) peut s'écrire comme:

$$x(t) = x_0 + t \frac{\mathrm{d}x}{\mathrm{d}t}\bigg|_{t=0} + \frac{t^2}{2}\frac{\mathrm{d}^2x}{\mathrm{d}t^2}\bigg|_{t=0} . \tag{2.11}$$

Associez maintenant $x \equiv t$, $\Delta x \equiv t - t_0$, et $f \equiv x$, et comparez à l'eq. (2.2)!

Le développement en série de Taylor apparaitra de manière récurrente dans les chapitres qui suivent. Pour l'instant, nous nous limiterons à la situation suivante: on cherche à évaluer de manière discrète les dérivées d'une fonction continue d'une variable $f(x)$. On suppose que cette fonction est calculable pour tout x, mais a une forme telle qu'il nous est impossible d'en calculer la dérivée analytiquement.

2.2 Dérivées d'une fonction

2.2.1 Discrétisations d'une fonction

Nous devons d'abord introduire l'idée de la **discrétisation** d'une fonction continue $f(x)$, soit sa représentation en terme d'un nombre fini de valeurs numériques, processus incontournable lorsqu'on cherche à obtenir une solution numérique de quoi que ce soit. Le point de départ est la définition d'une **maille** (ici en 1D) sur laquelle $f(x)$ est évaluée à un nombre fini de **noeuds** (ou **points de maille**) x_j:

$$x \to \{x_1, x_2, ..., x_N\}, \qquad x_{j+1} > x_j , \tag{2.12}$$

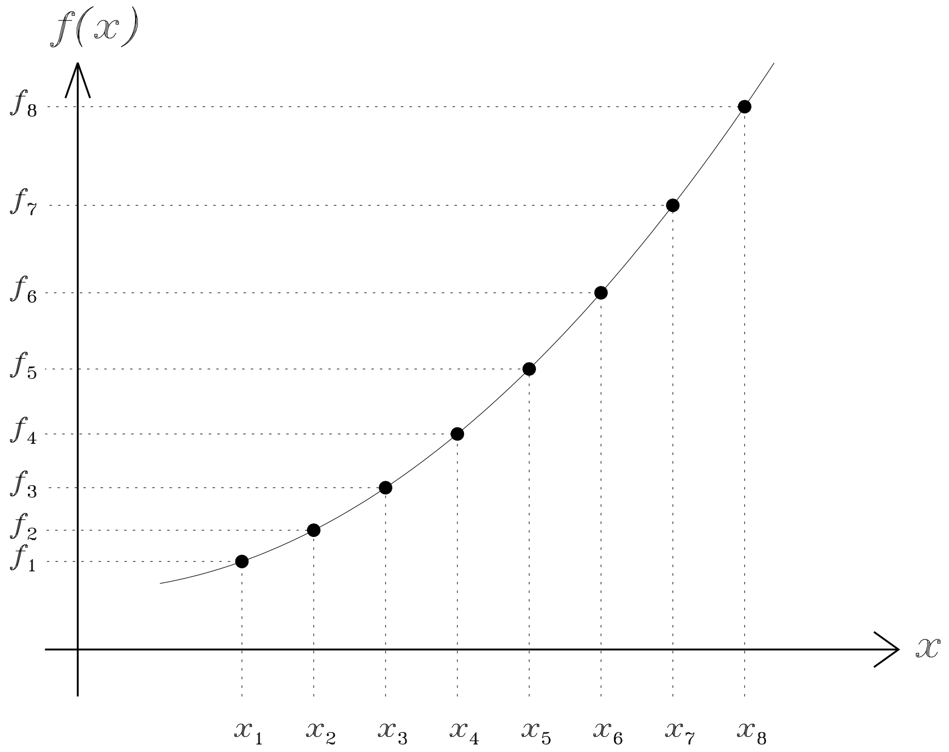


Figure 2.4: Discrétisation d'une fonction parabolique sur un maillage régulier, avec $N = 8$. Plus N est grand, plus la discrétisation donne une représentation précise de la fonction.

Un maillage est dit **régulier** si les intervalles entre chaque x_j successifs sont égaux:

$$x_{j+1} - x_j \equiv h \qquad \forall j. \tag{2.13}$$

L'évaluation de $f(x)$ aux x_j produit une ensemble de N valeurs de f, qui collectivement définissent la discrétisation de $f(x)$:

$$f_j \equiv f(x_j) \ , \qquad j = 1, 2, ..., N \ . \tag{2.14}$$

Cette procédure de discrétisation est illustrée à la Figure 2.4, pour une fonction parabolique discrétisée sur un maillage régulier avec $N = 8$.

2.2.2 Les différences finies

Le développement en série de Taylor est à la base du calcul des dérivées d'une fonction discrétisée. Par exemple, en tronquant tous les termes au membre de droite de l'éq. (2.2) sauf les deux premiers, on peut immédiatement obtenir une approximation pour la dérivée première de la fonction discrétisée $f(x_k)$:

$$\left.\frac{\mathrm{d}f}{\mathrm{d}x}\right|_{x_j} = \frac{f_{j+1} - f_j}{h} + O(h) \ . \tag{2.15}$$

Le terme en $O(h)$ au coté droit de cette **formule de différences finies** indique que **l'erreur de discrétisation** y étant associée décroit proportionnellement à h ($\equiv \Delta x$) dans la limite $h \to 0$, conséquence directe de notre omission ici des termes en $(\Delta x)^2, (\Delta x)^3, ...$ dans l'éq. (2.2). Notez que cette approche de troncation "brute" fonctionne pour la dérivée première, mais ne nous permet pas de produire de formule équivalente pour la dérivée seconde. Voyons comment faire mieux.

Notre point de départ consiste à établir les deux développements suivants pour $f(x_j \pm h)$ au voisinage de x_j:

$$f_{j+1} \equiv f(x_{j+1}) \equiv f(x_j + h) = f(x_j) + h\frac{\mathrm{d}f}{\mathrm{d}x}\bigg|_{x_j} + \frac{h^2}{2!}\frac{\mathrm{d}^2f}{\mathrm{d}x^2}\bigg|_{x_j} + \frac{h^3}{3!}\frac{\mathrm{d}^3f}{\mathrm{d}x^3}\bigg|_{x_j} + ... \tag{2.16}$$

$$f_{j-1} \equiv f(x_{j-1}) \equiv f(x_j - h) = f(x_j) - h\frac{\mathrm{d}f}{\mathrm{d}x}\bigg|_{x_j} + \frac{h^2}{2!}\frac{\mathrm{d}^2f}{\mathrm{d}x^2}\bigg|_{x_j} - \frac{h^3}{3!}\frac{\mathrm{d}^3f}{\mathrm{d}x^3}\bigg|_{x_j} + ... \tag{2.17}$$

De nouvelles formules de différences finies sont obtenues par manipulations algébriques simples des éqs. (2.16)—(2.17); par exemple, en soustrayant l'équation (2.17) de (2.16) et en éliminant les termes proportionnels aux puissance de h de trois et plus, on peut isoler $\partial f/\partial x$ et on obtient:

$$\frac{\mathrm{d}f}{\mathrm{d}x}\bigg|_{x_j} = \frac{f_{j+1} - f_{j-1}}{2h} + O(h^2) . \tag{2.18}$$

Et voilà pour la dérivée première. Maintenant, On peut aussi additionner les équation (2.17) et (2.16) et isoler $\mathrm{d}^2f/\mathrm{d}x^2$, ce qui produit

$$\frac{\mathrm{d}^2f}{\mathrm{d}x^2}\bigg|_{x_j} = \frac{f_{j+1} - 2f_j + f_{j-1}}{h^2} + O(h^2) , \tag{2.19}$$

Par utilisation récursive des éqs. (2.16) et (2.17) on peut ensuite obtenir des formules de différences finies pour les dérivées d'ordre plus élevées:

$$\frac{\mathrm{d}^3f}{\mathrm{d}x^3}\bigg|_{x_j} = \frac{f_{j+2} - 2f_{j+1} + 2f_{j-1} - f_{j-2}}{2h^3} + O(h^2) , \tag{2.20}$$

$$\frac{\mathrm{d}^4f}{\mathrm{d}x^4}\bigg|_{x_j} = \frac{f_{j+2} - 4f_{j+1} + 6fj - 4f_{j-1} + f_{j-2}}{h^4} + O(h^2) . \tag{2.21}$$

Qu'avons nous gagné ici? Le terme en $O(h^2)$ aux cotés droits des éqs. (2.18)—(2.19) indique que l'erreur de discrétisation associée à ces formules de différences finies pour la dérivée première décroit maintenant proportionnellement à h^2 dans la limite $h \to 0$. Ces différences finies sont donc dites **d'ordre 2**. En principe la formule (2.18) calcule la même chose que l'éq. (2.15), mais pour une maille donnée, est en général plus précise.

Par d'autres manipulations des éqs. (2.16) et (2.17) et utilisation récurrente des formules d'ordre $O(h)$ and $O(h^2)$ ci-dessus, on peut obtenir des formules alternatives, caractérisées par des erreurs de discrétisation d'ordre plus élevé:

$$\frac{\partial f}{\partial x}\bigg|_{x_j} = \frac{-f_{j+2} + 8f_{j+1} - 8f_{j-1} + f_{j-2}}{12h} + O(h^4) , \tag{2.22}$$

$$\frac{\mathrm{d}^2f}{\mathrm{d}x^2}\bigg|_{x_j} = \frac{-f_{j+2} + 16f_{j+1} - 30f_j + 16f_{j-1} - f_{j-2}}{12h^2} + O(h^4) . \tag{2.23}$$

Les formules de différences finies (2.18)—(2.23) sont dites **centrées** sur le noeud x_j où la dérivée est évaluée. Elles ne peuvent dont pas être utilisées sur le premier ($j = 1$) et dernier ($j = N$) noeud de la maille. Cependant d'autres manipulation algébriques des éqs. (2.16) et (2.17) peuvent produire des formules de différences finies **avant** ou **arrière**. Les intéressé(e)s peuvent consulter les références citées en fin de chapitre.

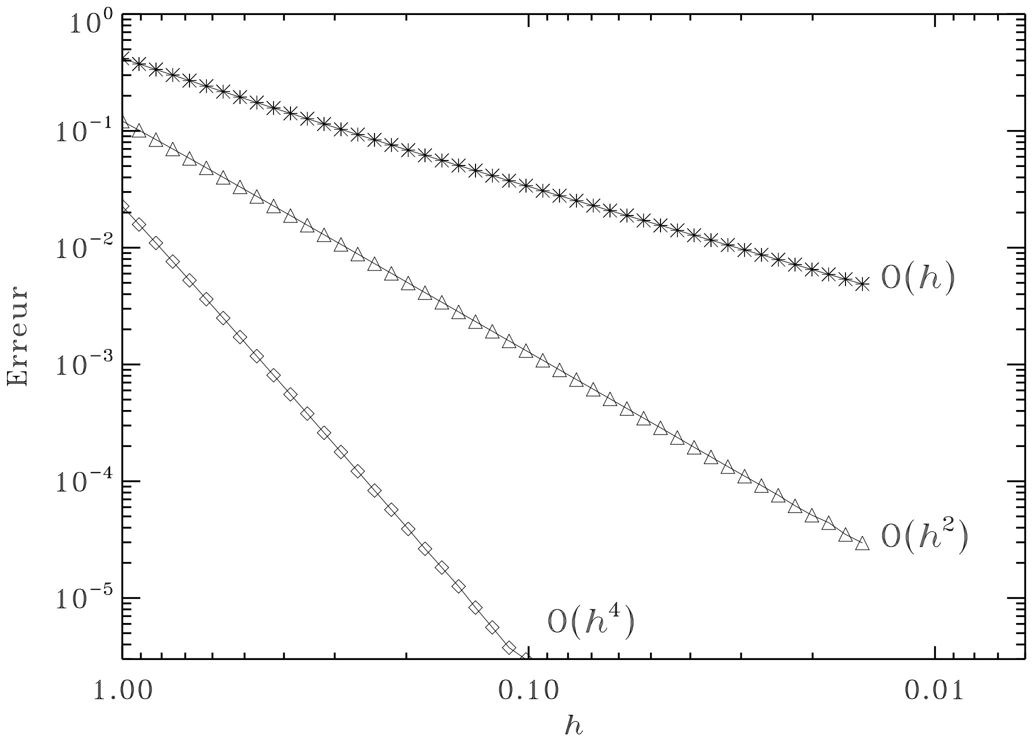


Figure 2.5: Variations de l'erreur de discrétisation en fonction de la taille du pas, pour l'évaluation numérique de la dérivée de $\sin\theta$ à $\theta = \pi/6$, utilisant les formules de différences finies données par les éqs. (2.18) ($\triangle$), (2.15) ($*$) et (2.22) ($\diamond$). Notez bien que les deux axes sont logarithmiques, et que l'abcisse est telle que h *décroit* de la gauche vers la droite.

2.2.3 Exemple: dérivée numérique de $\sin\theta$

Pour apprécier la signification de l'ordre de discrétisation, considérons l'exemple suivant. Suite à une année sabbatique particulièrement mouvementée, vous avez complètement oublié que la dérivée d'un sinus est un cosinus:

$$\frac{\mathrm{d}\sin\theta}{\mathrm{d}\theta} = \ ????? \ , \tag{2.24}$$

ce qui est excessivement problématique dans votre cours de mécanique (entre autre), mais votre calculatrice vous permet heureusement de calculer $\sin\theta$. La Figure 2.5 montre l'erreur associée à l'utilisation des éqs. (2.15), (2.19), et (2.22), pour évaluer la dérivée de $\sin\theta$ ($= \cos\theta$!!) à $\theta_k = \pi/6$. On suppose ici que

$$f_k = \sin\theta_k \ , \qquad f_{k-1} = \sin(\theta_k - h) \ , \qquad f_{k+1} = \sin(\theta_k + h) \ . \tag{2.25}$$

La mesure de l'erreur utilisée ici est

$$\varepsilon = \left| \frac{\mathrm{d}\sin\theta}{\mathrm{d}\theta} - \cos\theta \right| \ , \tag{2.26}$$

où la dérivée est évaluée à l'aide de nos trois différences finies pour la dérivée première, soit les équations (2.15), (2.18), et (2.22). On remarque que dans tous les cas l'erreur diminue à mesure que h diminue, mais que l'ordre de la formule utilisée détermine le *taux* auquel diminue l'erreur à mesure que h diminue:

$$O(h): \ \varepsilon \propto h^1 \ , \qquad O(h^2): \ \varepsilon \propto h^2 \ , \qquad O(h^4): \ \varepsilon \propto h^4 \ , \tag{2.27}$$

puisque les axes sur la Fig. 2.5 sont logarithmiques.

Les données requises pour construire chacune des courbes sur la Figure 2.5 se calculent en C à l'aide du code listé sur la Figure 2.6 (ici pour la formule $O(h^2)$ donnée par l'éq. (2.18)). Ce code utilise les fonctions mathématiques prédéfinies `sin(x)`, `cos(x)`, et `fabs(x)`, cette dernière calculant la valeur absolue de x. La **librairie** qui les contient doit être incluse, et invoquée au moment de la compilation du code; La première ligne `#include <math.h>` est donc requise ici, et la compilation doit maintenant spécifier un lien vers ces librairies:

```
gcc cosinus.c -lm
```

Remarquez aussi ici que l'appel à la librairie ("`m`" pour "math") est placé *après* le nom du code source. Sinon ca va foirer. C'est crétin mais c'est comme ça...

Dans le choix de l'une ou l'autre des formules de différences finies listées ci-dessus pour traiter un problème numériquement, il est recommandé d'utiliser des formules ayant le même ordre de discrétisation pour toutes les dérivées quelque soit leur ordre. Il faut également garder en tête que les erreurs de discrétisation associées aux différentes formules ($O(h^2)$, etc) ne tiennent vraiment que dans la limite d'un maillage serré, i.e., $h \to 0$; si le maillage spatial n'est pas suffisamment serré pour bien résoudre la fonction discrétisée, l'éq. (2.23) produira des résultats probablement d'aussi piètre qualité que l'éq. (2.19). Mais "suffisamment", c'est combien? Il n'y a pas de réponse unique à cette question; mais en première approximation on peut s'attendre à ce que h doive être pas mal plus petit que l'échelle sur laquelle varie la fonction pour qu'un résultat soit crédible.

2.2.4 Précision, troncation et double précision

À strictement parler, toutes les formules de différences finies listées ci-dessus deviennent exactes dans la limite $h \to 0$. Mais évaluées sur un ordinateur où le nombres de chiffres significatifs définissant un nombre réel est restreint, il existe un h en dessous duquel les erreurs d'arrondissement en viennent à dominer. Ceci est illustré sur la Figure 2.7, d'un format identique à la Fig. 2.5 mais s'étendant à des h et ε beaucoup plus petits. On y voit bien que lorsque l'erreur chute sous un seuil allant de 10^{-4} à 10^{-6} (dépendant de la formule de différence finie utilisée), l'erreur ne décroit plus à mesure que h diminue.

Les traits pleins sur la Figure 2.7 utilisent les trois mêmes formules de différences finies, sauf que cette fois les variables ont été déclarées en **double précision**, i.e., en C, la seconde ligne du bloc de déclaration dans le code de la Fig. 2.6 est remplacée par:

```
double dsinxdx, erreur, h, ang ; /* variables locales */
```

La Figure 2.7 illustre bien la subtile différence entre les erreurs de troncation (associée à l'ordre de la formule de différence finie choisie) et les erreurs d'arrondissement, associées au choix de la représentation numérique —ici `float` versus `double`. Une fois que le pas h devient petit au point où les diverses évaluations de $\sin(x \pm h)$ ne diffèrent plus que dans le dernier chiffre significatif pouvant être encodé dans la mantisse M du nombre réel (cf. eq. (1.4)), toutes les formules de différences finies se mettent à déconner, quelque soit leur ordre de troncation. Bien qu'on ait pu imaginer à prime abord qu'il soit souhaitable de choisir un h le plus petit possible afin de justifier notre omission des termes d'ordre élevé en h ($\equiv \Delta x$) dans le développement en série de Taylor, les restrictions associées à nos représentations numériques des nombres réels posent une limite inférieure incontournable à la taille du pas; loin d'améliorer la précision de nos estimés des dérivées, pousser sous cette limite les dégrade!

2.3 Interpolation

Dans bien des situations en sciences, incluant en physique, vous tomberez sur des séries de mesures expérimentales tabulées qui deront être utilisées comme base à des calculs subséquents. Considérons par exemple les "données" portées en graphique sur la Figure 2.8, qui montrent

```
#include <stdio.h>
#include <math.h>
/* Ce programme calcule cos(x) par differences finies,              */
/* pour differentes tailles de pas                                  */
int main(void)
{
/* Declarations ---------------------------------------------------- */
  int k, niter ;
  float dsinxdx, erreur, h, ang ;   /* variables locales */
  float pi=3.1415926536 ;           /* la valeur de pi */
/* Executable ------------------------------------------------------ */
  niter= 95 ;                       /* Nombre de pas testes */
  h    = 1.0 ;                      /* Valeur initiale du pas */
  ang  = pi/6. ;                    /* Derivee a cet angle (radian) */
  for ( k=1 ; k<= niter ; k++ )     /* Boucle sur les tailles de pas */
    {
      dsinxdx = (sin(ang+h)-sin(ang-h))/(2.*h) ; /* Formule O(h2) */
      erreur  = fabs(dsinxdx-cos(ang)) ;         /* Calcul de l'erreur */
      printf ("h: %f   erreur: %f\n", h,erreur) ;
      h       = h/1.1 ;                          /* on reduit le pas */
    }
}
```

Figure 2.6: Code C pour calculer les erreurs d'estimé de la dérivée d'un sinus portées en graphique à la Fig. 2.5. Remarquez comment, à chaque itération de la boucle, la valeur du pas spatial `h` utilisé pour le calcul de la dérivée décroit d'un facteur 1.1. Le descripteur de format `%f` indique que le chiffres "imprimés" en sortie (les variables `h` et `erreur`) sont des `float`, et le `\n` insère un saut de ligne à chaque exécution de l'instruction `printf`.

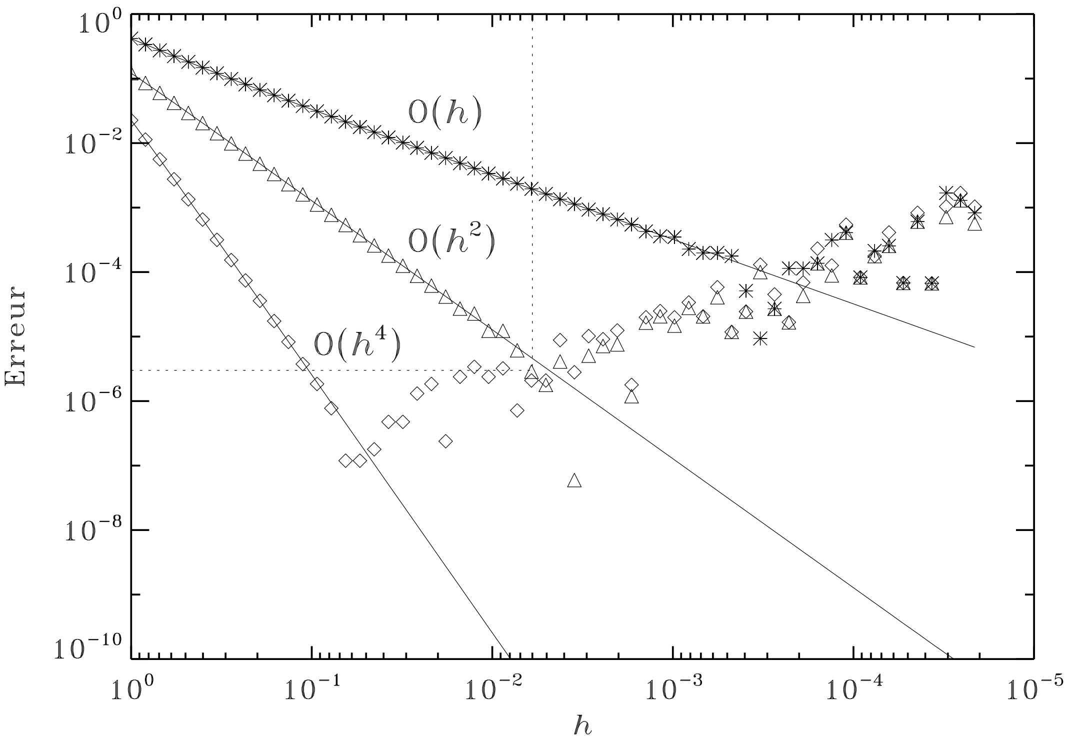


Figure 2.7: Variations de l'erreur de discrétisation en fonction de la taille du pas, toujours pour le calcul de la dérivée de $\sin(x)$ par différences finies. Les symboles donnent l'erreur associée au calcul des dérivées en simple précision, comme sur la Figure 2.5, tandis que les traits plein donnent l'équivalent calculé en double précision. Les traits pointillés délimitent l'intervalle couvert par la Figure 2.5

cinq "mesures" de la variation d'une quantité f en fonction d'une variable x dont dépend f. Ceci est typique des plusieurs types de mesures expérimentales; on peut bien faire un grand nombre de mesures, mais on se retrouve toujours avec un échantillonnage *discret* et *fini* de la quantité mesurée.

Supposons maintenant que sur la base de ces mesures, on vous demande de calculer la valeur que prendrait f si $x = 5$; problème, car vous disposez de mesures à $x = 4$ et $x = 7$, mais pas à $x = 5$. Vous vous trouvez donc dans l'obligation de *supposer* que f varie d'une manière choisie *a priori* entre $x = 4$ et $x = 7$, et ainsi produire un estimé de f à $x = 5$. Ce genre de tâche numérique s'appelle une **interpolation**.

On pourrait aussi vous demander quelquechose à prime abord de complètement différent, par exemple d'évaluer, sur la base des mesures expérimentales, une intégrale définie du genre:

$$\int_1^{10} f(x)\mathrm{d}x \tag{2.28}$$

Comme $f(x)$ n'a pas une forme analytique connue, vous devrez encore une fois *supposer* que f varie d'une manière analytiquement intégrable entre les points où les mesures de f sont disponibles. C'est là le lien fondamental entre l'interpolation et l'évaluation numérique des intégrales définies.

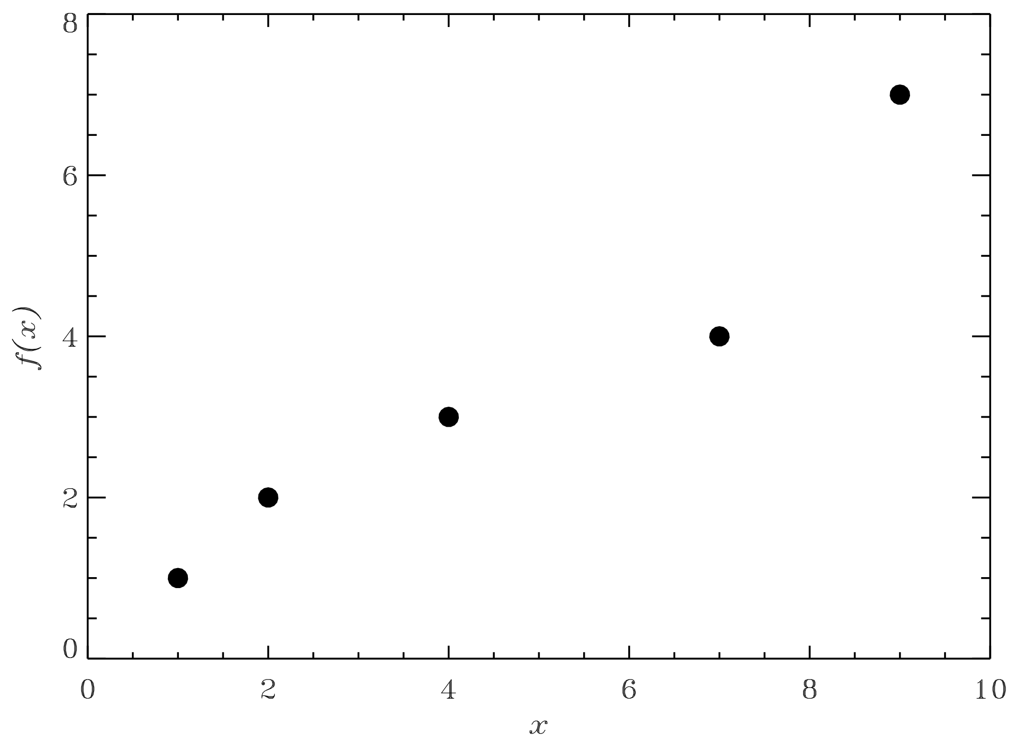


Figure 2.8: Une série de données artificielles correspondant à des mesures d'une certaine quantité $f(x)$ en fonction d'un ensemble fini de valeurs x dont dépend cette quantité.

2.3.1 Interpolation linéaire

Commençons avec l'interpolation, en établissant soigneusement notre notation. Une interpolation typique est définie comme suit: on nous fournit un ensemble de N valeurs numériques f_k d'une fonction mesurée (ou discrétisée) sur une maille x_k également donnée, et pas nécessairement équidistante, comme sur la Fig. 2.8:

$$f(x) \to f_k \equiv f(x_k)\ , \qquad k = 1, 2, ...N\ , \tag{2.29}$$

$$\mathbf{x} = [x_1, x_2, ..., x_N]\ . \tag{2.30}$$

L'interpolation consiste à obtenir un estimé f^* de $f(x_k^*)$ à un ensemble discret de P valeurs x_k^* ne coincidant habituellement pas avec les x_k:

$$\mathbf{x}^* = [x_1^*, x_2^*, ..., x_P^*]\ . \tag{2.31}$$

On exige habituellement que la fonction d'interpolation $f^*(x)$ soit continue, et exacte sur l'ensemble des données étant interpolées, i.e.,

$$f^*(x_k) = f(x_k)\ , \qquad k = 1, 2, ...N\ . \tag{2.32}$$

L'interpolation la plus simple qui satisfait à ces contraintes consiste à approximer la variation de $f(x)$ entre deux points successifs $[x_k, x_{k+1}]$ par une droite. On décrit alors ceci comme une **interpolation linéaire**. L'idée générale est illustrée à la Figure 2.9. Les points (x_k, f_k) définissant la fonction à interpoler sont en noir, et les valeurs interpolées indiquées en gris. Notons déjà que la fonction d'interpolation est l'union ici de quatre tronçons linéaires contigus

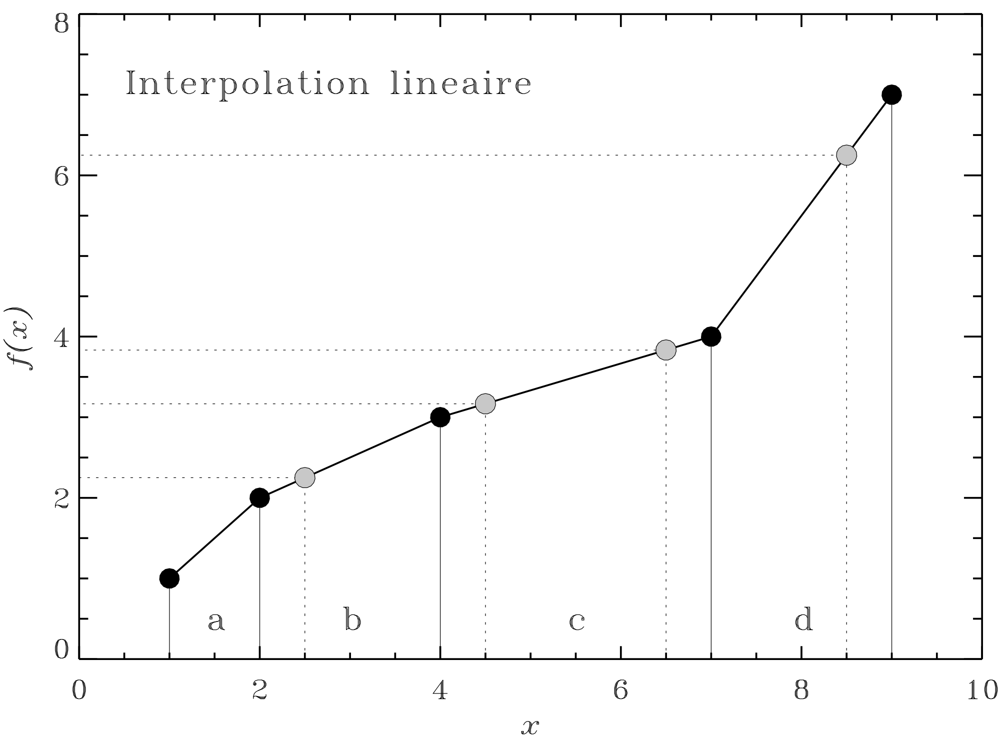


Figure 2.9: Interpolation linéaire d'une fonction discrète définie par $\mathbf{x} = [1, 2, 4, 7, 9]$, $f(\mathbf{x}) = [1, 2, 3, 4, 7]$ ($N = 5$, points noirs), aux points $\mathbf{x}^* = [2.5, 4.5, 6.5, 8.5]$ ($P = 4$, points gris). La fonction d'interpolation est ici définie par quatre plages d'interpolation, indiquées par **a**, **b**, **c** et **d**.

qui se rejoignent aux x_k's; la fonction d'interpolation est donc continue, mais sa dérivé première par rapport à x ne l'est pas, puisqu'elle varie de manière discontinue aux x_k's, et toutes ses dérivées d'ordre supérieur sont nulles.

Une fois la fonction d'interpolation choisie, son évaluation à chaque x_p^* est un processus en trois étapes:

1. trouver la **plage d'interpolation**, soit l'intervalle $[x_k, x_{k+1}]$ tel que $x_k \leq x_p^* < x_{k+1}$;
2. Calcul des paramètres de la droite passant par (x_k, f_k) et (x_{k+1}, f_{k+1}).
3. Calcul de $f_p^* \equiv f^*(x_p^*)$

Chercher la plage

Étant donné un tableau `x` de dimension N contenant les abcisses de la fonction à interpoler, le petit bout de code suivant détermine le k associé à la plage d'interpolation $[x_k, x_{k+1}]$ pour un point d'interpolation `xetoile`$= 4.5$ (disons):

```
k=0 ;
do { k=k+1 } while ( xetoile < x[k] ) ;
ketoile=k-1 ;
```

Remarquez que la valeur de `ketoile` est réduite de un par rapport au dernier x_k testé , puisqu'on cherche ici le k tel que $x_k \leq x^* < x_{k+1}$, et que le test de fin de boucle ne vérifie que la seconde inégalité, $x^* < x_{k+1}$, la première étant automatiquement satisfaite si les x_k sont croissants et distincts. La façon de faire ci-dessus est simple, compacte et lisible, mais n'est pas terriblement efficace du point de vue numérique; on verra plus loin comment faire mieux.

Calculer la droite

Avec `xetoile`= 4.5, nous travaillons donc dans la plage d'interpolation **c**. Une droite est définie paramétriquement par la relation

$$y = mx + b \ , \tag{2.33}$$

où m est la pente et b l'ordonnée à l'origine. Évaluons ceci aux points x_3 et x_4 qui définissent les limites de la plage c:

$$f_3 = mx_3 + b \ , \qquad f_4 = mx_4 + b \ ; \tag{2.34}$$

Comme les x_3,, x_4, f_3 et f_4 sont connus, ceci représente un système de deux équations pour deux inconnues, m et b. La solution en est:

$$m = \frac{f_4 - f_3}{x_4 - x_3} = \frac{4-3}{7-4} = \frac{1}{3} \ , \tag{2.35}$$

$$b = f_3 - \frac{f_4 - f_3}{x_4 - x_3} x_3 = 3 - \frac{4}{3} = \frac{5}{3} \tag{2.36}$$

L'interpolation dans la plage **c** est donc définie par:

$$f^*(x^*) = \frac{x^*}{3} + \frac{5}{3} \ , \qquad x_3 \leq x^* \leq x_4 \ . \tag{2.37}$$

Calculer l'interpolation

C'est ici trivial. On n'a qu'à évaluer l'éq. (2.37) à la valeur de x^* désirée. Dans le cas des données de la Figure 2.9, on aurait donc

$$f^*(x_2^*) = f^*(4.5) = 3.1666... \qquad f^*(x_3^*) = f^*(6.5) = 3.8333... \tag{2.38}$$

N'oubliez surtout pas que l'évaluation de f^* à $x_1^* = 2.5$ et $x_4^* = 8.5$ doit se faire dans les plages **b** et **d** respectivement, où la droite (2.37) n'est pas valide! La code C présenté à la Figure 2.10 effectue l'interpolation illustrée à la Figure 2.9.

2.3.2 Interpolation d'ordre plus élevé

À examiner l'interpolation linéaire présentée à la Figure 2.9, il n'est pas difficile d'imaginer qu'une meilleure interpolation puisse être obtenue en reliant les N points (x_k, f_k) par quelque chose de mieux que des segments de droites, par exemple un polynôme d'ordre plus élevé que $y = mx + b$. Les **polynômes de Lagrange** ne sont effectivement rien de plus que les polynômes classiques avec lesquels vous avez déjà fait connaissance au secondaire (linéaire, quadratique, cubique, etc), mais réécrit d'une manière différente qui s'avère particulièrement appropriée aux tâches d'interpolation. Si le calcul et l'évaluation des polynômes d'interpolation de Lagrange sont très simples (voir références en fin de chapitre), l'interpolation en résultant souffre de certains défauts, non le moindre étant que même pour les interpolations d'ordre plus grand que linéaire, les dérivées de la fonction d'interpolation globale ne sont pas continues aux jonctions des plages d'interpolation, bien qu'elles le soient à l'intérieur de chaque plage individuelle. Ce problème est clairement visible sur la Fig. 2.11, qui montre le résultat d'une interpolation quadratique sur les deux plages de trois points définies par les données de la Fig. 2.8. Les **splines cubiques** permettent d'éviter ce problème. Il ne s'agit de rien de plus qu'une interpolation polynomiale cubique (plage d'interpolation de 4 points), avec une contrainte supplémentaire de

```
#include <stdio.h>
int main(void)
{
/* Declarations ----------------------------------------------------- */
  int k, ketoile ;
  float x[5], y[5] ;
  float xetoile, yetoile, m, b ;
/* Initialisations -------------------------------------------------- */
  x[0]=1. ; x[1]=2. ; x[2]=4. ; x[3]=7. ; x[4]=9. ;
  y[0]=1. ; y[1]=2. ; y[2]=3. ; y[3]=4. ; y[4]=7. ;
/* Executable ------------------------------------------------------- */
  do {
    printf ("SVP donner une valeur de x :") ; scanf ("%f", &xetoile) ;
    if ( xetoile < x[0] || xetoile > x[4] ) {
        printf ("Valeur hors de l'intervalle, essayez de nouveau\n") ;
    }
    else {
      k=-1 ;
      do { k=k+1 ; } while ( xetoile > x[k] ) ;
      ketoile=k-1 ;
      m=(y[ketoile+1]-y[ketoile])/(x[ketoile+1]-x[ketoile]) ;
      b=y[ketoile]-m*x[ketoile] ;
      yetoile=m*xetoile+b ;
      printf ("x= %f f(x)= %f\n", xetoile,yetoile) ;
    }
  } while ( xetoile != 0 ) ;
  printf ("Au revoir...\n") ;
}
```

Figure 2.10: Code C pour l'interpolation linéaire de la Figure 2.9. Ici la fonction discrétisée $f_k \equiv f(x_k)$ est définie directement dans la section de déclarations, mais la valeur x^* à laquelle l'interpolation est désirée est lue au moment de l'exécution, via la fonction `scanf`; une fois l'interpolation terminée, le programme se met en attente de la valeur suivante. Ceci est notre premier exemple d'interaction entre le programme et l'usager durant l'exécution.

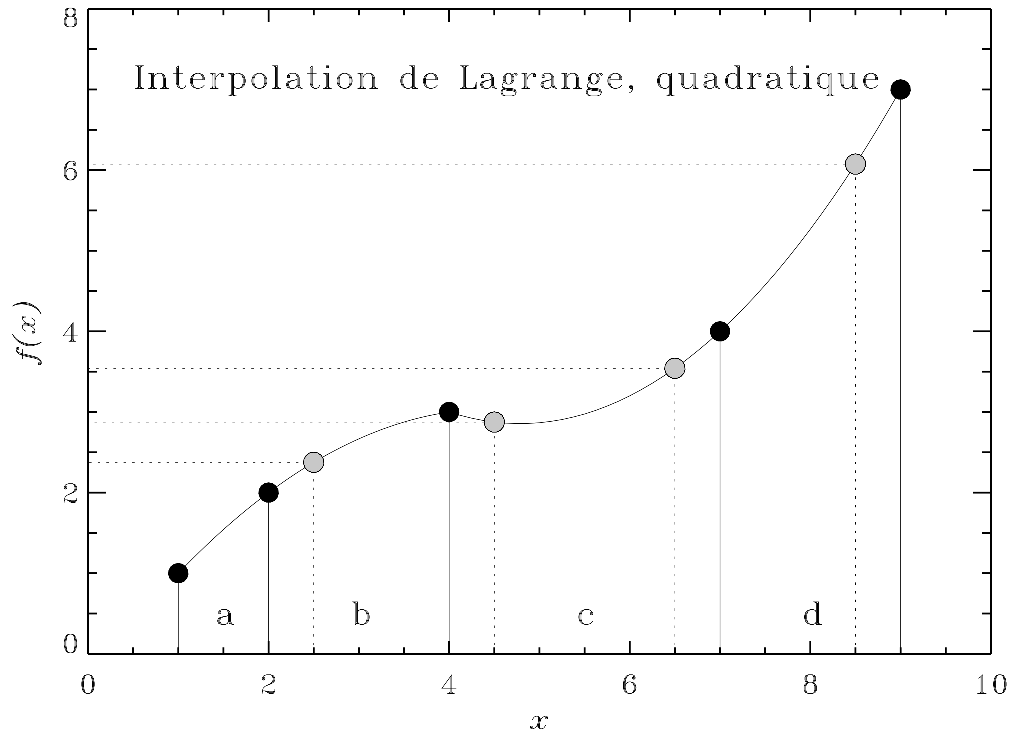


Figure 2.11: Interpolation quadratique par polynômes de Lagrange des données de la Figure 2.8. Avec $N = 5$, on a ici deux plages distinctes d'interpolation, $\mathbf{a}+\mathbf{b}$ et $\mathbf{c}+\mathbf{d}$, l'interpolation dans chacune étant définie par un segment de parabole. Remarquez la discontinuité de la dérivée première à la fonction des deux plages ($x = 4$), et le minimum secondaire produit dans la plage c.

continuité des dérivés premières et secondes aux jonctions des plages d'interpolation. La Figure 2.12 en montre un exemple, toujours pour les mêmes données que précédemment. Comparant aux Figs. 2.9 et 2.11, on constate que la fonction d'interpolation est maintenant beaucoup plus lisse.

La majorité des logiciels de traitements de données ou de graphisme incluent généralement des fonctions prédéfinies effectuant une interpolation par spline cubique. La théorie sous-jacente est expliquée dans la plupart des bouquins d'analyse numérique (voir bibliographie en fin de chapitre).

Comparant les Figures 2.9, 2.11 et 2.12, il est clair que le choix de la méthode d'interpolation peut avoir un impact substantiel sur les valeurs interpolées. Le Tableau 2.1, listant les valeurs numériques de ces trois interpolations aux mêmes x^* confirme cette impression. Les différences, qui atteignent les 10% dans plusieurs cas, n'ont rien à voir avec des imprécisions dans les données, des erreurs d'arrondissement, ou tout autre truc du genre; elles résultent entièrement du choix de la méthode d'interpolation. Laquelle est la meilleure ici? Difficile à dire...

En général, on pourrait penser que l'interpolation par splines cubiques est à préférer, si ce n'est parce que cette technique d'interpolation assure la continuité des dérivées de la fonction interpolée aux frontières des plages d'interpolation. Mais les splines ne doivent pas être utilisés aveuglément. La Figure 2.13 présente un exemple d'une situation où une interpolation en splines cubiques produit des résultats plus que douteux. Ici la fonction à interpoler (points noirs) a l'allure une fonction escalier, avec un $f(x)$ constant pour $x \leq 3$, sautant à une valeur constante plus élevée pour $x \geq 4$. Il est important de réaliser que les données ne contiennent aucune information nous permettant de spécifier où exactement entre $x = 3$ et $x = 4$ la discontinuité

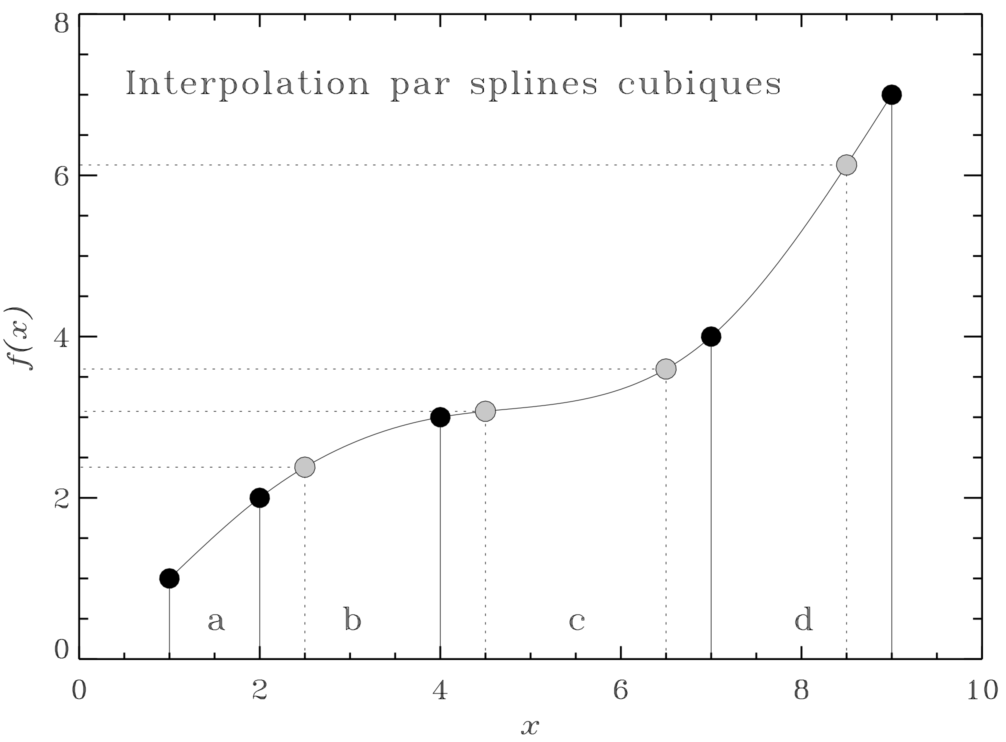


Figure 2.12: Interpolation par splines cubiques, à partir des mêmes données que sur les Figure 2.9 et 2.11. Les valeurs de $f(x)$ aux points interpolés (gris) ne diffèrent pas énormément de celles obtenues avec l'interpolation linéaire, mais ici les dérivées première et seconde sont continues partout.

est située,... ou même si il y a vraiment une discontinuité dans l'intervalle, par opposition à une variation très rapide mais continue en x.

Dans une telle situation l'interpolation par spline cubique produit une variation rapide et continue entre $x = 3$ et $x = 4$, comme on s'y attendrait d'une polynôme d'ordre relativement élevé, mais dans les deux intervalles voisins la contrainte de continuité de la fonction **et** de ses dérivées caractérisant les splines cubiques produit des "sursauts" où f^* s'éloigne substantiellement de l'intervalle défini par les deux valeurs constantes de la fonction (discrétisée) f étant interpolée ici. Dans un cas comme celui-ci, une interpolation linéaire (tirets) serait probablement préférable.

Dernier commentaire de nature quasi-philosophique: qu'une interpolation se fasse par polynôme de Lagrange ou par spline cubique, ultimement c'est de la tricherie; il n'y a tout simplement pas d'information entre les points (x_k, f_k) où la fontion est échantillonnée! Le choix de l'une ou l'autre type d'interpolation est tout à fait arbitraire, et le plus souvent déterminé par nos préjugés et/ou attentes par rapport au système ayant "généré" les données (x_k, f_k).

2.3.3 Interpolation versus extrapolation

L'**extrapolation** est une procédure douteuse basée sur la même idée que l'interpolation, soit la représentation d'un échantillonnage discret d'une fonction inconnue par une fonction (habituellement simple) de la variable indépendante. La Figure 2.14 illustre le danger inhérent à ce type de manoeuvre. On travaille toujours ici avec les données de la Figure 2.8 (points noirs), mais les polynômes d'interpolation sont maintenant utilisés pour calculer une extrapolation à des valeurs de x^* qui sont *à l'extérieur* de l'intervalle couvert par les données ($1 \leq x \leq 9$).

Table 2.1: Comparaison de trois interpolations des mêmes données

x^*	Linéaire	Quadratique	Spline Cubique
2.5	2.25	2.37500	2.38013
4.5	3.16667	2.87500	3.07348
6.5	3.83333	3.54167	3.59796
8.5	6.25000	6.07500	6.12895

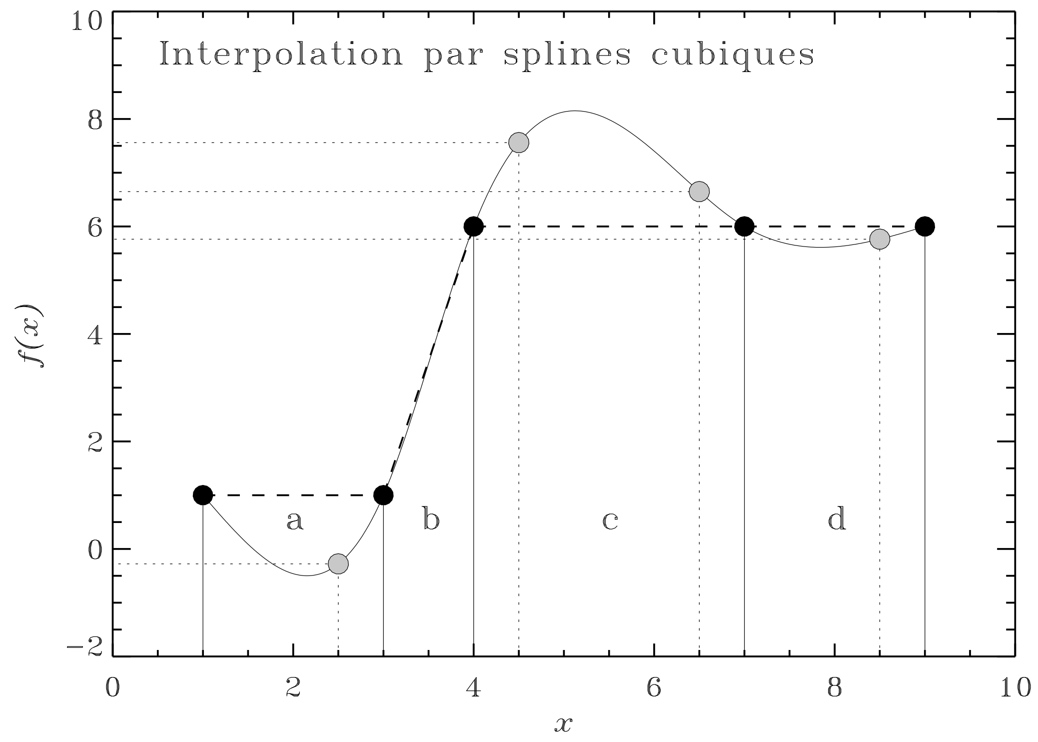


Figure 2.13: Exemple d'une situation où une interpolation linéaire (tirets) est très possiblement plus appropriée qu'une interpolation par splines cubiques.

Ici l'extrapolation linéaire (tirets) vers $x^* = 15$ utilise l'interpolation linéaire dans la plage **d**, tandis que l'extrapolation quadratique (trait plein) est basée sur un polynôme quadratique couvrant les plages $\mathbf{c} + \mathbf{d}$. Bien que dans la plage **d** les deux interpolations ne diffèrent que par quelques pourcents (cf. Tableau 2.1 pour $x^* = 8.5$), ici les deux extrapolations à $x^* = 15$ diffèrent par pas loin d'un facteur deux. Je vous laisse imaginer de quoi tout ça aurait l'air si on avait la brillante idée d'extrapoler jusqu'à $x^* = 100$...

Dans le cas spécifique des données de la Figure 2.14, l'extrapolation jusqu'à $x^* \simeq 10$ produirait des résultats ne divergeant pas plus que pour les interpolations compilées au Tableau 2.1; on pourrait en juger que l'extrapolation est alors "acceptable". Il n'en demeure pas moins que l'extrapolation est toujours un jeu *très* délicat à tous les points de vue.

L'extrapolation demeure parfois inévitable et nécessaire. La Figure 2.15 vous en montre un des exemple les plus médiatisé des dernières quelques décennies, soit l'augmentation prédite de la température moyenne de la Terre au cours du vingt-et-unième siècle. Huit extrapolations climatiques sont portées en graphique ici, chacune correspondant à un modèle climatique différent; différent pas seulement au sens purement numérique, mais aussi au niveau des processus physiques inclus, des hypothèses faites au niveau de l'évolution de la population planétaire

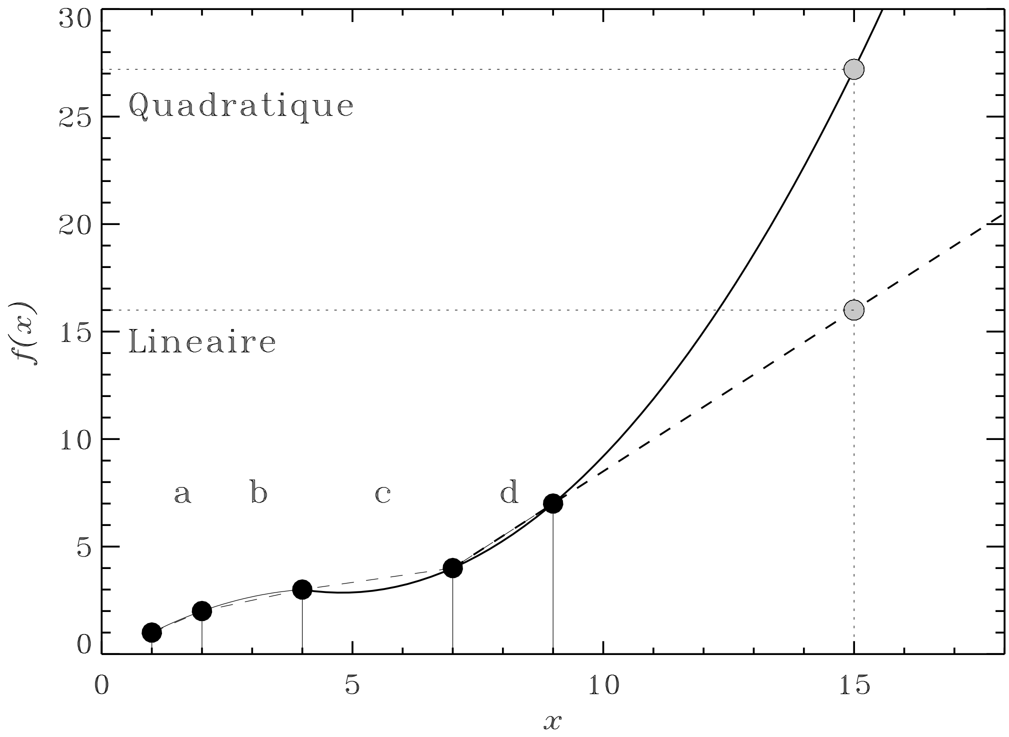


Figure 2.14: Exemple d'une situation où le choix de l'une ou l'autre méthode d'interpolation (linéaire et quadratique) conduit à des résultats divergents lorsque l'interpolation est utilisée pour extrapoler en dehors ($x^* = 15$) des plages d'interpolation ($1 \leq x \leq 9$). Les données (points noirs) sont les mêmes que sur les Figures précédentes.

et de ses habitudes énergivores, etc.

La "pathologie" est fondamentalement la même que sur l'exemple simpliste de la Fig. 2.14. Tous ces modèles climatiques sont "ajustés" de manière à bien reproduire la température moyenne observée dans la seconde moitié du vingtième siècle, et l'accord entre les modèle y est donc excellent (tout comme pour les traits plein et pointillés dans l'intervalle $1 \leq x \leq 9$ sur la Fig. 2.14). Mais une fois extrapolés jusqu'en 2100, le réchauffement prédit varie d'un peu plus de 2 degrés C à presque 5°C. Les extrapolations ne sont pas contraintes, et donc évoluent selon la dynamique interne (très complexe) de chaque modèle.

Une différence de 3 degrés dans la température moyenne terrestre peut paraitre mineure à prime abord, mais en pratique elle est en fait énorme: à +2°C, il y a encore des écosystèmes arctiques; à +5°C, non; À +5°C Bangkok (population métropolitaine: 14.6 million d'habitants en 2010) est complètement sous l'eau; à +2°C, pas encore. Maintenant vous savez pourquoi, si vous commencez à parlez d'extrapolation à un(e) climatologue, vous allez sentir très vite le niveau de stress monter...

2.4 Intégration

Passons maintenant au calcul numérique des intégrales. De la même façon que la dérivée d'une fonction peut être interprétée comme la pente d'une droite tangente à la courbe définie par la fonction, son intégrale peut être interprétée comme l'aire sous la courbe. Tout comme la dérivée (voir éq. (2.1)), l'intégrale peut être définie mathématiquement comme un passage à la

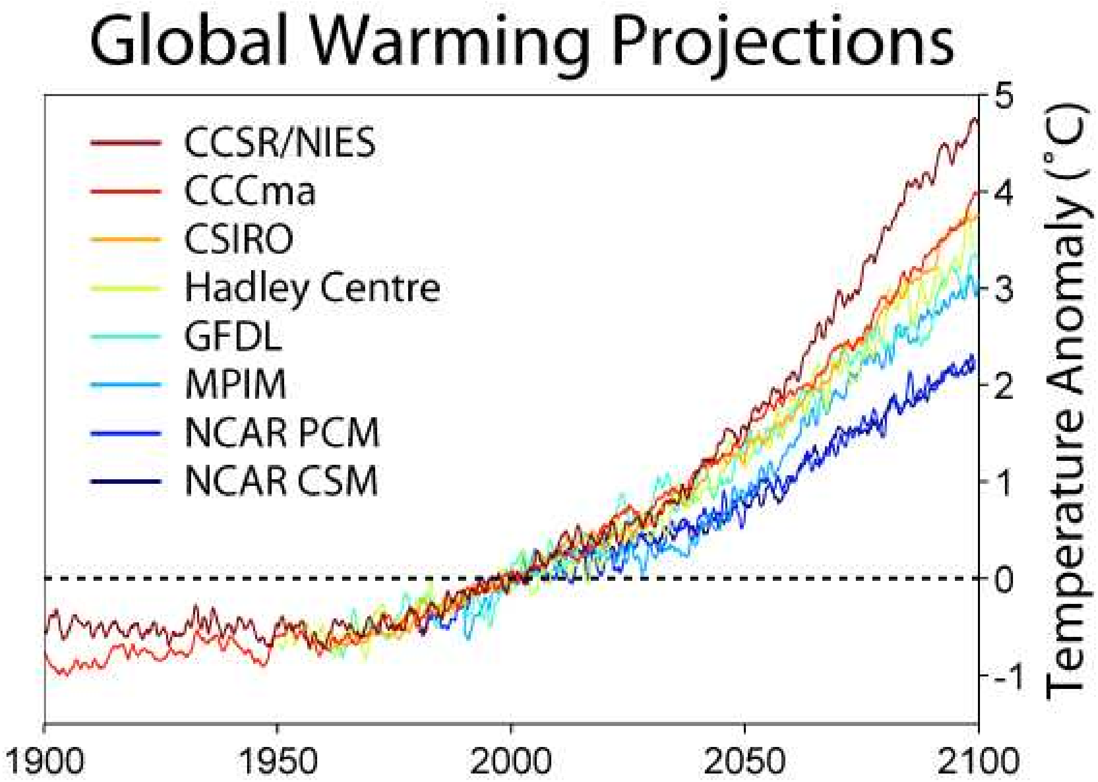


Figure 2.15: Exemple concret d'une extrapolation, soit ici la température moyenne de la Terre jusqu'en 2100. Les prédictions ont été faites en 2007, à l'aide de huit modèles climatiques distincts, tous très "haut-de-gamme". Toutes les extrapolations sont calibrées sur la seconde moitié du vingtième siècle, et donc y sont en excellent accord, mais divergent par la suite. Source: P.A. Rohde/The Global Warming Art Project, téléchargé de `http://en.wikipedia.org/wiki/File:Global_Warming_Predictions.png`.

limite $\Delta x \to 0$:

$$\int_a^b f(x)\mathrm{d}x = \lim_{\Delta x \to 0} \sum \bar{f}(x)\Delta x \tag{2.39}$$

où $\bar{f}$ est une valeur représentative de $f(x)$ sur l'intervalle Δx. L'idée de ce passage à la limite conduisant à "l'aire sous la courbe" est illustrée schématiquement à la Figure 2.16.

Si la fonction $f(x)$ est calculable pour tout x, le membre de droite de l'équation (2.39) peut être directement utilisée pour évaluer l'intégrale, en choisissant un Δx suffisamment petit (en se méfiant toutefois des erreurs d'arrondissement...). Cependant, pour une fonction tabulée, comme sur la Figure 2.8, où la grandeur du Δx est fixée, l'utilisation de l'éq. (2.39) peut conduire à des résultats plutôt imprécis. Il est en fait assez facile de faire mieux.

2.4.1 La méthode du trapèze

La **méthode du trapèze** est un algorithme simple et robuste permettant d'intégrer numériquement une fonction $f(x)$ échantillonnée sur une maille couvrant le domaine d'intégration, mais dont les incréments ne sont pas nécessairement équidistants:

$$(x_1, x_2, x_3, ..., x_N) \ , \qquad x_{k+1} - x_k \neq x_k - x_{k-1} \ .$$

L'idée derrière la méthode du trapèze est simplement de supposer que la fonction $f(x)$ varie *linéairement* entre les points où elle est échantillonnée; l'aire sous le segment de droite reliant

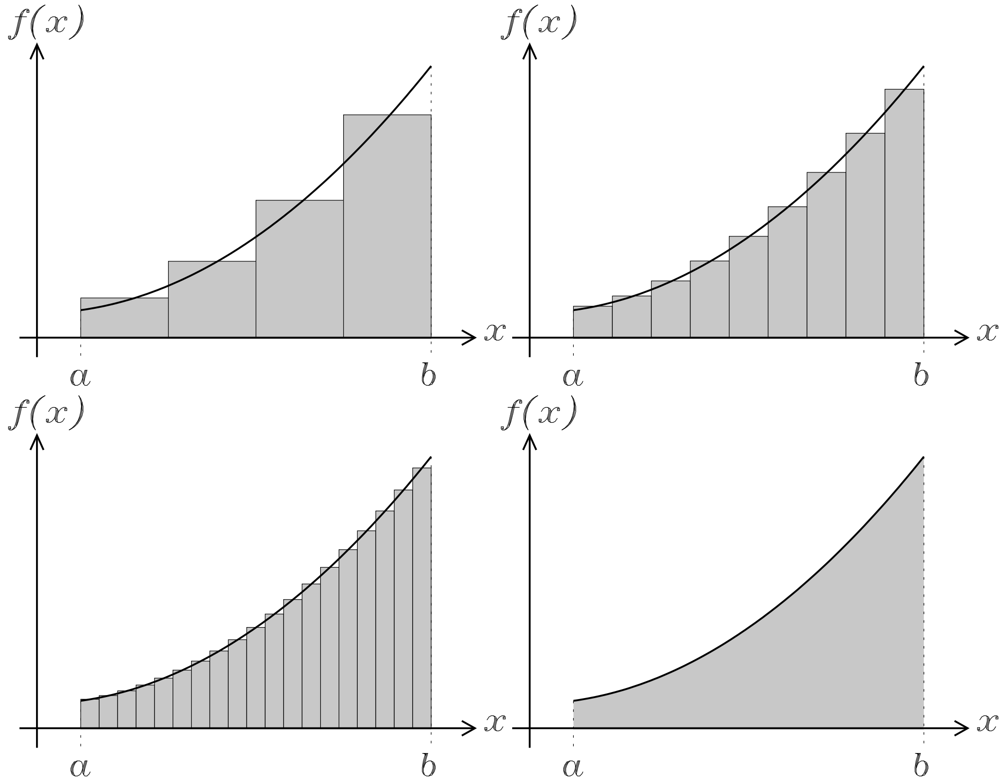


Figure 2.16: Le passage à la limite $\Delta x \to 0$ dans le calcul d'une intégrale définie, correspondant à l'aire contenue sous la courbe définie par $f(x)$ entre les bornes d'intégrations a et b.

les points (x_k, f_k) et (x_{k+1}, f_{k+1}) est:

$$\underbrace{f_k \times (x_{k+1} - x_k)}_{\text{rectangle}} + \underbrace{(f_{k+1} - f_k) \times (x_{k+1} - x_k)/2}_{\text{triangle}} = \frac{(f_{k+1} + f_k)}{2}(x_{k+1} - x_k) ,$$

d'où:

$$\int_{x_1}^{x_N} f(x)\,\mathrm{d}x \simeq \sum_{k=1}^{N-1} \frac{(f_{k+1} + f_k)}{2}(x_{k+1} - x_k) + O(h^2) , \tag{2.40}$$

avec comme d'habitude $f_k \equiv f(x_k)$. Plus la maille est serrée, plus la somme discrète au coté droit de cette expression approximera bien l'intégrale au coté gauche. Dit autrement, plus la maille est petite, plus un développement en série de Taylor tronqué après le second terme offre une représentation adéquate de la variation de la fonction dans l'intervalle d'intégration. L'erreur associée à la méthode du trapèze est formellement $O(h^2)$. En C, ça pourrait avoir l'air de ceci:

```
sum = 0. ;                                  /* variable d'accumulation */
for (k=0 ; k<N-1 ; k++) {
   sum+= 0.5*(f[k+1]+f[k])*(x[k+1]-x[k])    /* eq. (2.40) */
}
```

Ceci présuppose que les valeurs de f et x ont été déjà placées dans deux tableaux 1D chacun de longueur N. Notez l'utilisation d'une variable d'accumulation `sum`, qui, à la sortir de la boucle, contiendra la valeur de l'intégrale. Il faut s'assurer d'initialiser cette variable à zéro avant le

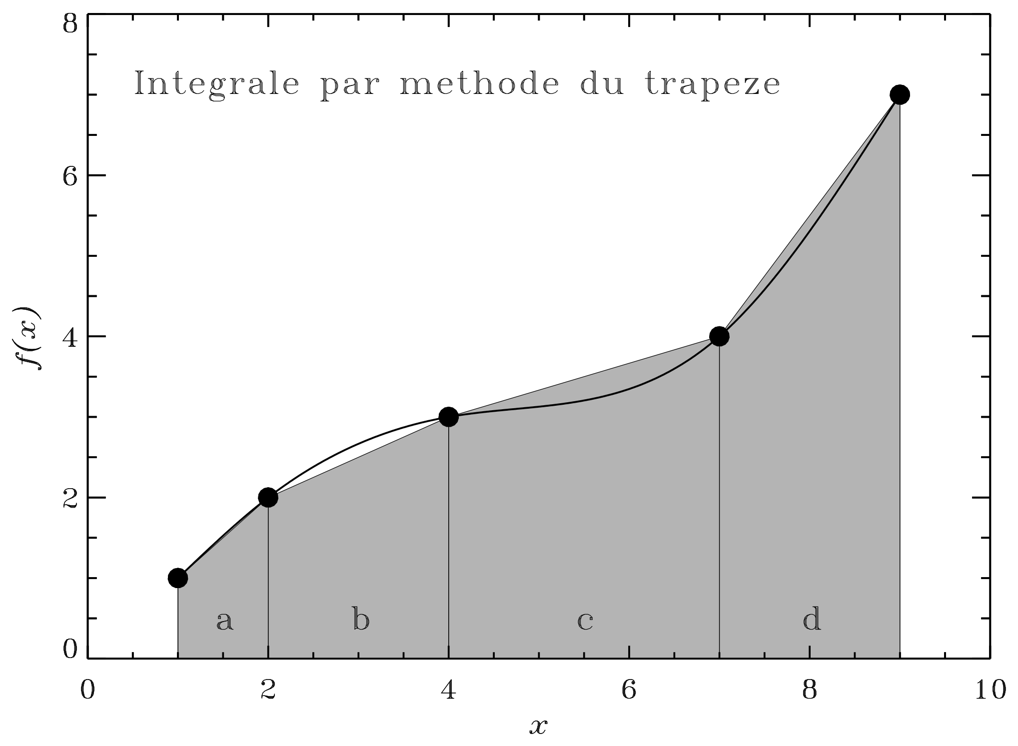


Figure 2.17: Calcul d'une intégrale définie par la méthode du trapèze. La fonction à intégrer (trait noir) est ici un spline cubique basé sur les cinq valeurs discrètes indiquées par des points noirs.

début de la boucle calculant l'intégrale. Notez également que la boucle terminera à $k = N - 2$ plutôt que $N - 1$, de manière à éviter un débordement de tableau sur évaluation de x_{k+1} et f_{k+1} à l'intérieur de la boucle.

La Figure 2.17 illustre l'application de la méthode du trapèze aux données de la Figure 2.9. Si ces données correspondaient vraiment à la fonction définie par le trait plein, on voit que la méthode du trapèze surestimerait les contribution à l'intégrale des plages **c** et **d**, sous-estimerait celle de la plage **b**, et serait très raisonnable pour ce qui est de la contribution de la plage **a**.

2.4.2 Les règles de Simpson

L'idée d'introduire une forme d'interpolation entre les points de maille pour calculer une intégrale est évidemment généralisable à des interpolations d'ordre plus élevé que linéaire. Dans le cas d'un maillage équidistant ($x_{k+1} - x_k = h \, \forall k$), on peut ainsi produire des formules d'intégration relativement simples, connues sous le nom de **Règles de Simpson**. Voir les références cités en fin de chapitre si vous désirez approfondir le sujet.

2.4.3 Intégration de Romberg

L'intégration de Romberg est un petit truc vraiment remarquable pour grandement améliorer la précision d'une intégrale numérique, avec très peu d'effort supplémentaire. Considérons par exemple l'utilisation de la méthode du trapèze sur une maille équidistante $h = h_1$. Pour une maille suffisamment serrée, la solution obtenue ($I^{(1)}$, disons) différera de la solution exacte (I^*) par un facteur proportionnel à h_1^2; on peut donc écrire en toute généralité:

$$I^{(1)} = I^* + Ch_1^2 \ , \tag{2.41}$$

où C est une constante qui est en fait une propriété de l'algorithme appliqué au problème donné, mais pas du choix de la maille. Considérons maintenant une seconde solution obtenue sur une maille $h_2 \neq h_1$. On aura encore une fois

$$I^{(2)} = I^* + Ch_2^2 \ , \tag{2.42}$$

avec le même C. Les équations (2.41) et (2.42) forment un système de deux équations pour deux inconnues, I^* et C; ce système peut être solutionné pour produire une évaluation de I^*:

$$I^* = \frac{h_2^2 I^{(1)} - h_1^2 I^{(2)}}{h_2^2 - h_1^2} \ , \qquad \text{methodes } O(h^2) \tag{2.43}$$

Ceci représente en fait une forme d'extrapolation; il sera donc judicieux de choisir une seconde maille h_2 différant de manière significative de la première, e.g., $h_2 = h_1/2$.

2.5 Coda, en plus qu'une variable

Nous allons clore ce chapitre par une brève présentation de la généralisation des formules de différences finies, d'interpolation et d'intégration à plus d'une variable spatiale.

2.5.1 Discrétisation

La procédure de discrétisation d'une fonction continue introduite à la §2.2.1 se généralise facilement à une fonction de plus d'une variable. Considérons par exemple une fonction $f(x, y)$ de deux variables, discrétisée sur une maille cartésienne régulière en deux dimensions spatiales (voir Figure 2.18):

$$x \to \{x_1, x_2, ..., x_N\}, \qquad x_{j+1} > x_j \ , \tag{2.44}$$

$$y \to \{y_1, y_2, ..., y_M\}, \qquad y_{k+1} > y_k \ . \tag{2.45}$$

Une maille est dite **cartésienne** si toutes les lignes $x =$constante sont des droites parallèles les unes aux autres, les $y =$constante le sont également, et chaque famille de droites coupe l'autre en formant des angles droits; autrement dit, comme sur du papier quadrillé! On identifie chaque noeud du maillage 2D par une paire d'indices (j, k), et on écrit pour la fonction discrétisée:

$$f_{j,k} \equiv f(x_j, y_k) \ , \tag{2.46}$$

Je vous laisse imaginer comment ceci se généralise à des fonctions de plus de deux variables.

2.5.2 Dérivées partielles et différences finies

Commençons donc avec l'évaluation de dérivées par différences finies. Pour des fonctions de plus d'une variable, on calcule habituellement les **dérivées partielles**, qui ne sont rien de plus que les dérivées de la fonction par rapport à une de ses variables indépendantes, considérant les autres fixes. Un bon petit exemple vaudra bien un long paragraphe; considérons la fonction de deux variables suivante:

$$f(x, y) = 1 + xy - y^2 \tag{2.47}$$

Les deux dérivées partielles de cette fonctions seront données par

$$\frac{\partial f(x,y)}{\partial x} = y \ , \qquad \frac{\partial f(x,y)}{\partial y} = x - 2y \tag{2.48}$$

Remarquez bien comment, du point de vue de la dérivée partielle par rapport à x, le terme $-y^2$ est considéré constant, et donc sa dérivée est nulle, tout comme l'est la dérivée du premier

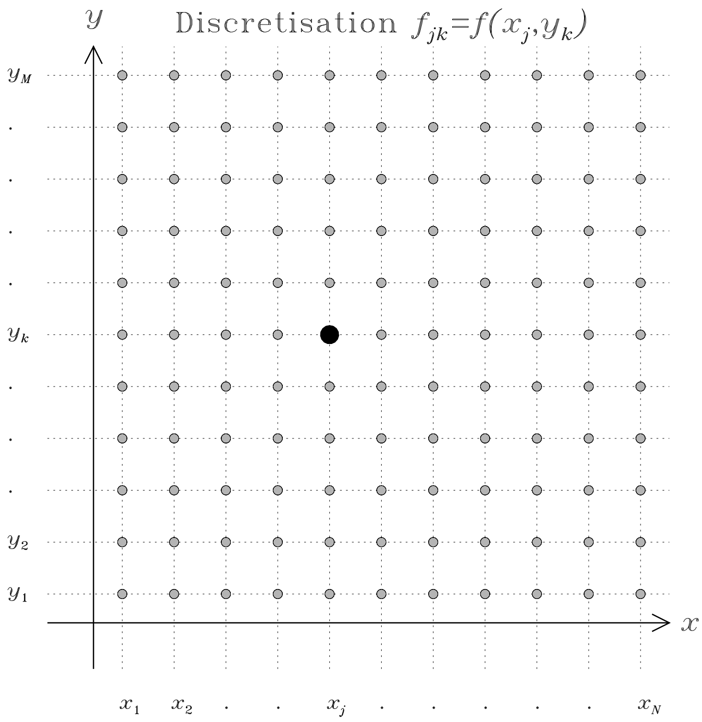


Figure 2.18: Discretisation d'une fonction de deux variables $f(x, y)$ sur un maillage cartésien régulier, ici équidistant dans les direction x et y et avec $N = M = 11$. Le noeud (j, k) est indiqué en noir.

terme au membre de droite de l'éq. (2.47). Ca continue de la même manière pour les dérivées secondes:

$$\frac{\partial^2 f}{\partial x^2} = \frac{\partial}{\partial x}\frac{\partial f(x,y)}{\partial x} = 0 \ , \qquad \frac{\partial^2 f}{\partial y^2} = \frac{\partial}{\partial y}\frac{\partial f(x,y)}{\partial y} = -2 \ . \tag{2.49}$$

Notez que pour les dérivées secondes d'une fonction de deux variables, il existe aussi des dérivées dites croisées; ici on aurait:

$$\frac{\partial^2 f}{\partial y \partial x} = \frac{\partial}{\partial y}\frac{\partial f(x,y)}{\partial x} = 1 \ , \qquad \frac{\partial^2 f}{\partial x \partial y} = \frac{\partial}{\partial x}\frac{\partial f(x,y)}{\partial y} = 1 \ . \tag{2.50}$$

Ce n'est pas un hasard si les deux dérivées croisées donnent ici le même résultat; ce sera toujours le cas pour une fonction analytique, i.e., sans discontinuité dans la fonction ou ses dérivées de tout ordre.

La procédure décrite ci-dessus peut être appliquée au calcul des dérivées partielles d'une fonction de plus d'une variable. Pour les formules d'ordres un et deux les expressions correspondantes sont des applications directes des formules listées ci-dessus, avec les autres coordonnées fixées: par exemple, sur une maille 2D:

$$\left.\frac{\partial f}{\partial x}\right|_{x_j, y_k} = \frac{f_{j+1,k} - f_{j-1,k}}{2\Delta x} + O(h^2) \ , \tag{2.51}$$

$$\left.\frac{\partial f}{\partial y}\right|_{x_j, y_k} = \frac{f_{j,k+1} - f_{j,k-1}}{2\Delta y} + O(h^2) \ , \tag{2.52}$$

$$\left.\frac{\partial^2 f}{\partial x^2}\right|_{x_j,y_k} = \frac{f_{j+1,k} - 2f_{j,k} + f_{j-1,k}}{(\Delta x)^2} + O(h^2) \ , \tag{2.53}$$

$$\left.\frac{\partial^2 f}{\partial y^2}\right|_{x_j,y_k} = \frac{f_{j,k+1} - 2f_{j,k} + f_{j,k-1}}{(\Delta y)^2} + O(h^2) \ , \tag{2.54}$$

où $f_{j,k} \equiv f(x_j, y_k)$, et on a considéré que l'intervalle de maille en x ($h = \Delta x$) puisse être différent de l'intervalle en y ($h = \Delta y$), ce qui est tout-à-fait légal sur une maille Cartésienne 2D. Notons que pour les formules d'ordre plus élevé, les noeuds diagonaux (e.g. $f_{j+1,k+1}$) font leur apparition. Voir les références citées en bibliographie pour plus de détails.

2.5.3 Interpolation multidimensionnelle

En plus d'une dimension, l'interpolation d'ordre élevée (genre spline) devient rapidement très lourde; pour la majorité des situations auxquelles vous pourriez faire face, l'interpolation multi-linéaire devrait suffire. La première étape consiste à identifier la plage d'interpolation. Dans le cas d'un maillage cartésien 2D comme sur la Figure 2.18, la procédure utilisée en 1D (§2.3.1) est utilisée séparément dans les directions x et y. La plage d'interpolation se retrouve à prendre la forme d'un rectangle dont les coins correspondent aux quatre noeuds (x_j, y_k), (x_{j+1}, y_k), (x_j, y_{k+1}), (x_{j+1}, y_{k+1}). Il s'agit ensuite simplement de représenter, dans la plage d'interpolation, la variation de $f(x, y)$ comme un produit de fonctions linéaires le long de chaque axe de coordonnées. L'algèbre est un peu fastidieuse, mais conduit éventuellement à la formule d'interpolation dite **bilinéaire**:

$$\begin{aligned} f^*(x^*, y^*) &= \frac{(x^* - x_j)(y^* - y_k)}{(x_{j+1} - x_j)(y_{k+1} - y_k)} f_{j+1,k+1} \\ &+ \frac{(x_{j+1} - x^*)(y^* - y_k)}{(x_{j+1} - x_j)(y_{k+1} - y_k)} f_{j,k+1} \\ &+ \frac{(x_{j+1} - x^*)(y_k - y^*)}{(x_{j+1} - x_j)(y_{k+1} - y_k)} f_{j,k} \\ &+ \frac{(x^* - x_j)(y_k - y^*)}{(x_{j+1} - x_j)(y_{k+1} - y_k)} f_{j+1,k} \ . \end{aligned} \tag{2.55}$$

Ceci revient a calculer une moyenne pondéré des valeurs de f évaluée aux quatres noeuds définissant les coins de la plage d'interpolation, avec des facteurs de pondération qui confèrent un poids proportionnellement plus grand aux noeuds situés plus près de (x^*, y^*). Plus spécifiquement, la pondération est proportionnelle à la surface du rectangle défini par les droites $x = x^*$ et $y = y^*$ situé en direction diagonalement opposée à chaque noeud, comme l'illustre la Figure 2.19. Notez également que la forme même de l'éq. (2.55) garantit que

$$f^*(x_j, y_k) = f_{j,k} \ , \tag{2.56}$$

et en particulier, si (x^*, y^*) est situé exactement au centre de la plage d'interpolation, la relation ci-dessus se réduit à:

$$f^*(x^*, y^*) = \frac{1}{4}(f_{j+1,k+1} + f_{j,k+1} + f_{j,k} + f_{j+1,k}) \ , \tag{2.57}$$

soit la moyenne des quatres valeurs de f aux quatre coins de la plage d'interpolation. Ce résultat nous servira maintenant à établir un équivalent de la règle du trapèze pour une intégrale en deux dimensions spatiales.

2.5.4 Intégrales multidimensionnelles

Si ce n'est pas déjà fait, vous aurez l'occasion de tout apprendre sur les intégrales de fonctions de plus d'une variable dans vos cours MAT. Dans le cas de fonctions de deux variables, l'analogie

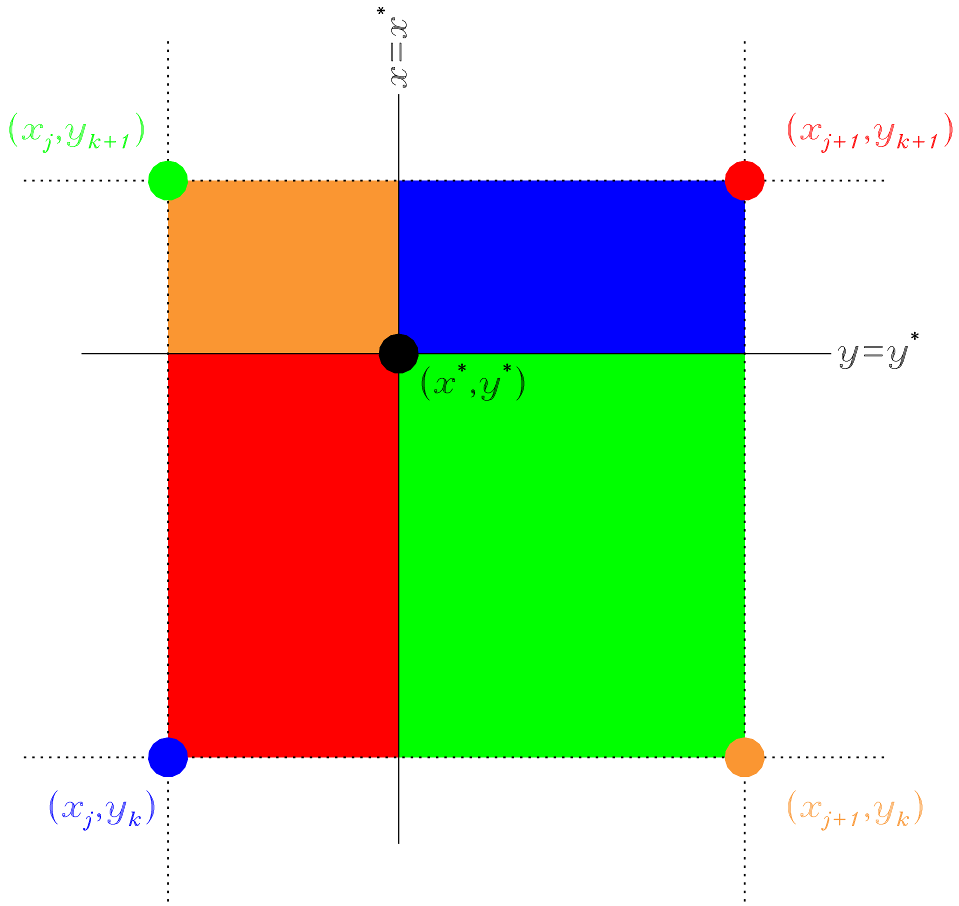


Figure 2.19: Interpolation bilinéaire en deux dimensions spatiales. L'interpolation de $f(x,y)$ au point (x^*,y^*) revient au calcul d'une moyenne pondérée de f évaluée aux quatre noeuds définissant la plage d'interpolation, avec une pondération donnée par la surface du rectangle diagonalement opposé à chaque noeud, tel qu'indiquée ici par le code couleur.

avec les intégrales de fonctions d'une seule variable est simple: cette dernière correspond à calculer "l'aire sous la courbe" délimitée par les bornes d'intégrations; pour une fonction de deux variables, l'intégrale par rapport à ces deux variables correspond à calculer le "volume sous la surface" (2D) délimitée par les bornes d'intégration, qui sont maintenant des lignes ou courbes dans le plan $[x,y]$.

Notez que dans certains cas, les intégrales sont **séparables** et on peut tout simplement les évaluer comme le produit de deux intégrales 1D; e.g.:

$$\int\int e^{x+y}\,\mathrm{d}x\,\mathrm{d}y = \int\int e^x e^y\,\mathrm{d}x\,\mathrm{d}y = \left(\int e^x\,\mathrm{d}x\right)\times\left(\int e^y \mathrm{d}y\right) \ , \tag{2.58}$$

tandis que d'autres ne le sont pas, e.g.:

$$\int\int \sqrt{x^2+y^2}\,\mathrm{d}x\,\mathrm{d}y \ . \tag{2.59}$$

Supposons que la fonction $f(x,y)$ à intégrer est définie sur une maille cartésienne régulière (comme sur la Fig. 2.18). À partir de la fonction d'interpolation bilinéaire introduite ci-dessus, il est de nouveau possible, par intégration directe, d'obtenir l'équivalent de la règle du trapèze applicable maintenant en deux dimensions spatiales:

$$\int\int f(x,y)\mathrm{d}x\,\mathrm{d}y = \sum_{j=1}^{N-1}\sum_{k=1}^{M-1}\frac{1}{4}(f_{j,k}+f_{j+1,k}+f_{j,k+1}+f_{j+1,k+1})(x_{j+1}-x_j)(y_{k+1}-y_k) \tag{2.60}$$

Remarquez que le nombre d'évaluations requises de la fonction f à intégrer augmente géométriquement avec la dimensionalité de l'intégrale; autrement dit, si N intervalles sont requis pour bien représenter les variations de f dans l'une ou l'autre dimension spatiale, alors l'éq. (2.60) exigera $(N+1)^2$ évaluations de f. Qui plus est, son équivalent en D dimensions demanderait $(N+1)^D$ évaluations. On verra plus loin des alternatives moins couteuses au calcul des intégrales en plusieurs dimensions.

Exercices:

1. Calculez le développement de Taylor d'un polynôme quartique d'ordre 4 de la forme générale:
$$f(x) = ax^4 + bx^3 + cx^2 + dx + e$$
Commencez par démontrez qu'à partir de la troncation à l'ordre 4, le développement devient exact quelque soit la grandeur de Δx. Ensuite,
   (a) Calculez maintenant le développement de Taylor d'une fonction sinus:
   $$f(x) = \sin(x) \ ;$$
   (b) Comparez la précision des estimés $f(x_0 + \Delta x)$ par rapport à la valeur exacte pour $x_0 = \pi/6$ (rad) et $\Delta x = 0.01$ rad, pour les développements tronqués aux ordres 1 à 4.
   (c) Démontrez qu'aucun ordre fini de troncation ne peut conduire à un estimé exact, contrairement au cas du polynôme quartique.
2. Écrivez un petit code en C qui calcule la factorielle d'un entier n (voir éq. (2.3)).
3. À partir des formules de différences finies centrées pour les dérivées première et seconde (éqs. 2.18) et (éqs. 2.19)), obtenez les formules de différences fineis centrées pour les dérivées troisième et quatrième (éqs. 2.20) et (éqs. 2.21)).
4. Établissez la forme analytique de l'interpolation linéaire d'une fonction $f(x)$ entre deux points x_1 et x_2 où la fonction vaut f_1 et f_2, respectivement. Intégrez cette interpolation par rapport à x entre x_1 et x_2, et vérifiez que vous retrouvez bien la règle du trapèze.
5. Vérifiez que la solution du système d'équations algébriques (2.41)—(2.42) conduit bien à l'éq. (2.43).
6. Considérez l'intégrale définie suivante:
$$\int_0^1 \sin(2\pi x)\mathrm{d}x \ ,$$
   (a) Calculez l'intégrale analytiquement;
   (b) Calculez numériquement l'intégrale à l'aide de la méthode du trapèze et Simpson 1/8 pour $h = 0.1$, 0.01 et 10^{-3}. Vérifiez que vos résultats sont compatibles avec les taux de convergence attendus pour ces deux algorithmes.
   (c) Utilisez maintenant vos résultats obtenus par méthode du trapèze à $h = 0.01$ et $h = 10^{-3}$ pour obtenir un estimé amélioré de l'intégrale via la méthode de Romberg.
7. Démontrez que l'intégration explicite d'une interpolation bilinéaire de la forme donnée par l'éq. (2.55) conduit bien à la formule de quadrature donnée par l'éq. (2.60)
8. Utilisez la formule de quadrature (2.60) pour calculer l'intégrale suivante:
$$\int_0^1 \int_0^1 \sin(2\pi x)\sin(2\pi y)\mathrm{d}x\,\mathrm{d}y \ ,$$
Répétant le calcul pour un maillage de plus en plus serré (avec $\Delta x = \Delta y = h$), déterminez si cette formule d'intégration est d'ordre $O(h)$, ou $O(h^2)$, etc, en mesurant l'erreur par rapport à la solution exacte (facilement calculable ici puisque l'intégrale est séparable en x et y).

9. Intuitez les fonctions d'interpolation requises pour une interpolation trilinéaire (i.e., d'ordre 2) d'une fonction de trois variables spatiales. Commencez par une comparaison attentive des fonctions correspondantes en 1D et 2D.

Bibliographie:

Le calcul des formules de différences finies est traité dans tous les bouquins d'analyse numérique, incluant celui de Press *et al.* cité en bibliographie au chapitre 1. J'ai gardé une préférence pour la présentation du sujet faite à la §2.1 de

Lapidus, L., & Pinder, G.F. 1982, *Numerical solution of partial differential equations in science and engineering*, John Wiley & Sons.

L'interpolation et l'intégration sont des sujets également traités dans tous les ouvrages de méthodes numériques. Encore une fois le *Numerical Recipes* vaut la peine d'être consulté: chapitre 3 pour l'interpolation, et chapitre 4 pour l'intégration. Les chapitres 3 et 4 de l'ouvrage suivant sont également une bonne introduction:

Gerald, C.F. 1978, *Applied numerical analysis*, Addison-Wesley.

Sur l'interpolation, les matheux(ses) pourront consulter également le chapitre 2 de

Stoer, J., & Bulirsch, R. 1980, *Introduction to numerical analysis*, Springer.

Chapitre 3

Équations différentielles ordinaires

3.1 Repenser $F = ma$

Plus vous progresserez dans vos études de la physique, plus vous verrez ses grands principes (comme la conservation de l'énergie, de la quantité de mouvement, etc) sous la forme **d'équations différentielles**, et la capacité de solutionner ces équations deviendra essentielle. Ce chapitre vise à vous apprendre quelques techniques numériques pour la solution d'équations différentielles dites **ordinaires**, soit une équation impliquant des dérivées d'une variable dépendante par rapport à une seule variable indépendante.

Un exemple spécifique vaut bien une longue discussion. Prenons encore une fois la très justement célèbre Loi de Newton:

$$\mathbf{F} = m\mathbf{a} \ . \tag{3.1}$$

Ceci représente en fait trois équations, puisque la force et l'accélération sont des quantités vectorielles dont la définition requiert trois composantes distinctes. Vous n'aurez certainement pas oublié que l'accélération est la dérivée temporelle de la vitesse $\mathbf{v}$, et que cette dernière peut être considérée comme une dérivée temporelle du vecteur position $\mathbf{x}$. L'équation (3.1) peut donc être écrite sous les formes alternatives:

$$\frac{\mathrm{d}\mathbf{v}}{\mathrm{d}t} = \frac{\mathbf{F}}{m} \ , \tag{3.2}$$

$$\frac{\mathrm{d}^2\mathbf{x}}{\mathrm{d}t^2} = \frac{\mathbf{F}}{m} \ . \tag{3.3}$$

Ces deux expressions sont en tout point physiquement équivalentes à (3.1).

Appliquée à un mouvement uniformément accéléré par une force de grandeur constante et exercée dans la direction-x (disons), l'équation (3.3) devient

$$\frac{\mathrm{d}^2 x}{\mathrm{d}t^2} = \frac{F}{m} \ , \tag{3.4}$$

les composantes-y et z étant satisfaite trivialement (elles donnent $0 = 0$!). Deux intégrations successives de cette expression consuisent à la célèbre formule que vous avez mémorisé déjà dans votre tendre jeunesse:

$$x(t) = x_0 + v_0 t + \frac{1}{2} a t^2 \ , \tag{3.5}$$

où ici $a = F_x/m$. Les quantités x_0 et v_0 sont les deux constantes d'intégration, correspondant aux position et vitesse au temps $t = 0$. Je me permet de vous rappeler que la spécification de ces constantes d'intégration est un aspect essentiel de la solution du problème. L'EDO étant ici d'ordre deux (la plus haute dérivée est une dérivée seconde), deux conditions initiales doivent être spécifiées.

C'est la forme particulièrement simple —une constante— du membre de droite de l'équation (3.3) qui permet une intégration analytique. Souvent, pour des forces ayant des dépendances explicites sur x et/ou t, ceci n'est plus possible et on doit solutionner directement l'équation différentielle de manière numérique. C'est ce que nous ferons dans ce chapitre, dans le contexte d'un problème physique que vous connaissez bien, soit le mouvement du pendule. Mais pour commencer nous allons développer deux algorithmes numériques dans le contexte d'une équation différentielle ordinaire encore plus simple.

3.2 La méthode d'Euler

Considérons une fonction $f(t)$ obéissant à l'équation différentielle ordinaire (EDO) suivante:

$$\frac{\mathrm{d}f}{\mathrm{d}t} = f \ , \qquad f(0) = 1 \ , \qquad t \in [0, 2] \ . \tag{3.6}$$

"Intégrer" cette EDO consiste à calculer la fonction $f(t)$ satisfaisant à la fois à l'EDO même, ainsi qu'à la condition initiale spécifiée. Premier point important ici: un problème bien posé doit inclure non seulement l'EDO, mais des conditions initiales/limites (ici $f(0) = 1$) , et un domaine d'intégration (ici $t \in [0, 2]$). Dans le cas du problème défini par l'éq. (3.6), la solution analytique est donnée par la fonction exponentielle que vous connaissez bien:

$$f(t) = \exp(t) \ . \tag{3.7}$$

En fait, l'EDO (3.6) peut être considérée comme une *définition* de la fonction exponentielle. Mais voyons maintenant comment solutionner cette EDO numériquement.

3.2.1 Euler explicite

La méthode d'**Euler explicite** est basée directement sur le développement en série de Taylor, tronqué au second terme:

$$f(t_0 + h) = f(t_0) + h\left.\frac{\mathrm{d}f}{\mathrm{d}t}\right|_{t_0} \ ; \tag{3.8}$$

mais, selon notre EDO (3.6) on sait également que $\mathrm{d}f/\mathrm{d}t = f$, donc on peut reécrire ceci comme

$$f(t_0 + h) = f(t_0) + h \times f(t_0) \tag{3.9}$$

La stratégie consiste à utiliser cette expression pour "avancer" la solution dans le temps, à partir de la condition initiale, pour un pas de temps h choisi *a priori*. Le calcul numérique de la solution commence par la discrétisation de la variable indépendante t sur une maille couvrant l'intervalle désiré (ici $t \in [0, 2]$). Choisissons une maille équidistante:

$$t_{k+1} = t_k + h \ , \qquad k = 0, 1, 2, ...$$

où h mesure ici la grandeur (constante) du pas de temps, et ù t_0 correspond à la valeur de t à laquelle est spécifiée la condition initiale. On peut donc appliquer l'éq. (3.9) séquentiellement à chaque point de maille, en commençant par la condition initiale (connue!) fournie pour $t = 0$, correspondant au point de maille $k = 0$:

$$f_{k+1} = f_k + h \times f_k \ , \qquad k = 0, 1, 2, ... \tag{3.10}$$

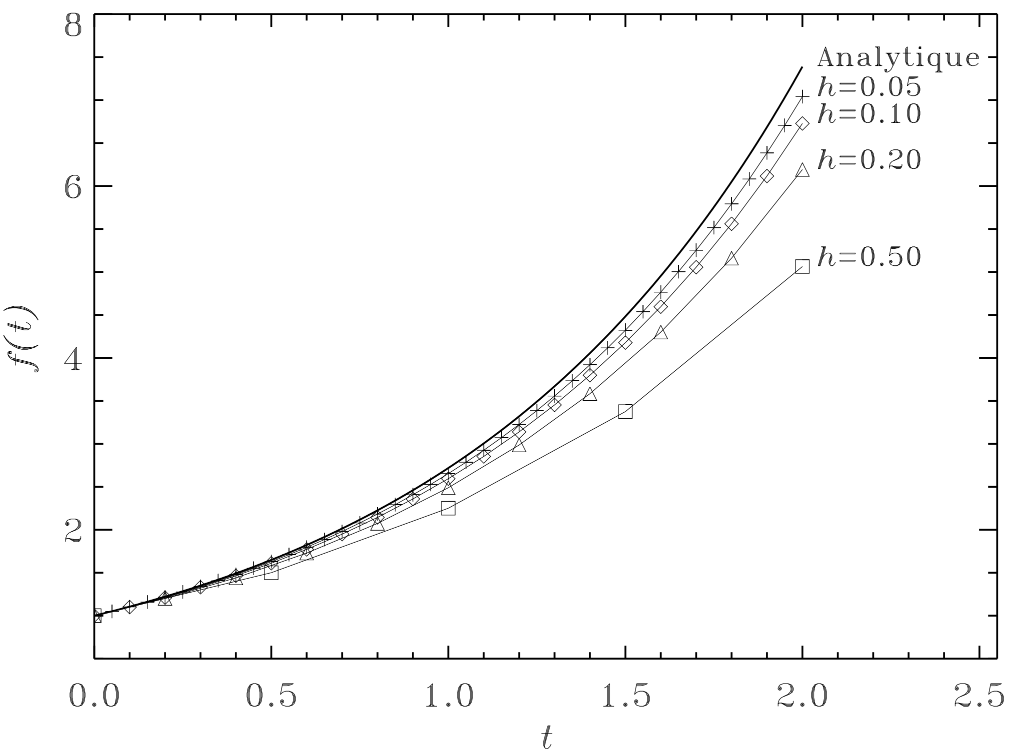


Figure 3.1: Solution numérique de l'EDO (3.6) à l'aide de la méthode d'Euler explicite, pour différentes tailles du pas de temps h, tel qu'indiqué. Le trait épais est la solution analytique donnée par l'éq. (3.7).

où on a utilisé la notation $f_k \equiv f(t_k)$. On aurait pu arriver directement à cet algorithme en appliquant la formule de différence finie d'ordre $O(h)$ (éq. (2.15)) au membre de gauche de l'éq. (3.6) et en évaluant le membre de droite à t_k, ce qui est tout à fait équivalent à la dérivation ci-dessus puisque la dite formule de différence finie résulte d'un développement en série de Taylor tronqué au second terme... ! En langage C, cet algorithme tient en quelques lignes:

```
h = 0.1 ;                 /* Choix du pas de temps */
t=0 ; fk=1. ;             /* Initialisations */
while (t < 2. ) {
   fkp1=fk+h * fk ;       /* eq. (3.10) */
   fk=fkp1 ;              /* On avance la solution d'un pas */
   t =t+h ;
}
```

Notez, avant la sortie de la boucle, l'assignation `fk=fkp1`, essentielle ici pour que la valeur nouvellement calculée f_{k+1} deviennent la valeur "connue" f_k à la prochaine itération de la boucle temporelle; également, l'incrémentation de la variable temporelle t, de manière à ce que la boucle se termine une fois $t = 2$ atteint.

La Figure 3.1 montre le résultat de l'algorithme, pour 4 tailles de pas de temps, tel qu'indiqué. On note que plus le pas de temps est petit, plus la solution numérique s'approche de la solution analytique (trait épais). Ceci est tout à fait naturel, puisque la précision d'un développement en série de Taylor tronqué, sur lequel est basé notre algorithme, s'améliore à mesure que la taille du pas diminue (retournez voir la Figure 2.5, courbe en $*$ pour la formule $O(h)$). On remarquera aussi que, quel que soit la grandeur du pas, à tout t la solution numérique se retrouve toujours systématiquement inférieure à la solution analytique. Ceci vient du fait que la fonction $\exp(t)$ est concave, dans le sens que sa dérivée seconde est positive; autrement dit, sa dérivée première augmente avec t. Ceci implique que l'utilisation de f_k pour estimer $\mathrm{d}f/\mathrm{d}t$ dans le passage de l'éq. (3.8) à l'éq. (3.12) sous-estimera toujours la valeur "moyenne" de la dérivée dans l'intervalle $t \in [t_k, t_{k+1}]$, donc il est inévitable que f_{k+1} se retrouve sous la valeur "réelle" à t_k. Vous aurez déjà anticipé (j'espère...) que si la fonction intégrée avait été convexe plutôt que concave, alors la solution numérique se serait systématiquement retrouvée au dessus de la solution exacte. Ce problème s'amenuise évidemment à mesure que h diminue, mais pensez-y un peu et vous devriez réaliser qu'il persistera même dans la limite $h \to 0$. Voyons comment contourner cette difficulté.

3.2.2 Euler explicite avec extrapolation linéaire

La piètre performance de la méthode d'Euler explicite s'explique principalement par le fait que les membres de gauche et droite de l'équation (3.6) sont traités différemment par rapport à la variable temporelle. La différence finie avant (2.16) peut se réinterpréter comme une différence finie centrée sur un temps correspondant à un demi pas, $t_{k+1/2}$; dans un tel cas il deviendrait naturel d'évaluer le membre de droite également à $t_{k+1/2}$, par exemple comme sa moyenne entre t_k et t_{k+1}; on pourrait alors écrire:

$$\frac{f_{k+1} - f_k}{h} = \frac{f_{k+1} + f_k}{2} \; , \qquad k = 0, 1, 2, ... \tag{3.11}$$

Mais voilà, nous ne connaissons pas encore la valeur de f_{k+1} apparaissant aux membres de droite, puisque nous sommes à ce stade encore en train d'essayer de calculer le pas $k+1$! Cependant, il est possible d'approximer ces quantités en effectuant une **extrapolation linéaire** à partir des valeurs au pas k, basée sur devinez quoi, notre très cher développement en série de Taylor, tronqué après le second terme:

$$f_{k+1} = f_k + h\frac{\mathrm{d}f}{\mathrm{d}t} = f_k + h \times f_k \; , \tag{3.12}$$

où la seconde égalité résulte encore une fois de l'utilisation de l'éq. (3.6). La substitution de cette expression pour le f_{k+1} apparaissant au membe de droite de l'éq. (3.11), (mais attention, pas pour celui du membre de gauche!), produit alors:

$$\frac{f_{k+1} - f_k}{h} = \frac{1}{2}(f_k + h \times f_k + f_k) \; , \qquad k = 0, 1, 2, ... \tag{3.13}$$

Ceci conduit directement à notre algorithme d'Euler "amélioré":

$$f_{k+1} = f_k + h \times \left(f_k + \frac{h}{2} f_k \right) \; , \qquad k = 0, 1, 2, ... \tag{3.14}$$

En terme de code C, ça aurait maintenant l'air de:

```
h = 0.1 ;                          /* Choix du pas de temps */
t=0 ; fk=1. ;                      /* Initialisations */
while (t < 2. ) {
   fkp1=fk+h * (fk + h*fk/2.) ;    /* eq. (3.14) */
   fk=fkp1 ;
   t =t+h ;
}
```

Comparez bien ce bout de code à celui de la méthode d'Euler standard. Tout ce qui change est l'évaluation de f_{k+1} (≡`fkp1`), mais la structure générale demeure la même. La Figure

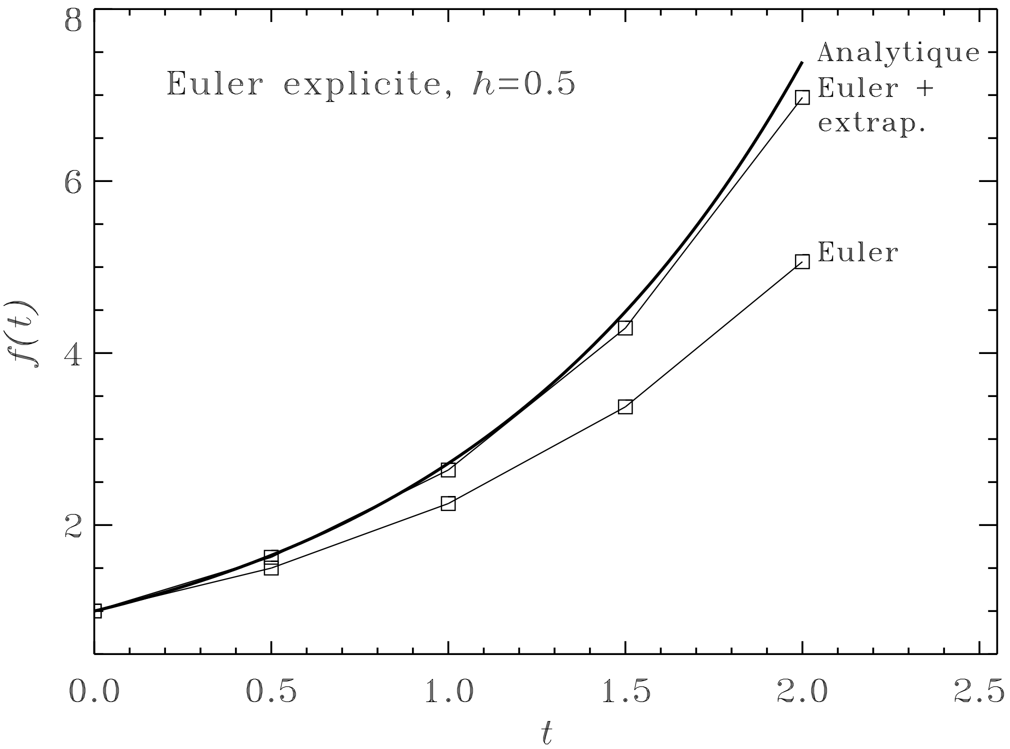


Figure 3.2: Semblable à la Figure 3.1, sauf que cette le pas de temps est fixé à $h = 0.5$, et les deux solutions numériques portées en graphiques ont été calculée à l'aide des méthodes d'Euler avec ou sans extrapolation linéaire du membre de droite, tel qu'indiqué.

3.2 montre une comparaison entre deux solutions avec le même pas de temps $h = 0.5$, l'un utilisant la méthode d'Euler explicite et la seconde utilisant la version avec extrapolation linéaire introduite ci-dessus. Comparant ceci à la Fig. 3.1, on constate que l'évaluation du membre de droite de l'EDO en terme d'une valeur moyenne à mi-pas produit une solution ayant une précision comparable à celle caractérisant la méthode d'Euler explicite opérant à un pas de temps 5 fois plus petit, et ce *même si dans les deux cas la dérivée est approximée par une formule de différences finies d'ordre $O(h)$*. Il y a un message important ici: L'ordre de l'erreur de discrétisation contrôle le taux auquel diminue l'erreur dans la limite $h \to 0$; mais le choix d'un algorithme plutôt qu'un autre a également un fort impact sur le niveau d'erreur *à une taille de maille donnée.*

Les expressions algorithmiques (3.10) et (3.14) sont spécifiques à la solution numérique de l'EDO (3.6), où le membre de droite est égal à la fonction f dont le membre de gauche représente la dérivée temporelle. Il ne faut pas appliquer aveuglément ces algorithmes à d'autres EDOs ! Par exemple, pour l'EDO

$$\frac{\mathrm{d}f}{\mathrm{d}t} = at^2 \ , \tag{3.15}$$

avec a une constante connue, les équations (3.10)—(3.14) deviendrait respectivement:

$$f_{k+1} = f_k + h \times a\, t_k^2 \ , \qquad k = 0, 1, 2, ... \qquad \text{[Euler explicite]} \tag{3.16}$$

$$f_{k+1} = f_k + \frac{ah}{2} \times (t_{k+1}^2 + t_k^2) \ , \qquad k = 0, 1, 2, ... \qquad \text{[Euler avec extrapolation]} \tag{3.17}$$

Notons que dans ce dernier cas, l'extrapolation du membre de droite est triviale puisque t est la variable dépendante, et donc t_{k+1} est directement calculable, sans extrapolation. Assurez vous de bien comprendre comment arriver aux deux expressions ci-dessus à partir de l'éq. (3.15), avant de passer à la suite.

3.2.3 Euler pour équations nonlinéaires

Les équation (3.6) et (3.15) sont **linéaires**, dans le sens qu'à partir de toute solution ($f_1(t)$, disons), il est possible de construire une infinité de nouvelles solutions de la forme $f_2(t) = a \times f_1$, où a est une constante; et toute combinaison linéaire de solutions satisfaisant toutes à la condition initiale sera également une solution valide. Mathématiquement, ceci définit une sous-classe très spécifique d'EDO, mais la méthode d'Euler est d'applicabilité beaucoup plus générale. Écrivons une EDO nonlinéaire prototypique sous la forme:

$$\frac{\mathrm{d}f}{\mathrm{d}t} = g(t, f) \ , \tag{3.18}$$

où le membre de droite g est en général une fonction nonlinéaire de f dont la forme analytique est connue (e.g., $-f^2$), et peut de surcroit inclure une dépendance temporelle explicite (e.g., $g(t, f) = -f^2 \sin(\omega t)$). Évaluant le membre de gauche à l'aide de la différence finie avant (2.15), et le membre de droite au pas de temps t_k, la méthode d'Euler explicite prend maintenant la forme:

$$f_{k+1} = f_k + h \times g(t_k, f_k) \ , \qquad k = 0, 1, 2, ... \tag{3.19}$$

Du point de vue d'une solution numérique par Euler explicite, il n'y a vraiment aucune différence entre une EDO linéaire ou nonlinéaire (ce qui est loin d'être le cas si on cherchait plutôt une solution analytique!). On peut aussi de nouveau évaluer le membre de droite à mi-pas:

$$\frac{f_{k+1} - f_k}{h} = \frac{1}{2}\left(g(t_k, f_k) + g(t_k + h, f_k + h\left.\frac{\mathrm{d}f}{\mathrm{d}t}\right|_{t_k})\right) \ , \qquad k = 0, 1, 2, ... \tag{3.20}$$

ce qui, via l'utilisation de l'éq. (3.18), conduit à l'algorithme:

$$f_{k+1} = f_k + \frac{h}{2}\left[g(t_k, f_k) + g(t_{k+1}, f_k + h\, g(t_k, f_k))\right] + O(h^2) \ . \tag{3.21}$$

Notez la ressemblance avec à la méthode d'Euler explicite avec extrapolation linéaire, sauf que cette fois c'est $g(t, f)$ qui est évalué à (t_k, f_k) et à (t_{k+1}, f_{k+1}), via une extrapolation linéaire de f. C'est la **méthode de Heun**. On peut montrer (mais nous ne le ferons pas ici) que cette variation sur le thème de base suffit à réduire l'erreur de discrétisation à $O(h^2)$.

Avant de continuer, assurez vous de bien comprendre comment les éqs. (3.14) et (3.17) représentent chacune une application spécifique de l'éq. (3.21) à leur EDOs respectives.

3.3 Le pendule linéaire

Maintenant que nous savons intégrer des EDOs, passons à un problème physique plus intéressant, soit, ahem, le pendule...

La "Loi du pendule", soit le fait que la période d'oscillation d'un pendule n'étant pas déplacé trop loin de la verticale ne dépende que de la longueur de sa corde ou tige est généralement attribuée à Galilée. La légende veut que l'illustre personnage ait euréké sur la chose au début des années 1580, alors qu'il n'était encore qu'un vulgaire petit étudiant sous-gradué en médecine à l'Université de Pise, en observant l'oscillation d'une lampe suspendue à l'intérieur de la nef de la cathédrale de Pise (le sermon devait être particulièrement gazant...). Les experts ayant décortiqué les écrits et la correspondance de Galilée placent plutôt la découverte une vingtaine d'années plus tard. Ce qui semble certain, c'est que Galilée avait rapidement réalisé

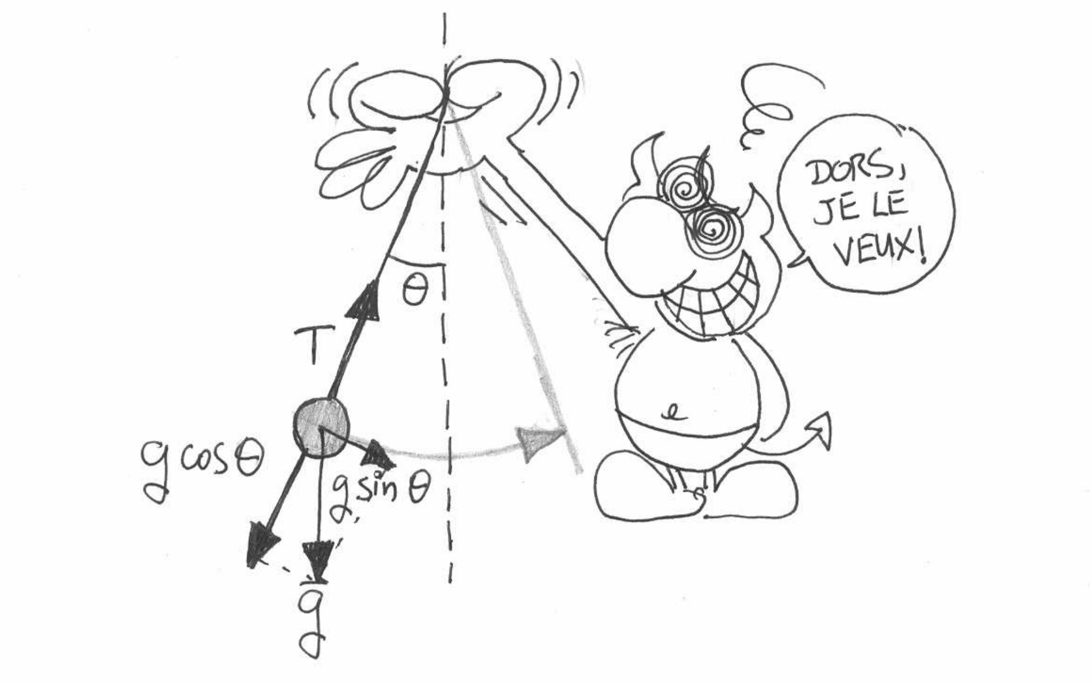


Figure 3.3: Géométrie et décomposition de la force gravitationnelle pour le problème du pendule classique. La gravité pointe verticalement vers le bas, une direction qu'il s'avère pratique de faire coincider avec $\theta = 0$ dans le système de coordonnées polaires 2D (r, θ) utilisé pour mesurer la position de la masse m, l'origine du système coincidant avec le point d'attache du pendule.

que le pendule pouvait servir d'étalon de mesure du temps (avant 1603 Galilée utilisait son pouls comme mesure du temps dans ses expériences sur le mouvement des corps!). Il fallut cependant attendre un demi-siècle et Christian Huygens (1629–1695) pour le perfectionnement de la première véritable horloge basée sur le battement d'un pendule.

3.3.1 Solution analytique

On vous a déjà passablement assomé avec le pendule dans vos cours de physique au CEGEP, donc nous pourrons passer rapidement sur l'idée générale (voir Figure 3.3). Une masse m est suspendue au bout d'un fil ou tige de masse nulle et de longueur L. Initialement suspendue verticalement, la masse est déplacée latéralement et relâchée, ce qui résulte en une oscillation dans le plan défini par la verticale et la direction de déplacement initial du pendule. L'application de la Loi de Newton est facilitée si l'on choisit de travailler en coordonnées polaires (r, θ), avec l'origine au point de suspension du pendule. Le déplacement est alors uniquement dans la direction $\hat{\mathbf{e}}_\theta$, et la composante-r de $\mathbf{F} = m\mathbf{a}$ se réduit à un équilibre instantané entre la tension et la composante de la force gravitationnelle projetée dans la direction $\hat{\mathbf{e}}_r$. La composante-θ conduit à quelque chose de plus intéressant. L'accélération dans la direction $\hat{\mathbf{e}}_\theta$ est donnée par

$$a_\theta = L\frac{\mathrm{d}v_\theta}{\mathrm{d}t} = L\frac{\partial^2\theta}{\partial t^2} \ , \tag{3.22}$$

Comme la composante de la force gravitationnelle projetée dans la direction θ est donnée par $mg\sin\theta$ (voir Figure 3.3), la composante θ de $F = ma$ conduit à:

$$\frac{\mathrm{d}^2\theta}{\mathrm{d}t^2} = -\frac{g}{L}\sin\theta \ . \tag{3.23}$$

Dans la limite où le déplacement angulaire du pendule (mesuré en radians) est très petit, on peut approximer les sinus et cosinus comme suit:

$$\sin\theta \simeq \theta \ , \qquad \cos\theta \simeq 1 \ . \tag{3.24}$$

L'équation (3.23) se réduit alors à:

$$\frac{\mathrm{d}^2\theta}{\mathrm{d}t^2} = -\omega^2\theta \ , \tag{3.25}$$

où on a défini

$$\omega = \sqrt{\frac{g}{L}} \ . \tag{3.26}$$

L'équation (3.25) accepte des solutions de la forme générale

$$\theta(t) = A\sin(\omega t) + B\cos(\omega t) \ , \tag{3.27}$$

où les constantes d'intégration A et B sont déterminées par les conditions initiales du problème, et ω représente la fréquence angulaire d'oscillation (unités: rad s^{-1}) par rapport à la position d'équilibre $\theta = 0$. Remarquez qu'on a encore ici deux constantes d'intégration, comme il se doit puisque l'éq. (3.25) est d'ordre deux. Pour un pendule déplacé à un angle Θ_0 de la verticale et relâché à $t = 0$ sans lui donner de vitesse initiale, on a $A = 0$ et $B = \Theta_0$. En terme de conditions initiales sur la variable $\theta(t)$, ceci revient à exiger:

$$\theta(0) = \Theta_0 \ , \qquad \frac{\mathrm{d}\theta}{\mathrm{d}t} = 0 \ . \tag{3.28}$$

Les équations (3.25) et (3.27) définissent le **mouvement harmonique simple**, dont les subtilités vous tiendront fort occupé(e)s l'hiver prochain en PHY-1620...

Dans le contexte du pendule, tout ceci devrait vous être déjà familier. Utilisons maintenant nos nouvelles habiletés numériques et voyons où ça nous mènera. Nous allons d'abord solutionner numériquement l'équation du pendule linéaire (3.25), pour laquelle nous connaissons la solution analytique exacte (3.27), à l'aide de la méthode d'Euler (§3.2). Nous allons par la suite (§3.4) nous concentrer sur la solution du pendule nonlinéaire, telle que décrite par l'équation (3.23). Ça, ce sera définitivement du nouveau!

3.3.2 Reformulation en deux équations d'ordre 1

Du point de vue purement numérique, face à une EDO d'ordre élevé (ici d'ordre 2 pour l'éq. (3.25)), il est préférable d'introduire des variables dépendantes additionnelles permettant de transformer l'EDO en un système de deux EDOs couplées, chacune d'ordre un. Dans le cas de l'éq. (3.25), il suffit de définir:

$$\frac{\mathrm{d}\theta}{\mathrm{d}t} = v \ , \tag{3.29}$$

de telle sorte que l'éq. (3.25) devient:

$$\frac{\mathrm{d}v}{\mathrm{d}t} = -\omega^2\theta, \tag{3.30}$$

Le système de deux équations (3.29) et (3.30) est mathématiquement équivalent à l'éq. (3.25), mais il s'avère souvent plus facile d'en obtenir des solutions numériques précises[1]. Quelque soit la formulation mathématique utilisée, tout comme précédemment le problème du pendule nous offre un problème dit "aux valeurs initiales", c'est à dire qu'une condition initiale nous est donnée à $t = 0$, et doit être "avancée" dans le temps.

[1] Ceci peut paraitre bizarre, mais c'est en fait relié à l'application des conditions initiale à un niveau de précision identique à celui de la discrétisation de l'EDO même.

3.3.3 Euler explicite, bis

La première étape consiste encore une fois à discrétiser la variable indépendante t sur une maille, que nous choisirons ici équidistante:

$$t_{k+1} = t_k + h \ , \qquad k = 0, 1, 2, ...$$

où t_0 correspond à la valeur de t à laquelle est spécifiée la condition initiale. Utilisons maintenant la méthode d'Euler explicite, qui rappellez-vous consiste à (a) évaluer ensuite le membre de droite des éqs. (3.29)—(3.30) à t_k, et (b) utiliser l'éq. (2.15) pour évaluer les dérivées temporelles; les éqs. (3.29) et (3.30) deviennent ainsi

$$\frac{\theta_{k+1} - \theta_k}{h} = v_k + O(h) \ , \qquad k = 0, 1, 2, ... \tag{3.31}$$

$$\frac{v_{k+1} - v_k}{h} = -\omega^2 \theta_k + O(h) \ , \qquad k = 0, 1, 2, ... \tag{3.32}$$

ce qui conduit immédiatement à l'algorithme:

$$\theta_{k+1} = \theta_k + h v_k + O(h) \ , \qquad k = 0, 1, 2, ... \tag{3.33}$$

$$v_{k+1} = v_k - \omega^2 h \theta_k + O(h) \ , \qquad k = 0, 1, 2, ... \tag{3.34}$$

où (v_0, θ_0) sont donnés en condition initiale. Pour un pendule déplacé de son point d'équilibre à un angle Θ_0, et simplement relâché sans lui donner une vitesse initiale, comme auparavant, on a simplement:

$$[v_0, \theta_0] = [0, \Theta_0] \ . \tag{3.35}$$

Connaissant la condition initiale $[v_0, \theta_0]$, les équations discrétisées (3.33)—(3.34) permettent donc de calculer (v_1, θ_1), et ensuite (v_2, θ_2), et ainsi de suite séquentiellement jusqu'au temps final d'intégration désiré. La Figure 3.4 montre un exemple de code C complet qui effectue cette intégration du pendule. Le résultat est illustré à la Figure 3.5, pour trois choix de pas de temps h. Notez que la variable temps est exprimée en unités de la période d'oscillation $T = 2\pi/\omega$, et l'amplitude en unités du déplacement angulaire initial Θ_0. La partie A montre l'évolution temporelle des solutions numériques, utilisant différents pas de temps tous relativement grands pour raisons didactiques, tandis que la partie B illustre la variation de l'erreur (ε), définie ici comme la différence entre la solution numérique et la solution exacte $\cos(\omega t)$:

$$\varepsilon(t_k) = \theta_k - \cos(\omega t) \ . \tag{3.36}$$

Remarquons que:

1. Plus h est petit, plus la solution approche de la solution exacte; on s'y attendait mais c'est tout de même rassurant de constater que c'est bien ce qui se passe.

2. De manière générale, l'erreur croit avec le temps; ceci est une conséquence de la nature séquentielle de l'algorithme d'Euler; toute erreur effectuée au pas k est "réinjectée" dans la solution au pas $k + 1$, et est donc cumulative.

3. L'erreur de discrétisation conduit à une déviation au niveau de l'amplitude de la variation sinusoidale, mais aussi de sa période.

```
#include <stdio.h>
#define NPAS 1000                    /* Nombre de pas de temps */
/* Ce programme integre les deux EDO couplees definisant            */
/* le probleme du pendule lineaire par la methode d'Euler explicite  */
int main(void)
{
/* Declarations ---------------------------------------------------- */
  int k=0 ;                          /* Compteur d'iterations */
  float h=0.01 ;                     /* Taille du pas de temps */
  float omega2 ;                     /* Frquence linaire (au carr)  */
  float t[NPAS] ;                    /* Le temps */
  float theta[NPAS], v[NPAS] ;       /* Les deux variables dpendantes */
/* Executable ------------------------------------------------------ */
  omega2=1. ;
  theta[0] = 1. ; v[0]=0. ;          /* Condition initiale */
  t[0]=0. ;
  for (k=0 ; k < NPAS-1 ; k++ )      /* Boucle temporelle */
    {
     theta[k+1]=theta[k]+h * ( v[k] ) ;              /* eq. (3.33) */
     v[k+1]    =v[k]    -h * omega2 * ( theta[k] ) ;  /* eq. (3.34) */
     t[k+1]    =t[k]    +h ;
     printf ("%f  %f  %f\n", t[k+1], theta[k+1], v[k+1]) ;
    }
}
```

Figure 3.4: Code C pour la solution du problème du pendule par la méthode d'Euler. Parmi les nouveautés dans ce code, notons la définition des tableaux unidimensionnels `theta`, `v` et `t`, qui à la fin de la boucle temporelle contiennent les valeurs des variables θ et v à un ensemble de `NPAS` ($= 10^3$) valeurs discrètes de la variable temps t. La dimension de ces tableaux doit être définie au moment de la compilation, ce qui est fait ici via l'instruction `#define NPAS 1000` placée en en-tête au programme même.

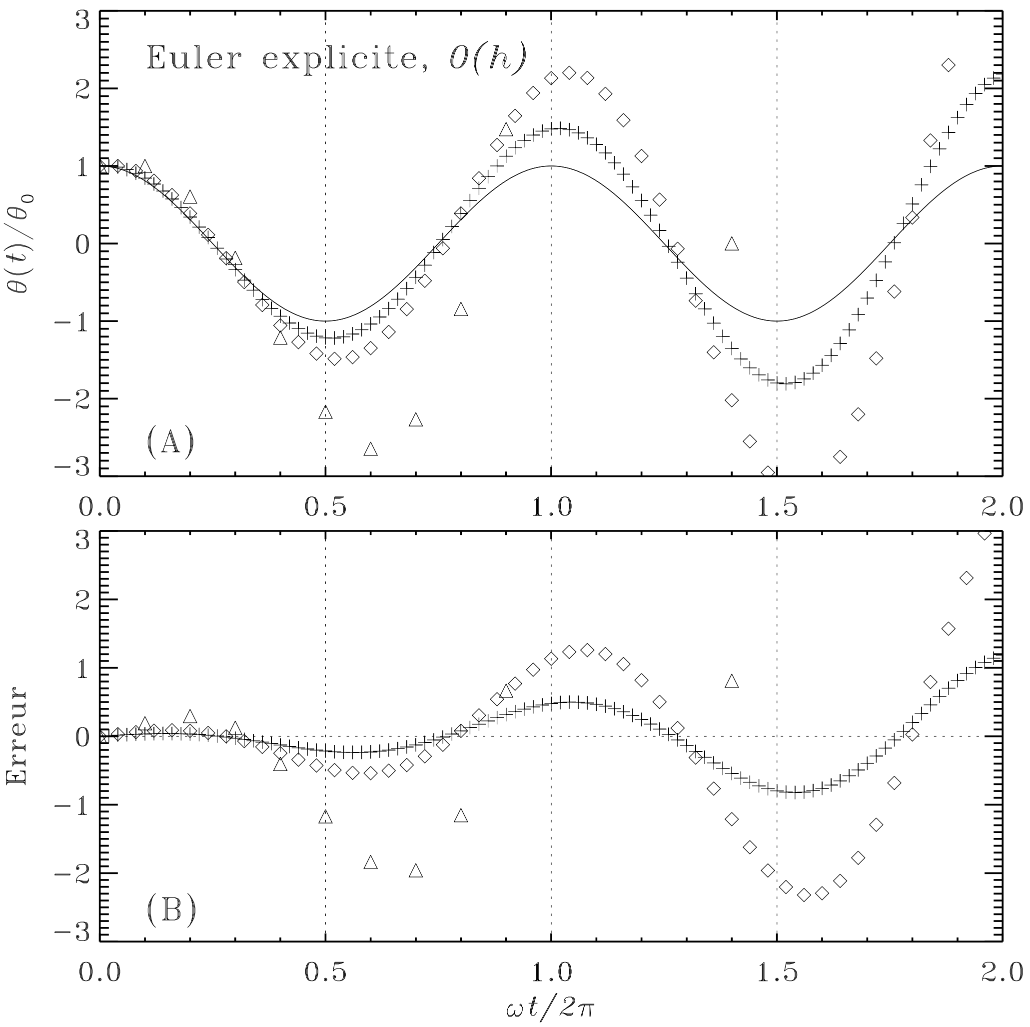


Figure 3.5: Solutions numériques de l'équation du pendule linéaire par la méthode d'Euler explicite, pour différentes valeurs du pas de temps h: ($\omega h = \pi/12.5 \to \triangle$, $\omega h = \pi/25 \to \diamond$, $\omega h = \pi/50 \to +$). Le temps est exprimé en unités de l'inverse de la fréquence d'oscillation ω, donc la période d'oscillation est égale ici à 2π dans ces unités. Les solutions utilisent la discrétisation définie par les éqs. (3.31)—(3.32), soit l'utilisation d'une différence finie avant d'ordre $O(h)$ pour évaluer la dérivée à t_k. La partie (A) compare ces diverses solutions à la solution exacte (trait plein), avec l'amplitude exprimée en unités de l'amplitude Θ_0 du déplacement angulaire initial. La partie B montre les erreurs des solutions numériques par rapport à cette solution exacte.

3.3.4 Euler explicite avec extrapolation linéaire, bis

La précision de la méthode d'Euler explicite peut être encore une fois grandement améliorée en évaluant les membres de droite des équations (3.29)—(3.30) à la mi-pas, soit comme leur valeur moyenne entre entre t_k et à t_{k+1}:

$$\frac{\theta_{k+1}-\theta_k}{h}=\frac{1}{2}(v_{k+1}+v_k)+O(h)\ , \qquad k=0,1,2,... \tag{3.37}$$

$$\frac{v_{k+1}-v_k}{h}=-\frac{\omega^2}{2}(\theta_{k+1}+\theta_k)+O(h)\ , \qquad k=0,1,2,... \tag{3.38}$$

On approxime de nouveau les θ_{k+1} et v_{k+1} aux membres de droite à partir des valeurs au pas k par extrapolation linéaire basée sur un développement en série de Taylor, tronqué après le second terme:

$$\theta_{k+1}=\theta_k+h\frac{\mathrm{d}\theta}{\mathrm{d}t}=\theta_k+hv_k\ , \tag{3.39}$$

$$v_{k+1}=v_k+h\frac{\mathrm{d}v}{\mathrm{d}t}=v_k-h\omega^2\theta_k\ , \tag{3.40}$$

où les secondes égalités résultent de l'utilisation des éqs. (3.29)—(3.30) pour évaluer les deux dérivées temporelles associées à l'extrapolation. L'algorithme devient alors:

$$\theta_{k+1}=\theta_k+\frac{h}{2}\left(v_k-h\omega^2\theta_k+v_k\right)=\theta_k+hv_k-\frac{\omega^2h^2}{2}\theta_k+O(h)\ , \qquad k=0,1,2,... \tag{3.41}$$

$$v_{k+1}=v_k-\frac{\omega^2h}{2}\left(\theta_k+hv_k+\theta_k\right)=v_k-\omega^2\left(h\theta_k+\frac{h^2}{2}v_k\right)+O(h)\ , \qquad k=0,1,2,... \tag{3.42}$$

La seule modification à notre code C pour la méthode d'Euler (Figure 3.4) consiste à remplacer les deux lignes de code faisant le calcul de `theta[k+1]` et `v[k+1]` à l'intérieur de la boucle temporelle:

```
theta[k+1]=theta[k]+h * ( v[k] - omega2*h*theta[k]/2.) ; /* eq. (3.41) */
v[k+1]    =v[k]    -h * omega2 * ( theta[k]+h*v[k]/2.) ; /* eq. (3.42) */
```

Il est important de réaliser que l'erreur associée à l'évaluation des dérivées temporelles aux membres de gauche par différence finies est toujours $O(h)$. Cependant, à un maillage donné cet algorithme s'avère à produire des solutions beaucoup plus précises que l'algorithme d'Euler original, comme on peut le constater en comparant la Figure 3.6 ci-dessous à la Fig. 3.5 ci-dessus. Notons déjà les échelles verticales très différentes sur les parties B! De tout évidence, notre réevaluation des membres de droite à la mi-pas a produit encore une fois une grande amélioration dand la précision des solutions, sur une maille donnée.

3.4 Le pendule nonlinéaire

Revenons à notre bon vieux pendule. Si nous ne faisons pas l'approximation du petit déplacement par rapport à la position d'équilibre ($\sin\theta\simeq\theta$), l'équation (3.23) demeure nonlinéaire en θ, et la séparation en deux équations d'ordre 1 conduit maintenant à:

$$\frac{\mathrm{d}\theta}{\mathrm{d}t}=v\ , \tag{3.43}$$

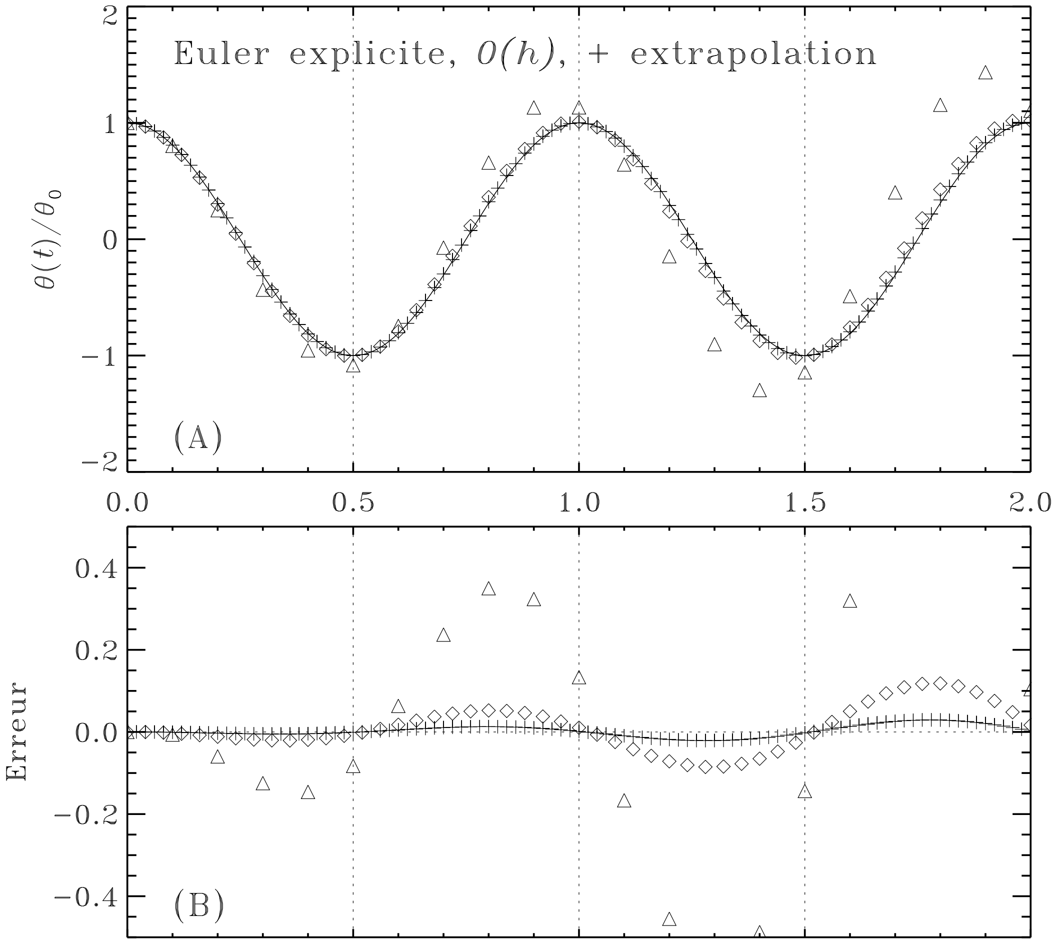


Figure 3.6: Identique à la Figure 3.5, sauf qu'ici le membre de droite des équations est évalué comme la moyenne aux pas k et $k+1$, par extrapolation linéaire à partir du pas k (éqs. (3.41)) et (3.42)). Notez le changement des échelles verticales, en particulier sur la partie B, par rapport à la Fig. 3.5.

$$\frac{\mathrm{d}v}{\mathrm{d}t} = -\omega^2 \sin\theta \ . \tag{3.44}$$

La méthode D'Euler explicite s'applique facilement à cette situation, produisant l'algorithme

$$\theta_{k+1} = \theta_k + h\frac{\mathrm{d}\theta}{\mathrm{d}t} = \theta_k + hv_k \ , \tag{3.45}$$

$$v_{k+1} = v_k + h\frac{\mathrm{d}v}{\mathrm{d}t} = v_k - h\omega^2 \sin\theta_k \ . \tag{3.46}$$

La première de ces expression est identique à son équivalent pour le pendule linéaire (éq. (3.39)), mais la seconde est maintenant nonlinéaire au membre de droite; cependant, dans le contexte de la méthode d'Euler explicite θ_k est connu, et calculer $\sin\theta_k$ ne présente donc aucune difficulté particulière. S'en tenant à la méthode d'Euler explicite pour l'instant, on n'a qu'à changer la ligne de code:

```
v[k+1]    =v[k]    -h * omega2 * sin( theta[k] ) ;
```

Calculons maintenant une série de solutions au problème du pendule nonlinéaire, obtenues à l'aide de l'algorithme ci-dessus. Les conditions initiales correspondent à de grands déplacements angulaire initiaux (voir Fig. 3.7), mais le pendule est encore une fois relâché sans lui donner de vitesse initiale. La Figure 3.8 illustre la variation temporelle du déplacement angulaire (mesuré en radians), pour un ensemble de déplacement initiaux allant jusqu'à $\pi/2$, soit une position initiale du pendule à l'horizontale. La première chose à remarquer est que la fréquence d'oscillation dépend maintenant de la condition initiale, contrairement au pendule linéaire où $\omega = \sqrt{g/L}$ quelle que soit l'amplitude initiale. Cette dépendance de la fréquence d'oscillation sur l'amplitude explique pourquoi il est si difficile de demeurer synchronisé avec un(e) petit(e) camarade, chacun sur sa balançoire; même une petite différence d'amplitude conduira déjà après quelques cycles d'oscillation à un déphasage perceptible.

Un autre truc qui n'est pas particulièrement évident sur la Figure 3.8, c'est que la forme de l'oscillation n'est plus exactement sinusoidale ici. La différence ne devient visuellement perceptible que pour des déplacements initiaux approchant π, comme l'illustre la Figure 3.9. Évidemment, si l'on veut considérer des déplacements initiaux dépassant $\pi/2$ nous sommes forcé de supposer que la masse m n'est pas reliée au point de pivot par une corde, mais par une tige rigide; sinon la chute initiale de m se fera verticalement vers le bas, et une corde plierait; ceci nous entrainerait dans un tout nouveau régime dynamique.

3.5 Au delà d'Euler

La méthode d'Euler explicite est un algorithme d'une grande simplicité, facilement généralisable à des systèmes d'EDOs, et même sans extrapolation linéaire du membre de droite est souvent suffisante dans bien des applications. Cependant son erreur de discrétisation varie comme $O(h)$, ce qui force l'utilisation d'une maille très serrée, qui se retrouve donc sensible à l'accumulation des erreurs d'arrondissement au niveau des opérations arithmétiques. Si vous avez suivi le raisonnement dans notre discussion des formules de différences finies au chapitre précédent, vous aurez peut-être déjà anticipé que des algorithmes plus précis que la méthode d'Euler peuvent être obtenus en conservant plus de termes dans le développement en série de Taylor de $f(t)$, et/ou en développant les membres de droite en série. Parmi la quasi-infinité d'algorithmes pouvant être produits de cette façon, celui que vous risquez de rencontrer le plus fréquemment est sans nul doute la méthode de Runge Kutta d'ordre 4 (souvent simplement appelée la méthode de Runge-Kutta tout court); l'idée ici est de subdiviser le pas de temps h en sous-pas à travers desquels la solution est avancée en réévaluant le membre de droite à chaque sous-pas. Appliqué à note EDO nonlinéaire prototypique (3.18), l'algorithme prend la forme suivante:

$$f_{k+1} = f_k + \frac{h}{6}(G_1 + 2G_2 + 2G_3 + G_4) + O(h^4) \ , \tag{3.47}$$

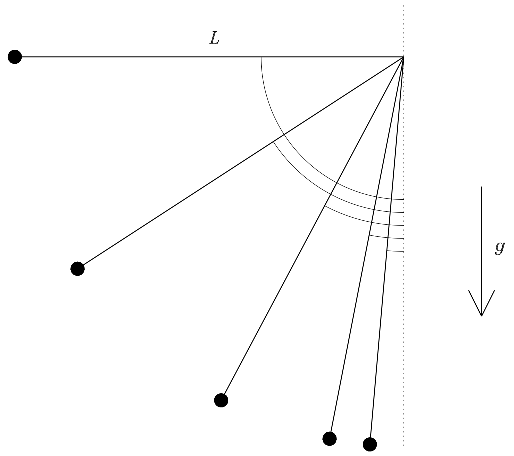


Figure 3.7: Le pendule nonlinéaire: le déplacement angulaire initial de la masse m est maintenant suffisamment grand pour que l'approximation $\sin\theta \simeq \theta$ ne tienne plus la route.

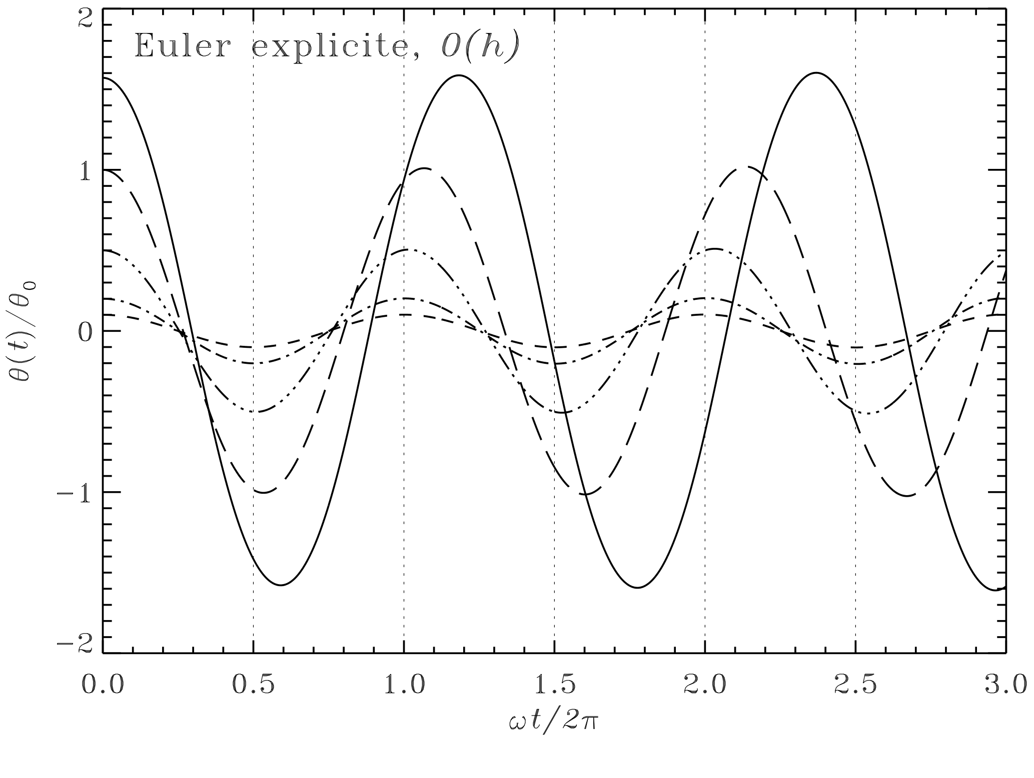


Figure 3.8: Solutions numériques au problème du pendule nonlinéaire, pour des amplitudes initiales $\Theta_0 = [0.1, 0.2, 0.5, 1.0, \pi/2]$ radians, tel qu'illustré sur la Figure 3.7. On remarquera que la période d'oscillation augmente avec l'amplitude du déplacement initial Θ_0. Les lignes verticales pointillées sont tracées aux demi-périodes du régime linéaire.

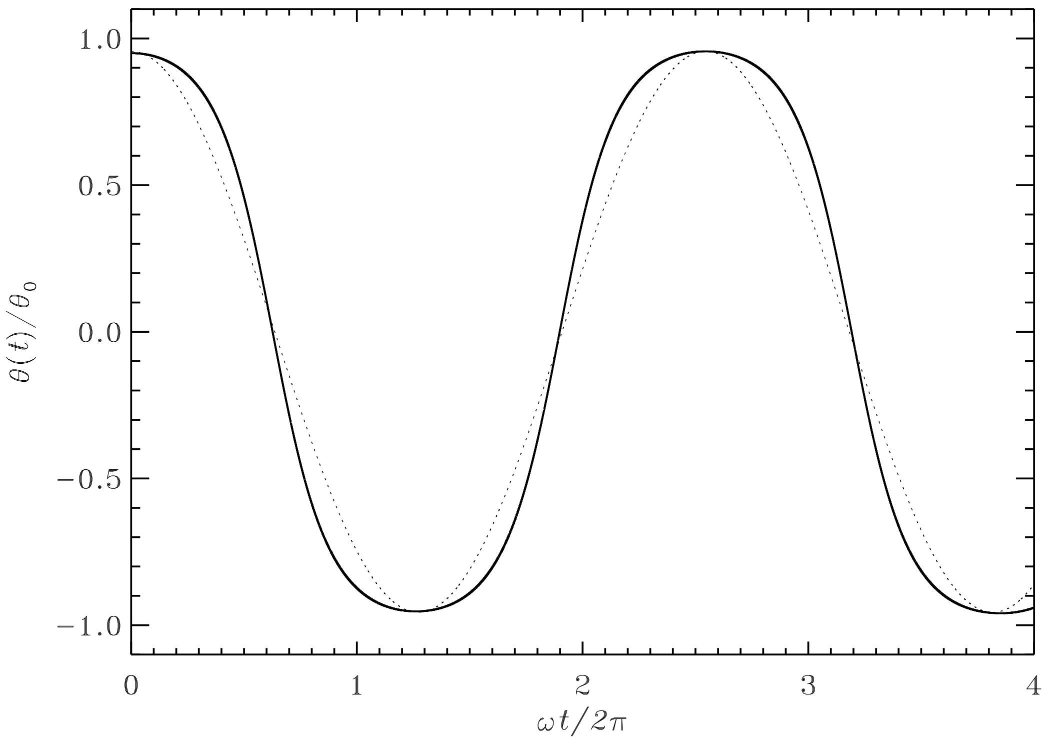


Figure 3.9: Solutions numériques au problème du pendule nonlinéaire, pour une amplitude initiale $\theta_0 = 0.95 \times \pi$, i.e., la masse m est relachée à $t = 0$ d'un point situé presqu'exactement au dessus du pivot du pendule. Le trait pointillé est une fonction sinusoidale ayant la même période et amplitude.

où

$$G_1 = g(t_k, f_k) \ , \tag{3.48}$$

$$G_2 = g\left(t_k + h/2, f_k + (h/2)G_1\right) \ , \tag{3.49}$$

$$G_3 = g\left(t_k + h/2, f_k + (h/2)G_2\right) \ , \tag{3.50}$$

$$G_4 = g\left(t_{k+1}, f_k + hG_3\right) \ . \tag{3.51}$$

Pour tous les détails concernant le design de ces algorithmes, voir le chapitre 2 de l'excellent ouvrage de Golub & Ortega cité en bibliographie à la fin du chapitre. Le bouquin de Wood, également listé ci-dessous, est aussi (à mon avis) une bonne référence, tout comme le chapitre 17 de Press *et al.* cité précédemment.

Nous n'avons fait ici qu'effleurer ce vaste sujet qu'est la solution numérique des équations différentielles ordinaires. En particulier, nous n'avons considéré ici que des EDO décrivant des problèmes dit **aux valeurs initiales**: on nous fournit une équation d'évolution (genre $\partial/\partial t = ...$) et une condition initiale à $t = t_0$, et le problème consiste à "avancer" cette condition initiale dans le temps. Un autre type de situation décrite par une EDO est le problème dit **aux valeurs limites**: on nous fournit une EDO décrivant un aspect d'un système défini sur un domaine (genre $x \in [a, b]$), et deux conditions limites à $x = a$ et $x = b$; le problème consiste alors à obtenir une solution à l'EDO dans le domaine, qui soit compatible avec ces conditions limites. Vous ferez connaissance avec ce genre de problème en mécanique quantique —entre autres— lorsque vous solutionnerez l'équation de Schrödinger dans un puit de potentiel.

Les références listées en fin de chapitre devraient suffire à ceux et celles désirant équiper encore plus leur "boite à outils" numérique de méthodes de solutions de ces diverses classes d'EDO. Pour nous ici, il est maintenant temps de passer à autre chose.

Exercices:

1. Une unité de temps "naturelle" pour le problème du pendule est l'inverse de la fréquence d'oscillation $\omega = g/L$; introduisez une nouvelle variable temps (t^*) telle que
$$t^* = t \times \omega \ ,$$
et montrez que les deux équations différentielles d'ordre 1 décrivant le mouvement pendulaire (les éqs. (3.29)–(3.30) prennent maintenant la forme:
$$\frac{\mathrm{d}\theta}{\mathrm{d}t^*} = v \ , \qquad \frac{\mathrm{d}v}{\mathrm{d}t^*} = -\theta \ .$$
2. Déterminez jusqu'à quel angle θ^* (en rad.) les approximations $\sin\theta \sim \theta$ et $\cos\theta \sim 1$ demeurent valides à 10^{-3}, 10^{-4} et 10^{-5} près.
3. Pourquoi cette séparation de la "vraie" équation du pendule (éq. (3.25)) en deux équations différentielles d'ordre 1? Utilisez une formule de différence finie pour la dérivée seconde pour produire un algorithme semblable à la méthode d'Euler explicite; codez le en C, et comparez sa précision (à pas de temps égal) à celle de l'algorithme standard.
4. Codez en C la méthode d'Euler explicite et sa version avec extrapolation linéaire, pour le problème du pendule linéaire. Intégrez sur une période complète d'oscillation, et examinez comment varie l'erreur associée à cette valeur finale à mesure que le pas de temps diminue; exprimez vos résultats sous la forme d'un équivalent de la Figure 2.5.
5. Codez en C la méthode de Runge-Kutta pour le problème du pendule linéaire. Produisez l'équivalent des Figures 3.5 et 3.6. Comment se compare cette méthode aux deux autres considérées dans ce chapitre, soit Euler explicite et Euler avec extrapolation linéaire?

Bibliographie:

Le chapitre 17 du *Numerical Recipes*, cité en bibliographie au chapitre 1, discute de manière plus approfondie les divers algorithmes décrits dans ce chapitre, en bien d'autres en plus. À un niveau mathématique plus poussé, les deux ouvrages suivant demeurent des classiques incontournables:

Gear, C.W., *Numerical Initial Value Problems in Ordinary Differential Equation*, Prentice-Hall (1971),

Stoer, J., & Bulirsch, R., *Introduction to Numerical Analysis*, 3e éd., Springer (2002).

Plus accessible, quoique que toujours dans le costaud, je recommanderais:

Golub, G.H., & Ortega, J.M., *Scientific Computing and Differential Equations*, Academic Press (1992),

Wood, W.L., *Practical time-stepping schemes*, Oxford (1990).

Chapitre 4

Nonlinéarité et chaos

4.1 Encore le pendule...

Vous en serez peut-être surpris, mais notre bon vieux pendule a encore en poche de quoi nous tenir occupé pendant tout un autre chapitre... Même avec le terme en $\sin\theta$ conservé comme tel, l'équation (3.23) demeure un modèle simplifié du mouvement pendulaire réel. Si vous avez déjà joué avec un pendule, vous aurez certainement remarqué que le mouvement ne perdure pas à l'infini. Ceci est du à la force de friction qui s'exerce entre la masse m (et le fil ou la tige) du pendule et l'air ambiant. Cette force de friction dissipe l'énergie cinétique de la masse m, et donc ralentit inexorablement le pendule. Nous commencerons par examiner ceci à la §4.3, non sans avoir tout d'abord introduit un concept physique qui sera fort utile, soit l'espace de phase (§4.2). Si l'on désire maintenir le mouvement pendulaire, nous devons clairement lui fournir de l'énergie, et ceci peut se faire par forçage mécanique du pendule, sujet de la §4.4. Nous verrons ensuite comment la combinaison du forçage et de l'amortissement peut conduire à des solutions stationnaires, dans le sens que le battement du pendule conserve une amplitude constante (§4.5). Cette situation à prime abord pas trop excitante nous mènera à la notion de bifurcation (§4.6), et deviendra encore plus surprenante quand nous examinerons le comportement de ces solutions plus "loin" dans l'espace des paramètres (§4.7), ce qui nous plongera littéralement en plein chaos.

4.2 L'espace de phase

Que ce soit en version linéaire ou nonlinéaire, nous avons l'habitude de penser au mouvement du pendule exclusivement en terme de la variation dans le temps de sa position angulaire θ (viz. Fig. 3.3). Cependant, notre traitement numérique de l'éq. (3.23) a commencé par l'introduction d'une seconde variable dynamique, $v \equiv \mathrm{d}\theta/\mathrm{d}t$. Cette variable est dite dynamique car son évolution est décrite par une équation différentielle impliquant une dérivée temporelle, au même titre que θ. D'un point de vue strictement mathématique, rien dans notre système de deux équations différentielles couplées d'ordre 1, soit les éqs. (3.43)—(3.44), ne confère à la variable θ un status particulier par rapport à la variable v.

Les variables θ et v peuvent être interprétées comme les coordonnées d'un "point" (θ, v) dans un espace bi-dimensionnel, et l'évolution dynamique du système —ici le pendule— est entièrement décrite par la **trajectoire** de ce points dans cet **espace de phase** bi-dimensionnel, à partir du point (θ_0, v_0) correspondant à la condition initiale.

Revenons à la forme linéaire du problème du pendule; on a vu que pour un pendule déplacé à une distance angulaire θ_0 ($\ll 1$) et relaché sans lui donner de vitesse initiale, l'évolution temporelle du déplacement angulaire est donnée par un mouvement harmonique simple:

$$\theta(t) = \theta_0 \cos(\omega t) \ , \tag{4.1}$$

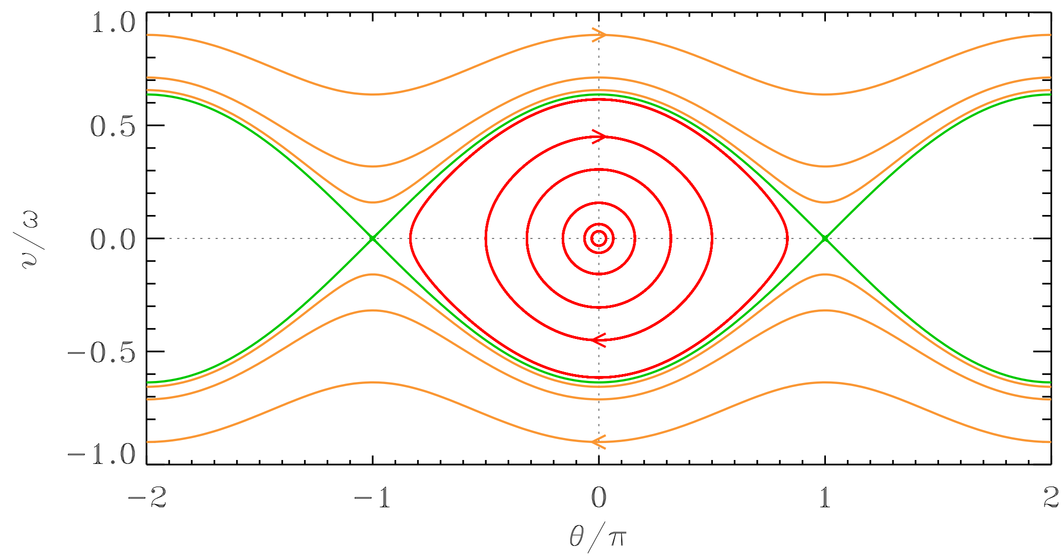


Figure 4.1: Espace de phase pour le pendule nonlinéaire (à tige rigide). Les trajectoires fermées (en rouge) correspondent à des mouvements oscillatoires autour de la position d'équilibre $(\theta, v) = (0, 0)$. Les courbes vertes délimitent la région des orbites fermées de celles des trajectoires ouvertes (orange), correspondant à une masse tournant perpétuellement dans le sens horaire ($v > 0$) ou antihoraire ($v < 0$). Les cinq plus petites orbites fermées correspondent aux solutions de la Figure 3.8.

avec $\omega^2 = g/L$; la variable v évolue donc dans le temps selon:

$$\frac{1}{\omega}v(t) = -\theta_0 \sin(\omega t) \ , \tag{4.2}$$

Les membres de droite des équations (4.1) et (4.2) sont les équations paramétriques d'un cercle de rayon θ_0 centré sur $(0, 0)$ dans un espace $[\theta, v/\omega]$. L'évolution temporelle du pendule est décrite par le déplacement à vitesse constante d'un point $(\theta, v/\omega)$ dans le sens horaire. Le point $(0, 0)$ correspond ici à une solution d'équilibre où le pendule est au repos et pend directement vers le bas.

Si le déplacement initial du pendule ne satisfait la condition $\theta_0 \ll 1$ nous permettant de remplacer $\sin\theta$ par θ, les trajectoires ne sont plus des cercles, mais l'évolution du pendule nonlinéaire peut toujours être décrite par une trajectoire fermée dans l'espace de phase. Ceci est illustré par les trajectoires en rouges sur la Figure 4.1. Partant du centre $(0, 0)$, les cinq premières correspondent aux cinq solutions de la Figure 3.8, et la sixième a comme condition intiale $(\theta_0, v_0) = (5\pi/6, 0)$. Notons que l'espace de phase du pendule (linéaire ou non) est périodique dans la direction-θ, la périodicité étant de 2π puisque $\sin(\theta \pm 2\pi) \equiv \sin(\theta)$. Seules les trajectoires centrées sur $\theta = 0$ sont illustrées ici. Les solutions linéaires ne sont valides que très près des points $(\theta, v) = (\pm 2n\pi, 0)$, avec $n = 0, 1, ...$

Le fait que ces trajectoires soient fermées est évidemment une conséquence directe de la périodicité du mouvement. Cette périodicité est ici une conséquence directe de la conservation de l'énergie. Le contenu énergétique de la masse m fixée au bout de notre pendule inclut deux contributions, soit son énergie cinétique et son énergie potentielle gravitationnelle. Si on choisit comme point zéro du potentiel gravitationnel la hauteur du pivot du pendule, on a:

$$E = \frac{1}{2}mv^2 - mgL\cos\theta \ , \tag{4.3}$$

En l'absence de toute friction avec l'air ambiant, cette quantité d'énergie, quelle qu'elle soit à $t = 0$, doit être conservée pour tout $t > 0$, et ce que le pendule opère en mode linéaire ou nonlinéaire. Le mouvement pendulaire peut donc être vu comme un échange cyclique entre l'énergie cinétique et potentielle gravitationnelle de la masse m.

La transposition du mouvement accéléré dans son espace de phase nous permet de pousser plus loin notre étude du pendule. Considérons un pendule orienté verticalement vers le haut ($\theta = \pi$) et au repos ($v = 0$). Pour une masse m montée au bout d'une tige rigide, c'est là une position d'équilibre, car la force de gravité est complètement équilibrée par la rigidité de la tige. L'énergie est ici $E = mgL$. Sujet maintenant à tout déplacement angulaire même des plus minuscules, l'équilibre sera rompu et le pendule se mettra à tourner, dans le sens du petit déplacement. Rendu en bas, à $\theta = 0$, l'éq. (4.3) nous garantit que la vitesse sera $v = \sqrt{2gL}$, et retombera à zéro une fois que le pendule aura complété un tour complet.

Que se passe-t-il maintenant si on donne une vitesse initiale $v_0 > 0$ au pendule dans sa position d'équilibre (instable) $\theta = \pi$? Comme auparavant le pendule accélérera durant la descente et décélérera durant la remontée, mais son tour complété et arrivé à $\theta = \pi$ sa vitesse sera redevenu égale à sa vitesse initiale... et donc ca repart pour un second tour, et ainsi de suite *ad aeternam*. On est ici dans une situation où la vitesse oscille entre des limites v_0 et $\sqrt{v_0^2 + 4gL}$, et θ augmente continuellement. Ceci corresponds aux trajectoires tracées en orange dans la moitié supérieure de l'espace de phase sur la Figure 4.1. Si la vitesse initiale est négative, le mouvement sera décrit par les trajectoires oranges dans la moitié inférieure de la Figure.

Les courbes vertes sur la Figure 4.1 définissent les **séparatrices** de l'espace de phase. Le point d'intersection de ces séparatrices avec la ligne $v = 0$ correspond au pendule en équilibre (instable) à la position $(\theta, v) = ((2n \pm 1)\pi, 0)$, $n = 0, 1, ...$, soit la masse m positionné verticalement exactement au dessus du pivot.

Dans un système déterministe comme notre pendule, deux trajectoires ne peuvent pas se croiser dans l'espace de phase; il n'est pas possible non plus à une trajectoire de se croiser soi-même; si c'était le cas, au point de croisement il y aurait un "choix" à faire, à savoir laquelle des deux branches possibles prendre pour la suite de l'évolution. Les points de croisement des séparatrices peuvent paraitre violer ce précepte, mais on verra au chapitre 4 que le temps requis pour atteindre un de ces points tend vers l'infini. Et attendre un temps infini, c'est très long, surtout vers la fin...

4.3 Le pendule amorti

Tout corps se déplaçant dans un fluide —et l'air compte comme un fluide— sera sujet (au minimum) à une **force de trainée** ($\mathbf{F}_D$) orientée dans la direction opposée à son vecteur vitesse. Pour des mouvements relativement lents, dans le sens que la trainée ne développe pas de turbulence, la grandeur de cette force de trainée se retrouve directement proportionnelle à la grandeur de la vitesse:

$$\mathbf{F}_D = -\alpha \mathbf{v} \ . \tag{4.4}$$

où α (> 0) est un coefficient de friction (unités: N s/m). Remarquez déjà que si cette force de trainée est la seule force en présence, la Loi de Newton

$$m\mathbf{a} = m\frac{\mathrm{d}\mathbf{v}}{\mathrm{d}t} = -\alpha \mathbf{v} \ . \tag{4.5}$$

nous indique immédiatement qu'un objet de masse m se déplaçant à une vitesse $\mathbf{v}_0$ verra sa vitesse décroitre exponentiellement selon:

$$\mathbf{v}(t) = \mathbf{v}_0 \exp\left(-\frac{\alpha}{m}t\right) \ . \tag{4.6}$$

Vous en apprendrez plus sur tout ça en ne manquant pas de vous inscrire à L'EXCELLENT cours PHY-3140, *Hydrodynamique*, enseigné à la session hiver par votre humble serviteur ici présent... Bon, bref, il ne s'agit plus que d'ajouter cette force au membre de droite de la composante-θ de notre équation du mouvement. Comme ici la vitesse $\mathbf{v} \equiv L\mathrm{d}\theta/\mathrm{d}t\,\hat{\mathbf{e}}_\theta$, on obtient

$$L\frac{\mathrm{d}^2\theta}{\mathrm{d}t^2} = -g\sin\theta - \frac{\alpha L}{m}\frac{\mathrm{d}\theta}{\mathrm{d}t} \ , \tag{4.7}$$

et notre séparation en deux équations différentielles ordinaires couplées d'ordre un nous donne maintenant:

$$\frac{\mathrm{d}\theta}{\mathrm{d}t} = v \ , \tag{4.8}$$

$$\frac{\mathrm{d}v}{\mathrm{d}t} = -\omega^2\sin\theta - \beta v \ , \tag{4.9}$$

où on a de nouveau défini $\omega^2 = g/L$, et également

$$\beta = \frac{\alpha}{m} \ . \tag{4.10}$$

La Figure 4.2 illustre deux solutions numériques obtenues à l'aide de la méthode de Heun (§3.5), dans l'espace de phase $[\theta, v]$ du modèle, pour deux valeurs distinctes de β mais la même solution initiale $(\theta_0, v_0) = (\pi - 0.1, 0)$. Avec $\beta = 0$, ces solutions suivraient une trajectoire ayant la forme d'une orbite fermée collant de près la séparatrice (trait noir). Ici, en présence d'amortissement, les trajectoires deviennent des spirales convergeant vers la position d'équilibre $(\theta, v) = (0, 0)$. Ceci correspond à une oscillation du pendule à partir de sa position de départ, avec une amplitude du battement décroissant exponentiellement avec le temps.

Ici, en fait, quelle que soit la condition initiale ou la valeur de β (> 0), les solutions convergent *toujours* vers la solution d'équilibre $(\theta, v) = (0, 0)$. On appelle un tel point un **attracteur** dans l'espace de phase puisqu'il "attire" à lui toutes les solutions, quelles que soient les conditions initiales. La vitesse de convergence des solutions vers l'attracteur dépendra évidemment de la valeur de β, et du contenu énergétique de la condition initiale, mais si l'on attend assez longtemps cette convergence aboutira toujours à $(0, 0)$; ou, plus précisément, à l'un ou l'autre des points situés à $(\theta, v) = (2n\pi, 0)$, $n = 0, 1, 2, ...$

4.4 Le pendule forcé

Passons maintenant à une autre variation sur le thème du pendule. Cette fois il s'agit d'un pendule dont le pivot est sujet à un forçage vertical harmonique; par exemple, le pivot peut être monté sur un mécanisme oscillant verticalement à une fréquence ω_0 ajustable, selon un mouvement harmonique:

$$z(t) = z_0\cos(\omega_0 t) \ . \tag{4.11}$$

À un tel mouvement est associé une accélération également harmonique, qui n'est que la dérivée seconde de cette expression:

$$g_z(t) = -g_0\cos(\omega_0 t) \ , \qquad g_0 = z_0\omega_0^2 \ . \tag{4.12}$$

L'idée générale est illustrée sur la Figure 4.3, pour un forçage passablement vigoureux. Pour un pendule à tige rigide, cette accélération additionnelle sera transmise à la masse m par l'intermédiaire de la tige, ce qui fait que la composante-θ de notre équation du mouvement aura maintenant l'air de:

$$L\frac{\mathrm{d}^2\theta}{\mathrm{d}t^2} = -(g + g_0\cos(\omega_0 t))\sin\theta \ . \tag{4.13}$$

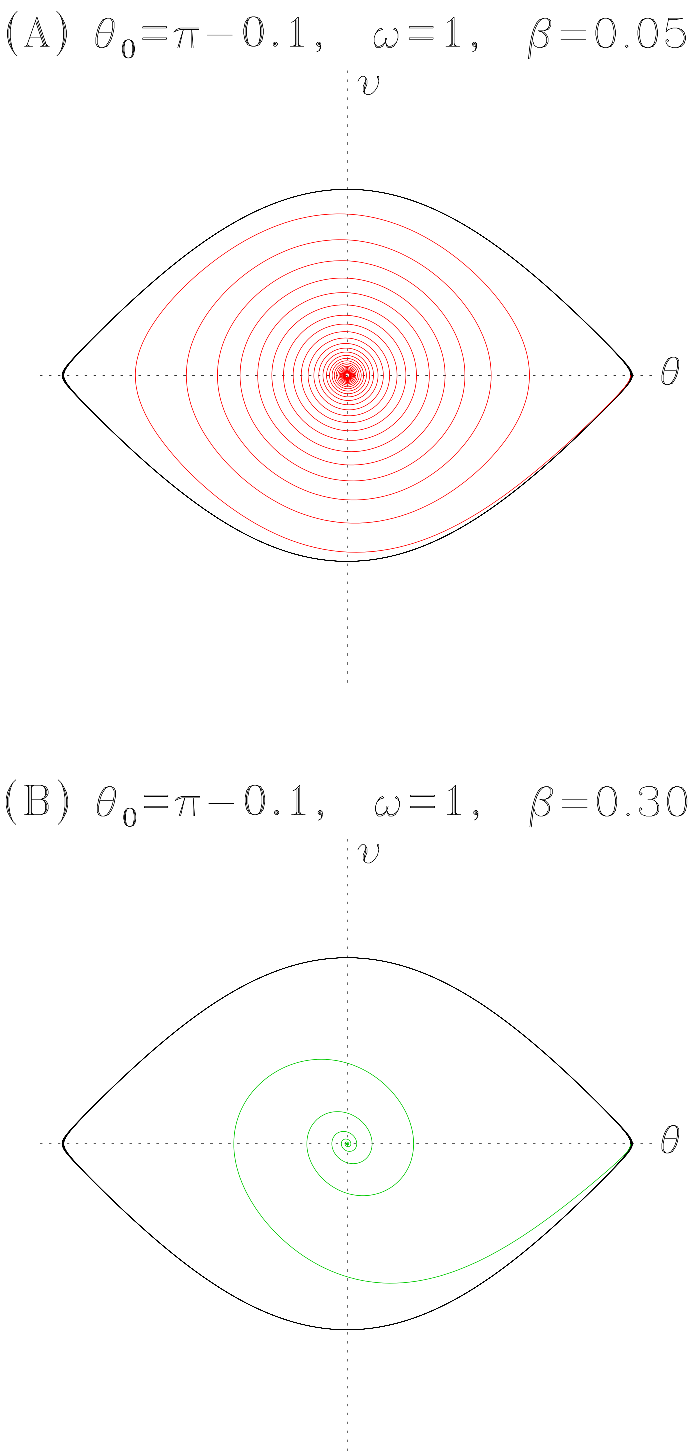


Figure 4.2: Trajectoires dans l'espace de phase pour le pendule nonlinéaire (à tige rigide) amorti. Les deux trajectoires ont la même condition initiale $(\theta_0, v_0) = (\pi - 0.1, 0)$, mais celui en (A) est faiblement amorti ($\beta = 0.05$), tandis que celui en (B) l'est plus fortement ($\beta = 0.3$). Dans les deux cas la trajectoire aboutit à $(\theta, v) = (0, 0)$.

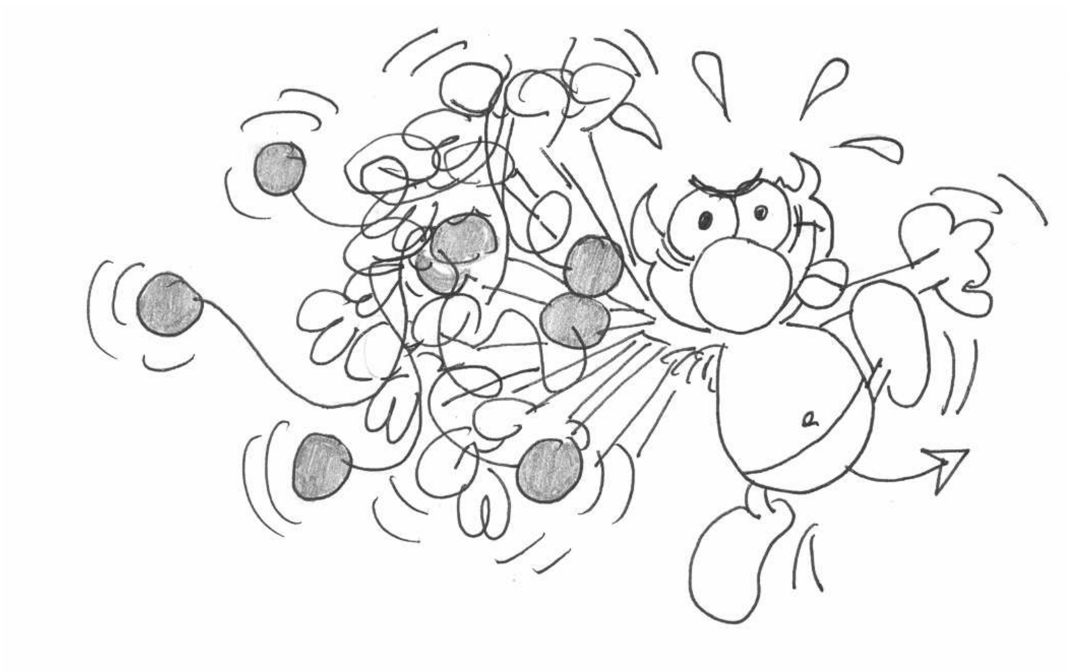


Figure 4.3: Géométrie du forçage pour le problème du pendule forcé (ici de manière exaggérée...).

La décomposition en deux équations différentielles d'ordre 1 donne alors:

$$\frac{\mathrm{d}\theta}{\mathrm{d}t} = v \ , \tag{4.14}$$

$$\frac{\mathrm{d}v}{\mathrm{d}t} = -\omega^2 \left(1 + \gamma \cos(\omega_0 t)\right) \sin\theta \ , \tag{4.15}$$

où on a défini un paramètre γ mesurant l'amplitude relative du forçage par rapport à la gravité:

$$\gamma = \frac{g_0}{g} \ . \tag{4.16}$$

Il y a donc maintenant deux paramètres additionnels contrôlant le comportement du pendule: la fréquence ω_0 et l'amplitude relative γ du forçage harmonique. Dépendant des valeurs choisies pour ces paramètres, le pendule peut se comporter de manière carrément bizarre. En particulier, on pourrait s'attendre que dans la limite $\gamma \to 0$ le comportement du pendule devienne identique à celui du pendule non-forcé; comme on le verra dans ce qui suit, celà s'avère ne pas être toujours le cas, dépendant de la valeur choisie pour la fréquence de forçage ω_0.

Certains comportements du pendule forcé harmoniquement sont merveilleusement contre-intuitifs. L'**excitation paramétrique** l'est particulièrement. Considérons ce qui se passe si le forçage harmonique du pendule se fait à une fréquence ω_0 qui est un multiple pair de la fréquence d'oscillation $\omega = \sqrt{g/L}$ (e.g., $\omega_0 = 2\omega$). Si l'amplitude d'oscillation est suffisamment petite pour que le pendule opère dans le régime linéaire, alors en relâchant notre pendule au moment opportun on se retrouve dans une situation où durant la phase "descendante" du pendule, le forçage du pivot est en phase remontante, ce qui augmente l'accélération angulaire de la masse m; et quand le pendule reprend la phase remontante, le forçage est vers le bas, ce qui réduit la gravité effective et fait que la masse m a moins tendance à ralentir (voir Fig. 4.4). Ces deux effets feront que la masse m remontera plus haut qu'elle ne l'aurait fait en l'absence du forçage. Mais puisque $\omega_0 = 2\omega$, à la prochaine descente ça repart de la même façon. Donc,

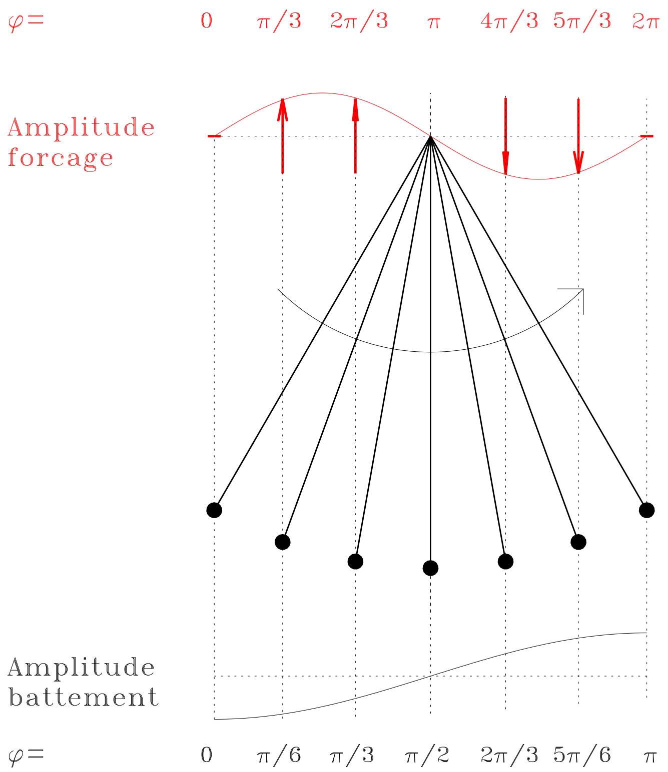


Figure 4.4: Opération du mécanisme d'excitation paramétrique, dans la situation où $\omega_0 = 2\omega$. Le diagramme illustre une demie-oscillation du pendule, ici de la gauche vers la droite. Les flèches verticales rouges au haut du diagramme indiquent la phase et l'amplitude du forçage du pivot. On notera que le forçage complète un cycle complet durant un demi-cycle d'oscillation du pendule.

en un cycle complet d'oscillation du pendule, la masse m subit deux cycles d'amplification par le forçage externe. Il est donc possible, à partir d'une oscillations infinitésimale, de produire une oscillation de forte amplitude; aurement dit, de notre point de vue le pendule se met à osciller "spontanément". Cette apparente violation des lois de la physique n'en est évidemment pas une, puisque que le système dispose d'une source d'énergie "externe", soit le mécanisme faisant osciller le pivot. Le point important demeure que l'oscillation du pivot peut transférer de l'énergie au pendule et amplifier son mouvement.

4.5 Le pendule forcé et amorti

Il est maintenant temps de combiner l'amortissement hydrodynamique et le forçage harmonique. Nous solutionnons donc maintenant:

$$L\frac{\mathrm{d}^2\theta}{\mathrm{d}t^2} = -(g + g_0\cos(\omega_0 t))\sin\theta - \frac{\alpha L}{m}\frac{\mathrm{d}\theta}{\mathrm{d}t} , \tag{4.17}$$

comme d'habitude exprimée sous la forme de deux équations différentielles d'ordre 1:

$$\frac{\mathrm{d}\theta}{\mathrm{d}t} = v , \tag{4.18}$$

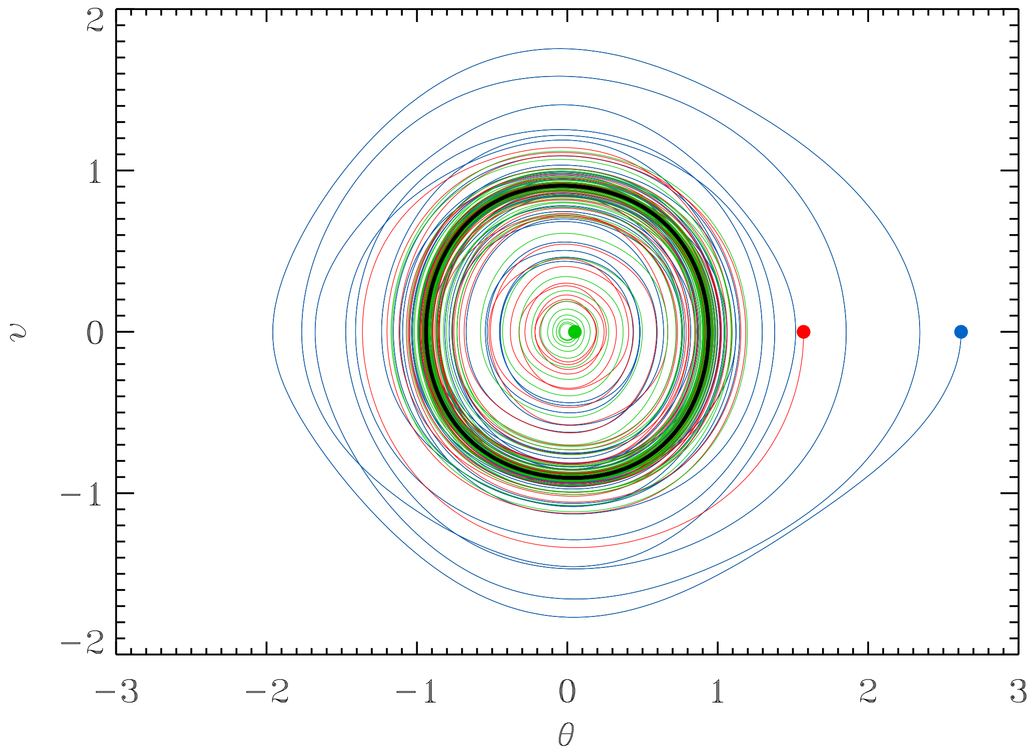


Figure 4.5: Trajectoires dans l'espace de phase pour trois solutions numériques au problème du pendule forcé et amorti. Toutes les solutions utilisent les valeurs de paramètres $\omega_0 = 2\omega$, $\gamma = 0.25$, et $\beta = 0.03$. les conditions initiales sont indiquées par des • colorés.

$$\frac{\mathrm{d}v}{\mathrm{d}t} = -\omega^2(1 + \gamma\cos(\omega_0 t))\sin\theta - \beta v \ , \tag{4.19}$$

avec γ et β donnés par les éqs. (4.16) et (4.10), comme auparavant. Notons que dans le cadre de la méthode de Heun, l'extrapolation linéaire au pas t_{n+1} du membre de droite de l'éq. (4.19) prend la forme:

$$v^*_{n+1} = v_n + h\left(-\omega^2(1 + \gamma\cos(\omega_0 t_n))\sin\theta_n - \beta v_n\right) \ , \tag{4.20}$$

où on note en particulier que le terme de forçage harmonique est également évalué à t_n dans le cadre de l'interpolation, même si ici il pourrait en fait être évalué à t_{n+1} de manière exacte. Il est important de préserver la cohérence interne de la méthode de Heun: tout au membre de droite doit être évalué à mi-pas!

Intuitivement, on peut imaginer que même en présence de l'amortissement il puisse exister un état stationnaire autre que $(\theta, v) = (0, 0)$, puisqu'une source d'énergie extérieure peut amplifier le mouvement pendulaire. La Figure 4.5 montre les trajectoires dans l'espace de phase de trois solutions numériques de l'équation (4.17) par la méthode de Heun, toutes avec $\omega_0 = 2\omega$, $g_0/g = 0.25$, et $\alpha = 0.03$, mais différant au niveau de la condition initiale. Quelle que soit cette dernière, les trajectoires aboutissent toutes sur l'orbite décrite par le trait noir épais. Nous avons de nouveau affaire ici à un attracteur, mais cette fois-ci ce n'est plus un point mais une figure géométrique plus complexe, soit une courbe fermée décrivant une oscillation quasi-harmonique d'amplitude constante.

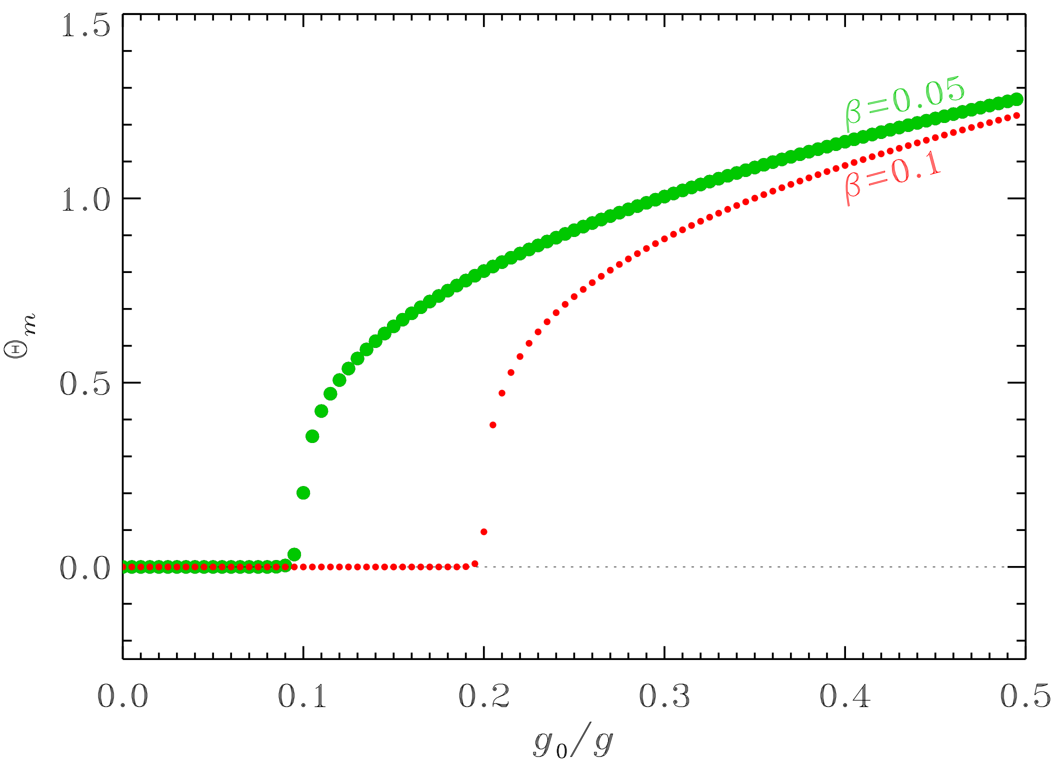


Figure 4.6: Amplitude maximale de battement dans l'état stationnaire, pour une séquence de solutions avec $\omega_0 = 2\omega$ et $\beta = 0.05$ (vert) et $\beta = 0.1$ (rouge), en fonction de de l'amplitude $\gamma = g_0/g$ du forçage harmonique.

4.6 Bifurcations

Si sur la Figure 4.5 la forme de l'attracteur dans l'espace de phase ne dépend pas de la condition initiale, elle dépend cependant des paramètres physiques du problème, soit le paramètre γ mesurant l'amplitude de forçage (éq. (4.16)), et le paramètre β mesurant l'importance de l'amortissement par la friction de l'air. Considérons la procédure suivante. Gardant tous les paramètres du modèles fixes autres que le paramètre de forçage γ, on produit une séquence de solutions pour des valeurs de γ de plus en plus grandes. Pour chaque solution, on intègre jusqu'à la convergence sur l'attracteur, et on mesure le rayon de l'orbite (plus précisément, la valeur maximale Θ_m atteinte par la variable angulaire (e.g., $\Theta_m = 1\,\mathrm{rad}$ sur la Figure 4.5). On porte ensuite en graphique ces valeurs de Θ_m, en fonction de la valeur du paramètre γ. Le résultat est illustré à la Figure 4.6, pour deux valeurs du paramètre d'amortissement β (points verts et rouge).

Certains aspects de cette Figure sont satisfaisant intuitivement. À forçage égal, les solutions avec plus faible amortissement se retrouvent à produire des amplitudes de battement plus élevées; et à amortissement égal, plus l'on force fort plus l'amplitude de battement est grande. Cependant le détail de la variation de Θ_m avec γ est pas mal moins intuitif. Pour un β donné, il existe de toute évidence une valeur de γ en dessous de laquelle il n'y a pas du tout de croissance de l'amplitude par excitation paramétrique, et quand elle débute, elle débute de manière soudaine et rapide. La valeur de γ à laquelle ceci se produit est appellée **point critique**, et le changement dans la dynamique globale du système se produisant quand on croise ce point est appelé **bifurcation**. On notera que la solution d'équilibre demeure une solution tout-à-fait valide même au delà du point de bifurcation; cependant, cette solution se retrouve **instable** par rapport à la solution oscillatoire, mais seulement si l'énergie injectée par le mécanisme de forçage peut dépasser l'énergie perdue en friction contre l'air. C'est pourquoi le point critique a une valeur numérique finie, qui croit à mesure que β augmente.

La bifurcation est un phénomène typique des systèmes nonlinéaires. Dans bien des situations que vous avez déjà rencontrées dans vos études de la physique, il existe une relation linéaire entre une "cause" et un "effet". Avec $\mathbf{F} = m\mathbf{a}$ par exemple, quelle que soit la valeur de $\mathbf{F}$, si vous l'augmentez de 1% (disons) vous vous attendez à une augmentation directement proportionnelle de l'accélération $\mathbf{a}$. Contrastez ceci à la situation présentée à la Figure 4.6, par exemple pour $\beta = 0.1$ (points rouge). Si $\gamma = 0.1$, le pendule n'oscille pas, et une augmentation de γ par 1% (à $\gamma = 0.101$) ne le fera pas osciller plus. À $\gamma = 0.4$ par contre, une augmentation de 1% à $\gamma = 0.404$ produira une augmentation quasi-proportionnelle de l'amplitude maximale d'oscillation. Mais à $\gamma = 0.2$, une petite augmentation de 1% nous fait passer d'un système qui n'oscille pas du tout à un système qui oscille avec une amplitude substantielle, $\Theta_m \simeq 0.5\,\mathrm{rad}$. En fait, au point de bifurcation même on a $\mathrm{d}\Theta_m/\mathrm{d}\gamma \to \infty$. Même à un niveau purement qualitatif (osciller ou ne pas osciller, zattize ze quèstcheune...), la réponse du système à une petite variation d'un paramètre dépend donc de manière très sensible de la valeur exacte de ce paramètre; quand un point de bifurcation est traversé, le système passe à un état dynamique différent. C'est ce qui fait toute la complexité des systèmes nonlinéaires.

4.7 Le chaos

Une orbite stable dans l'espace de phase, telle qu'illustrée à la Figure 4.5 est possible ici en présence d'un amortissement important en raison du fait que la fréquence de forçage a été judicieusement choisie afin de profiter de la résonance par excitation paramétrique. Qu'en est-il de l'évolution du pendule forcé si la fréquence de forçage n'a pas une valeur permettant l'excitation paramétrique? Le forçage transfère toujours de l'énergie au système, donc ou pourrait s'attendre à ce que pour un amortissement pas trop important et/ou une amplitude de forçage suffisamment élevée, le mouvement pendulaire puisse perdurer. Quelle forme prend alors ce mouvement?

La Figure 4.7 montre l'évolution temporelle d'un pendule forcé et amorti, avec valeurs de paramètres $\omega_0 = 1.3\omega$, $\gamma = 0.75$, et $\beta = 0.03$, et une condition intiale $(\theta_0, v_0) = (5\pi/6, 0)$. La solution est obtenue ici par la méthode de Heun, avec un pas de temps $h = 0.014$. Chaque image montre le pendule à 10 époques successives équidistantes dans le temps, avec la teinte de gris codant le temps: le pendule le plus pâle correspond que premier pas de temps du groupe de dix, et le plus foncé au dernier. Chaque image représente la suite de la précédente, de gauche à droite et du haut vers la bas, tel que numéroté; le pendule en noir sur la première image est donc à la même position que celui en gris le plus pâle sur la seconde, en ainsi de suite d'une image à l'autre[1]. Le mouvement est de toute évidence très complexe, et pas harmonique ou quasi-harmonique du tout! Notez en particulier comment le pendule parvient parfois à effectuer quelques tours complets successifs autour du pivot (e.g., images 17→20).

La Figure 4.8 montre l'évolution de la même solution, cette fois dans le sous-espace de phase $[\theta, v]$ du système. On parle de sous-espace ici parce que l'espace de phase complet inclut maintenant une troisième dimension, soit celle de l'amplitude de forçage du pivot, qui ajouterait ici une déviation harmonique des trajectoires hors du plan de la feuille. C'est pourquoi la trajectoire semble se croiser occasionnellement ici, chose impossible dans l'espace de phase complet d'un système déterministe. On voit que le pendule est parfois "capturé" par l'un ou l'autre des points fixes à $(\theta, v) = (2n\pi, 0)$ pendant quelques orbites, mais la grande amplitude de forçage utilisée ici ($\gamma = 0.75$) réussit à l'en extirper malgré l'amortissement substantiel ($\beta = 0.03$), ce qui fait que la solution "vagabonde" d'une cellule périodique de l'espace de phase à l'autre, d'une manière en toute apparence erratique. Un saut d'une cellulle correspond à une révolution complète du pendule autour du pivot, dans le sens horaire (saut à droite) ou antihoraire (saut à gauche).

[1] Une animation de cette solution peut être visionnée sur la Page Web du cours.

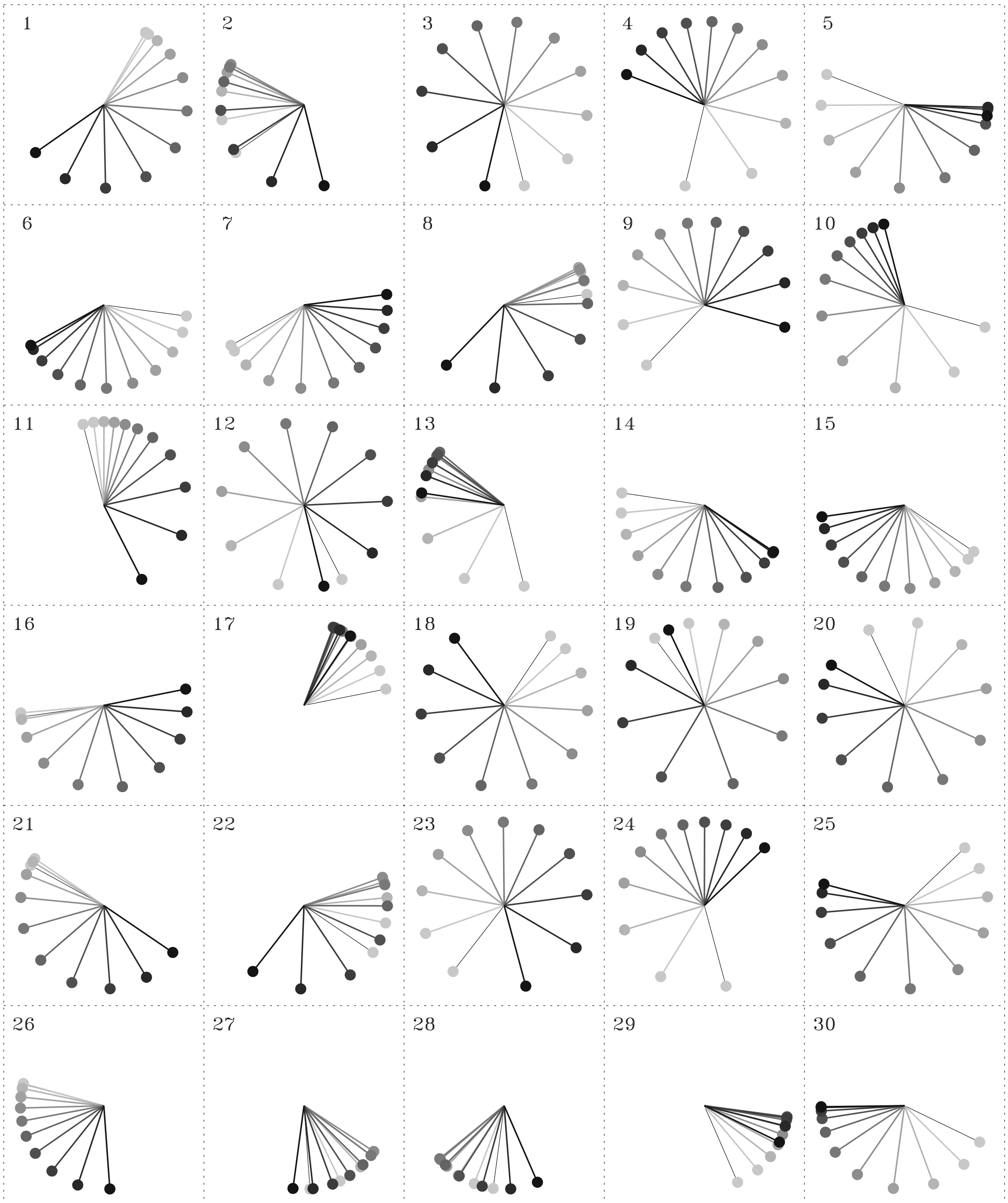


Figure 4.7: Battement d'un pendule nonlinéaire amorti et forcé ($\omega_0 = 1.3\omega$, $\gamma = 0.75$, $\beta = 0.03$). Chaque image montre 10 positions successives du pendule espacées également dans le temps, selon une séquence allant du gris clair vers le noir. Chaque image successive reprend là où la précédente se termine, de la gauche vers la droite, et du haut vers le bas, tel que numéroté.

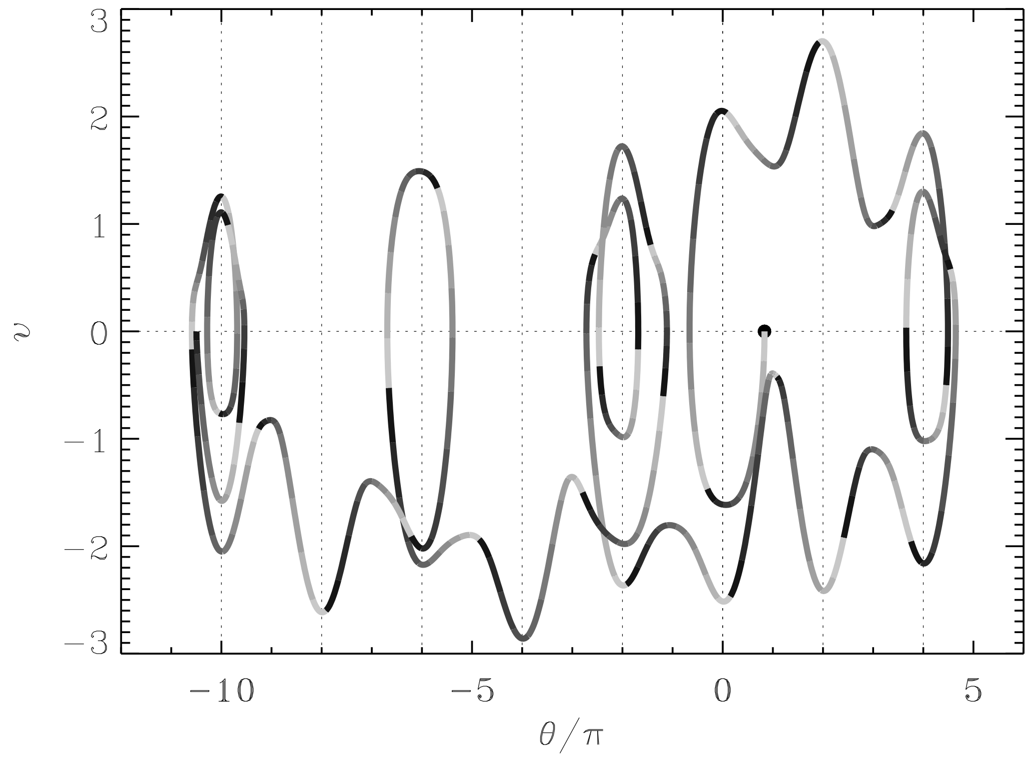


Figure 4.8: Évolution de la même solution qu'illustrée sur la Figure 4.7, cette fois dans le sous-espace de phase $[\theta, v]$ du système. Le codage en teintes de gris de la trajectoire suit celui de la Figure 4.7, chaque segment successif gris→noir correspondant à une image.

Mathématiquement parlant, le mouvement du pendule est en fait ici plus qu'erratique, il est **chaotique**. Le chaos est une notion qui a été apprêtée à toutes les sauces, et il est donc important de bien spécifier (et même quantifier) ce que l'on appelle chaos ici.

Considérons deux solutions ayant des valeurs de paramètres identiques, soit $\beta = 10^{-4}$, $\omega_0 = 1.3\omega$, $\gamma = 0.25$, mais ne diffèrant que par leur condition initiale, et même là vraiment très peu:

$$(\theta, v)_1 = (\pi/2, 0)\ , \qquad (\theta, v)_2 = (\pi/2 + 10^{-4}, 0)\ . \tag{4.21}$$

On utilise encore une fois ici la méthode de Heun, avec un pas de temps $h = 0.019$. Au début, les deux solutions sont évidemment indistingables "à l'oeil"... mais seulement pendant trois cycles d'oscillation; vers $\omega t/2\pi = 25$, une différence devient notable, et à partir de $\omega t/2\pi \simeq 30$ les deux solutions divergent complètement l'une de l'autre. Cette divergence n'est pas due à un effet d'instabilité numérique associée à une erreur d'arrondissement, ou rien du genre; la divergence des deux solution a lieu même si on diminue ou augmente la taille du pas de temps, ou si on passe à Runge-Kutta, etc. L'effet est réel, et nous devons en comprendre l'origine.

Commençons par nous définir une mesure quantitative de l'écart entre les deux solutions; il est naturel d'utiliser la distance entre les deux solutions dans le sous-espace de phase $[\theta, v]$, définie comme:

$$\delta(t) = \sqrt{(\theta_1(t) - \theta_2(t))^2 + (v_1(t) - v_2(t))^2}\ . \tag{4.22}$$

En vertu des conditions initiales adoptées ici, la distance initiale entre les deux solutions est de $\delta = 10^{-4}$ à $t = 0$. La Figure 4.9 montre l'évolution temporelle de cette quantité, avec

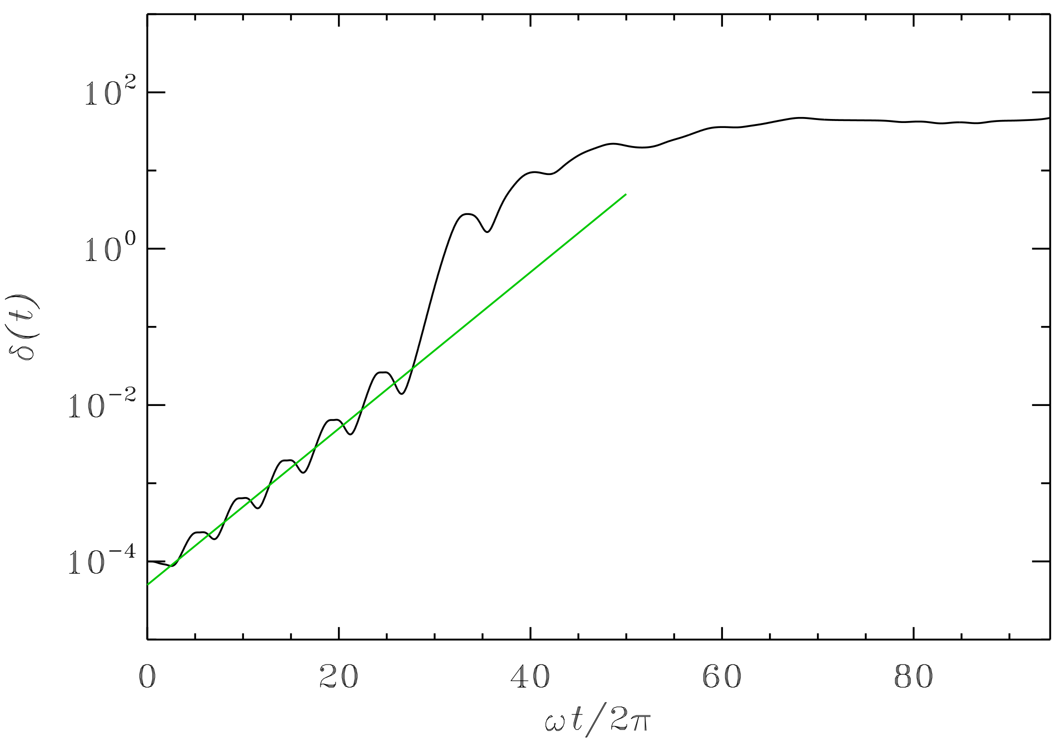


Figure 4.9: Évolution de la distance dans l'espace de phase, définie selon l'éq. (4.22) entre deux solutions au problème du pendule forcé et amorti, toutes les deux ici avec les même valeurs de paramètres $\omega_0 = 1.3\omega$, $\beta = 10^{-4}$, et $\gamma = 0.25$. Les deux solutions ne diffèrent que très légèrement au niveau de leur condition initiale: $(\theta, v) = (\pi/2, 0)$ et $(\pi/2 + 10^{-4}, 0)$. Le trait vert indique la phase de croissance exponentielle de l'écart (voir texte).

une échelle verticale logarithmique. On remarquera que l'écart entre les deux solutions croit *exponentiellement*, essentiellement depuis $t = 0$, jusqu'à ce que les deux solutions divergent complètement. Quelle que soit la condition initiale, ou l'amplitude de l'écart introduit à $t = 0$, pour des valeurs des paramètres β ω_0, et γ fixes, cette croissance exponentielle se fait toujours au même taux Λ, correspondant à la pente du trait vert sur la Fig. 4.9. En d'autres mots, on peut écrire:

$$\delta(t) = \delta_0 \exp(\Lambda t) \ . \tag{4.23}$$

La constante Λ est appelée **exposant de Lyapunov**, et le fait que cet exposant soit positif définit le système comme étant **chaotique**.

Il nous reste à comprendre ce qui mets fin à la phase exponentielle de croissance de l'écart entre les deux solutions. Pour ce faire il s'avèrera utile de considérer l'évolution des deux solutions dans l'espace de phase $[\theta, v]$, tel qu'illustré au haut de la Figure 4.10.

Les deux trajetoires débutent sur une orbite fermée centrée sur $(\theta, v) = (0, 0)$, mais après trois orbites le forçage les fait approcher, puis sauter la séparatrice, après quoi les solutions se dirigent vers le point répulseur $(\theta, v) = (\pi, 0)$ (voir premier encadré). Les deux solutions sont redéviées le long de la séparatrice. Cependant, la solution en rouge est maintenant "retardée" par rapport à la bleue, ayant pris plus de temps à contourner $(\theta, v) = (\pi, 0)$. Les deux solutions sont donc maintenant partiellement déphasées par rapport au forçage harmonique; comme on peut le voir sur l'encadré du bas, elles croisent et recroisent la séparatrice à différent temps et à différentes positions sur la séparatrice. Une fois une révolution supplémentaire complétée, à

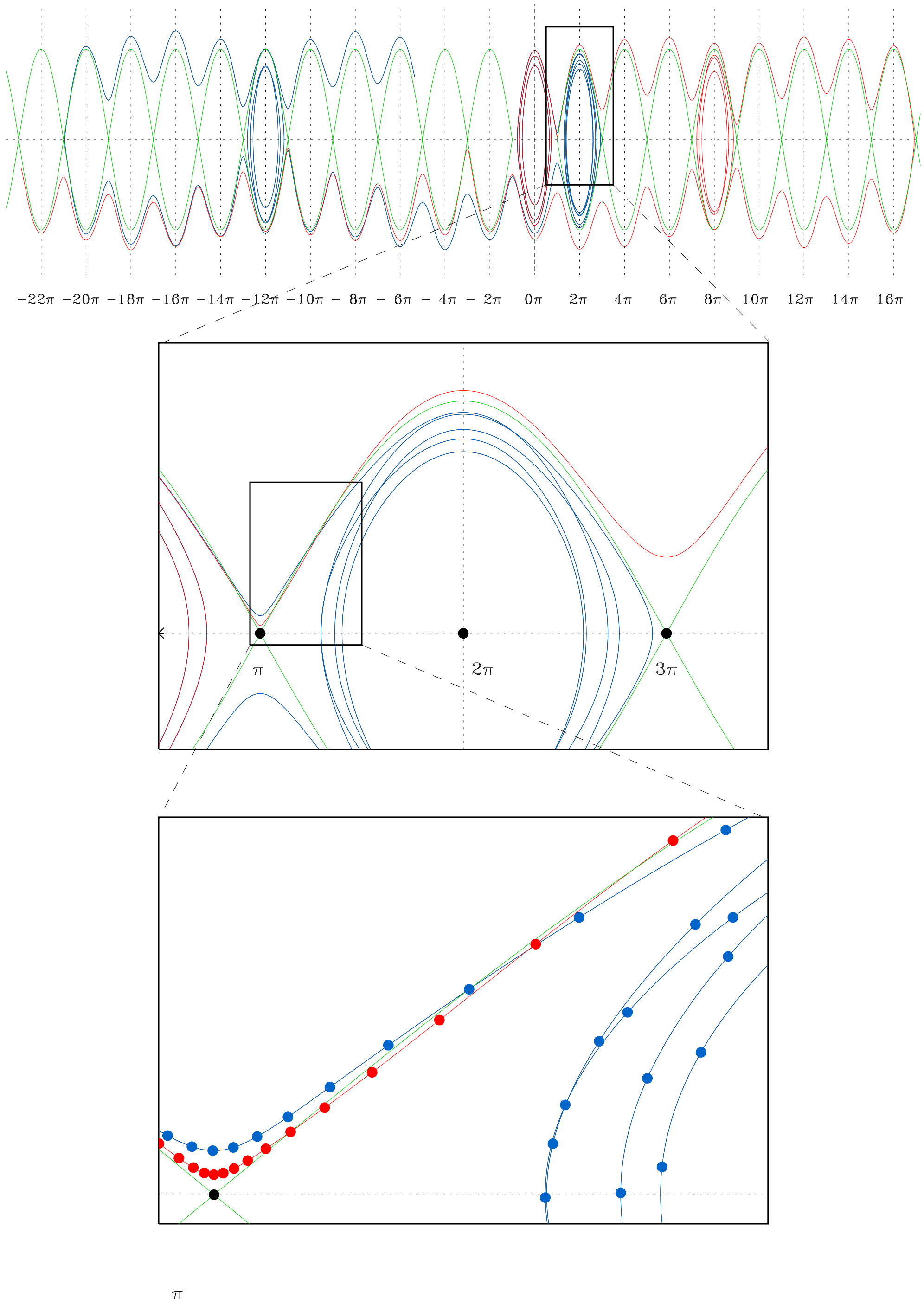


Figure 4.10: Trajectoires dans l'espace de phase des deux solutions de la Figure 4.9, avec deux zoom successifs sur la région où se produit la décorrélation (voir texte). Les courbes en vert correspondent aux séparatrices dans l'espace de phase. Sur l'encadré du bas, les • colorés sont tracés à des intervalles de temps constant; remarquez comme la solution en rouge "ralentit" plus que la bleue, en passant plus près du point $(\theta, v) = (\pi, 0)$.

l'approche du prochain point répulseur à $(\theta, v) = (3\pi, 0)$, la solution rouge est au dessus de la séparatrice tandis que la bleue est en dessous; la première repart donc vers le haut, tandis que la deuxième continue vers le bas sur une orbite fermée autour de $(\theta, v) = (2\pi, 0)$. Du point de vue du pendule dans notre "vrai" espace physique, la première solution correspond à un pendule qui, oscillant avec une forte amplitude, approche, ralentit, mais parvient tout de même à dépasser le point d'équilibre instable, tandis que la seconde solution décrit une solution qui ralentit au point mort tout juste avant d'avoir atteint la verticale $\theta = \pi$, et repart en sens inverse. C'est donc aux alentours du point d'équilibre instable que se décide la décorrélation du système.

4.8 Chaos ≠ aléatoire

Notre étude du chaos s'est limité ici à un système physique très simple, soit le pendule amorti et forcé. Le chaos s'observe cependant dans bon nombres de systèmes physiques ou biologiques, naturels ou artificiels: orbites d'astéroides, certains circuits électriques, turbulence dans les fluides, interactions prédateur-proie, certaines formes d'arythmie cardiaque, réactions chimiques, chavirement des navires par les vagues, fluctuations dans les cavités laser, pour n'en nommer que quelques-uns. Si on avait à trouver un point commun à la dynamique de ces systèmes si divers, ce serait leur grande sensibilité par rapport aux conditions initiales. Une minuscule différence entre deux solutions tend à s'amplifier avec le temps. Et comme vous pourrez le vérifier en faisant le problème 4.4 ci-dessous, cette "petite différence" peut même être produite par le choix du schéma numérique utilisé pour solutionner le problème.

Mais comment reconnaitre le chaos dans un système naturel ou artificiel? Un comportement "désordonné" ("chaotique" dans le sens colloquial du terme) est un indice mais non une preuve. Un système évoluant selon une dynamique stochastique (par exemple la marche aléatoire que nous découvrirons dans quelques semaines) évolue également de manière "désordonnée", mais on est très loin du chaos. Ajoutez à ça l'inévitable présence d'erreurs de mesure dans une observation ou une manip expérimentale, et faire la part observationnellement entre un système stochastique et un autre étant chaotique peut devenir un solide défi. Les références listées en bibliographie en fin de chapitre pourront vous montrer la voie si ce genre de trucs vous intéresse.

Exercices:

1. Ce problème vise à vous faire cogiter un peu plus sur le concept d'espace de phase;
   (a) Tracez la trajectoire, dans un espace de phase hauteur-vitesse, d'une masse m lancée verticalement vers le haut à partir du sol, à une vitesse initiale v_0 (à gravité constante).
   (b) Tracez "intuitivement" une seconde trajectoire, dans un cas ou le déplacement de la masse m est influencé de manière appréciable par la friction de l'air.

2. Revenons au pendule nonlinéaire classique, i.e. sans amortissement ou forçage (comme au chapitre 3). Obtenez une séquence de solutions numériques l'aide de la méthode de Heun, pour des conditions initiales du genre $v = 0$ et $\theta_0 = \pi - \epsilon$, où $\epsilon \to 0$. Calculez le temps T requis pour faire une oscillation complète, en fonction de ϵ. Utilisez vos résultats pour vérifier que:
$$\lim_{\epsilon \to 0} T \to \infty$$
   Attention: ce problème, comme tous les problèmes de type "stabilité", est très délicat du point de vue numérique. Assurez-vous de travailler en double précision.

3. Passons maintenant au pendule amorti, mais non-forcé. Toujours à l'aide de la méthode de Heun, calculez une séquence de solutions, toutes avec la condition initiale $(\theta, v) =$

$(3\pi/4, 0)$, pour des valeurs du paramètre d'amortissement β allant de 0.01 à 1.0. Pour chacune de vos solutions, mesurez le temps T requis pour que l'amplitude d'oscillation chute et demeure sous $\theta = 10^{-2}$ rad. Sur la base de vos résultats, essayer d'établir la relation mathématique décrivant la variation de T avec β.

4. Calculez deux solutions au problème du pendule forcé et amorti dans le régime chaotique, utilisant toutes la même condition initiale et le même pas de temps. Utilisez pour la première la méthode d'Euler explicite, et pour la seconde la méthode de Heun. Combien de temps s'écoule avant la décorrélation des deux solutions?

Bibliographie:

Le chaos est sujet qui a produit une littérature gigantesque et souvent indigeste. Parmi la douzaine de bouquins techniques que je connais sur le sujet, je recommenderais les deux suivants:

Mullin, T., *The Nature of Chaos*, Oxford University Press (1993),
Hilborn, R.C., *Chaos and Nonliner Dynamics*, 2^{e}éd., Oxford University Press (2000).

Parmi la quasi-infinité d'ouvrages à saveur plus philosophique, j'ai bien aimé:

Prigogine, I., *Les Lois du Chaos*, Flammarion (1994)

Chapitre 5

Monte Carlo

Supposons que vous vous voyez assigner la tâche (ingrate) de mesurer la superficie d'un étang de forme irrégulière. Vous pourriez obtenir une carte topographique de l'étang et de ses environs immédiats, quadriller le tout, compter le nombre de petit carrés contenus dans l'étang, et multiplier ce nombre par la surface d'un de vos petits carrés (voir Figure 5.1A). Ceci revient en fait à une forme d'intégration en deux dimensions spatiales, d'une fonction qui (par exemple) vaut un dans l'étang, et zéro sur la terre ferme. À examiner la Figure 5.1A, vous pouvez fort bien imaginer que la principale source d'incertitude ici est le degré auquel votre quadrillage réussit à bien capturer la forme du contour de l'étang, et que le plus fin sera votre quadrillage de l'étang, le plus précis sera votre détermination de sa surface.

Vous avez aussi une autre option; délimiter une surface A facilement calculable (e.g., un carré) englobant l'étang; lancez ensuite un nombre N de cailloux distribués de manière complètement aléatoire spatialement, et comptez le nombre P de fois où vous entendez un "plouf". L'idée est illustrée à la Figure 5.1B, pour $N = 10^3$. Pour un N très grand, la surface de l'étang sera donnée simplement par $(P/N) \times A$.

Ceux et celles dont la mémoire n'est pas encore saturée en cette seconde moitié de session auront reconnu que cette approche à la mesure de la superficie de l'étang est celle sous-jacente à notre calcul de π du chapitre 1. L'équivalent de l'étang y est un quart de cercle de rayon unitaire, et l'estimé de π vient du fait que l'aire d'un cercle (complet) de rayon unitaire vaut π, donc celle d'un quart de cercle vaut $\pi/4$, et la surface d'un carré englobant ce quart de cercle de rayon unitaire vaut 1. La boucle lance 10^6 cailloux à des positions aléatoires $(x, y) = (r_1, r_2)$, donc la distance radiale à l'origine est donnée par $\sqrt{r_1^2 + r_2^2}$; si cette distance est plus petite que un, le caillou est tombé dans le quart de cercle, et on ajoute un au compteur. La valeur finale de ce compteur divisée par 10^6 correspond donc au rapport de la surface du quart de cercle à celle du carré l'englobant; il ne reste plus qu'à multiplier par 4, et on a le nombre π. Voilà!

Cette approche statistique au calcul de quelque chose qui en principe pourrait se faire de manière déterministe est appelée une méthode **Monte Carlo**. Dans ce chapitre nous nous limiterons à deux applications spécifiques, soit le calcul des intégrales (§5.4), et une modélisation de type Monte Carlo qui revient effectivement à la solution de certaine classes d'équations différentielles (§5.5). Une troisième emergera "spontanément" au chapitre suivant (§6.6). Si vous avez suivi l'exemple de l'étang, vous vous doutez déjà qu'un aspect essentiel de l'approche Monte Carlo est la génération de nombres aléatoires, que nous devons d'abord régler avant de passer au Monte Carlo comme tel.

5.1 Nombres aléatoires

Une séquence de nombres est dite **aléatoire** si la valeur numérique de chaque membre de la séquence est complètement indépendante de celles des membres précédents dans la séquence.

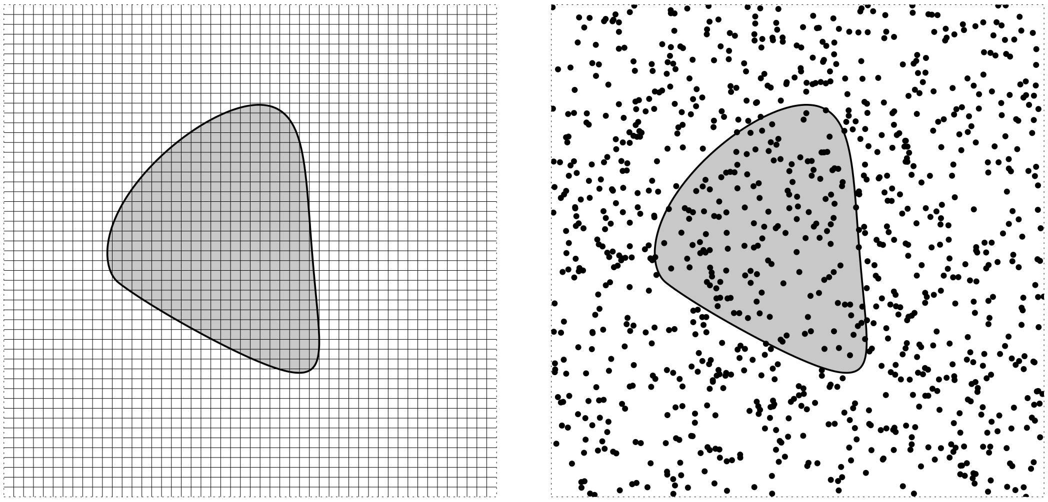


Figure 5.1: Deux approches fondamentalement différentes à la mesure de la surface d'une surface de forme irrégulière (en gris ici). À gauche, un quadrillage cartésien régulier, où la détermination de la surface revient à compter le nombre de carrés couvrant la surface. À droite, une approche statistique où la surface est donnée par la fraction du nombre de lancers aléatoires tombant sur la surface (voir texte).

Par exemple, si vous roulez un dé (non-pipé) douze fois de suite, vous pourriez tout aussi bien obtenir l'une ou l'autre des deux séquences suivantes:

$$4-2-6-3-4-4-2-6-1-5-2-6\ ,$$

$$6-6-1-2-3-3-5-6-3-6-2-3\ .$$

Vous serez peut-être surpris de noter que la seconde séquence ne contient pas de "4". La probabilité de ne pas rouler 4 est $1-1/6=5/6$; donc la probabilité de ne pas rouler un quatre 12 fois de suite est $(5/6)^{12}=0.112$, ce qui est petit mais bien loin de l'astronomiquement minuscule (contrairement à votre probabilité de gagner au 6-49, qui elle l'est). Moins évident mais tout aussi vrai, si chaque lancer est vraiment entièrement indépendant des précédents, la probabilité d'obtenir exactement l'une de ces séquences est $(1/6)^{12}$, soit un minuscule 4.6×10^{-10}, et *exactement* la même que celle d'obtenir la séquence qu'une majorité d'individus jugeraient (incorrectement) encore plus improbable:

$$1-2-3-4-5-6-1-2-3-4-5-6\ ,$$

ou encore

$$6-6-6-6-6-6-6-6-6-6-6-6\ .$$

Eh oui! Si vous roulez maintenant le dé un très grand nombre de fois (N, disons), on s'attendrait à obtenir $N/6$ fois le "1", $N/6$ fois le "2", et ainsi de suite jusqu'à 6. C'est la définition même d'un dé non-truqué! Le dé est donc un **générateur** de nombres aléatoires, ici des entiers distribués uniformément dans l'intervalle $[1,6]$.

Il s'agit maintenant de produire l'équivalent d'un dé utilisable sur l'ordinateur. Ceci peut paraitre un non-sens absolu et total; un programme d'ordinateur est déterministe au plus haut point, dans le sens que sur une architecture donnée, l'exécutable d'un algorithme auquel on fournit la même entrée produira *toujours* la même sortie. Donc, si on revient à notre séquence de lancers d'un dé, le résultat du n-ième lancer sera toujours *complètement* déterminé par l'état du générateur après le $n-1$-ième lancer, autrement dit, les valeurs numériques des lancers $n-1$ et n sont complètement corrélées, l'antithèse même de notre idée de l'aléatoire!

La solution à ce paradoxe tient au fait que même si les lancers de pseudo-dés numériques sont corrélés à 100%, il est possible de s'assurer que, tout comme avec un vrai dé, chaque valeur sort en moyenne aussi fréquemment que les autres, et que *sur l'ensemble de la séquence* aucune corrélation n'existe en la valeur numérique d'un lancer et celle du suivant. Ce sont ces propriétés *statistiques* de la séquence qui lui méritent le nom d'aléatoire (certains puristes insisteraient ici sur l'appellation "pseudo-aléatoires".

Les bases théorique, arithmétique, statistique et informatique de la génération des nombres aléatoires pourraient nous tenir occupés bien longtemps. Pour nos besoin présents, il suffit de réaliser que la génération de séquences qui satisfont aux propriétés énoncées ci-dessus est possible mais non-triviale, et que certains générateurs sur le marché sont nettement meilleurs que d'autres. Par exemple, le langage C standard inclut une fonction intrinsèque appelée `rand()`, qui produit un entier uniformément distribué dans l'intervale $[0, 2147483647]$, soit l'équivalent d'un dé à 2147483648 faces (sur un système monté en 64-bits; monté en 32-bits, ça aurait été 32768). C'est le générateur utilisé dans le calcul de π au chapitre 1. La manière facile et portative d'utiliser le générateur `rand()`, sans avoit à se préoccuper du 32 bits versus 64 bits, serait en fait la suivante:

```
#include <stdlib.h>
int main(void)
{
  ...
  float r ;
  ...
  r=1.*rand() / RAND_MAX
  ...
}
```

La variable globale `RAND_MAX` est une constante préféfinie du langage C, de type `int` et contenue dans la librairie `stdlib.h` (qui doit dont être incluse dans le code, comme ci-dessus), et est égale au plus grand entier représentable sur l'architecture donnée. Donc `rand()/RAND_MAX` sera un réel entre 0 et 1... seulement si on prémultiplie `rand()` par 1., comme ci-dessus, pour forcer la conversion au type `float` *avant* la division par `RAND_MAX`; sinon on aura toujours $r = 0.$, et rendu à ce stade vous devriez vraiment pouvoir comprendre pourquoi...

Pour fins d'illustration et de didactique, et pour tous les exercices et labos de ce cours, c'est suffisant, mais pour une "vraie" application il faut faire mieux. Je vous inclue à la Figure 5.2 le code C pour le générateur `ran0` discuté dans *Numerical Recipes in C* (Chapitre 7 dans l'édition de 1992). Si vous êtes rendu en Maitrise ou plus loin et relisez ces notes pour vous souvenir comment produire des nombres aléatoires, SVP utilisez au moins celui-là, et peut-être même la variation appellée `ran1` également présentée dans *Numerical Recipes*.

Un générateur de nombre aléatoire doit toujours être initialisé à une valeur (arbitraire) pour débuter le processus de production de la séquence aléatoire. Cette valeur est appelée le **germe**. Sa valeur numérique importe peu, mais doit être fournie. Une routine apparamment sans germe, comme `rand()`, utilisera en fait un germe prédéfini ou l'heure au moment du début de l'exécution, etc. Pour la grande majorité des compilateurs C, ce germe prédéfini vaut simplement 1. Il est parfois utile (et souvent important!) de pouvoir générer des séquences qui soient distinctes et/ou qui puissent être reproduites à une date ultérieure. En définissant plusieurs germes distincts, il devient donc possible de produire des séquences de nombres aléatoires complètement indépendantes l'une de l'autre.

Le générateur `ran0` reproduit à la Figure 5.2 demande donc en argument un germe qui doit être défini comme un entier de type `long`. Cependant, la valeur de cette variable doit être

```
#include <stdio.h>
/* Exemple d'utilisation du generateur ran0 de Numerical Recipes    */
/* Notez bien: La variable seed et son pointeur ne doivent pas tre */
/* explicitement modifies en cours d'execution!                     */
int main(void)
{
  float ran0(long *idum) ; /* decl. fonction avec pointeur en argument */
  long *p_seed ;           /* definition d'un pointeur */
  long seed=2345 ;         /* definition/initialisation du germe */
  int   k ;
  float r ;

  p_seed=&seed ;           /* association pointeur-variable */
  for (k=0 ; k<20 ; k++) {
     r=ran0(p_seed) ;      /* Appel avec NOM du pointeur */
     printf ("\n%f",r) ;
  }
}
#define IA 16807           /* NE PAS CHANGER CES INITIALISATIONS ! */
#define IM 2147483647
#define AM (1.0/IM)
#define IQ 127773
#define IR 2836
#define MASK 123459876     /* C'EST BIEN COMPRIS !? */
/* Generateur ran0 de Numerical recipes in C (2eme ed. 1992) */
float ran0 (long *idum)
{
  long k ;
  float ans ;
  *idum ^= MASK ;
  k=(*idum)/IQ ;
  *idum=IA*(*idum-k*IQ)-IR*k ;
  if (*idum < 0) *idum += IM;
  ans=AM*(*idum) ;
  *idum ^= MASK ;
  return ans ;
}
```

Figure 5.2: Le générateur `ran0` de Numerical Recipes (version C de 1992); à utiliser plutôt que la générateur intrinsèque `rand()` pour quoique ce soit de sérieux. Ce générateur produit un nombre aléatoire extrait d'une distribution uniforme dans l'intervale $[0, 1]$ selon l'algorithme de Park-Miller (voir références en fin de chapitre). La principale subtilité ici est que l'argument de la fonction doit être changé au moment du retour de la fonction, ce qui en C requiert que l'argument à la fonction ne soit pas une variable (ici `seed`), mais un **pointeur** vers l'addresse en mémoire de cette variable. Nous discuterons en plus de détails de ces questions de pointeurs dans un chapitre ultérieur. Ce code source est disponible sur la page Web du cours.

fournie à la fonction sous la forme d'un **pointeur** identifiant l'addresse en mémoire du germe, ici la variable nommée `seed`, plutôt que la valeur de la variable même. En C un nom de pointeur doit être déclaré en le précédant d'un "*"; son association à la variable `seed` se fait via ici via l'instruction `p_seed=&seed`, qui veut littéralement dire "assigner au pointeur `p_seed` la valeur numérique de l'addresse en mémoire où est emmagasinnée la valeur de la variable `seed`".

Cette curieuse façon de passer la valeur de `seed` à `ran0` est requise par le fait que la variable `seed` est modifiée au cours d'un appel de `ran0` à l'autre, et qu'en C une variable passée en argument à une fonction conserve toujours sa valeur même si elle est explicitement changée à l'intérieur de la fonction (comme `*idum` dans la fonction `ran0` de la Figure 5.2). Vous avez en fait déjà fait face à cette utilisation au Labo 1, avec l'utilisation de la fonction `scanf`; ses arguments devant être modifiés à l'exécution, ce sont les addresse en mémoires qui sont passées en argument à la fonction. Il est donc important ici de ne pas changer explicitement la valeur de `seed` (ou `p_seed`) entre deux appels à `ran0`, puisque ceci pourrait fort bien détruire les propriétés statistiques du générateur, plus spécifiquement l'uniformité de la distribution dans l'intervalle. C'est le genre de manoeuvre dangereuse qui ne causerait pas d'erreur à l'exécution, mais qui pourrait bien produire des résultats fautifs en bout de ligne.

5.2 Fonctions de distributions

Un aspect critique de l'approche Monte Carlo en modélisation physique est de bien comprendre et contrôler la **distribution statistique** des nombres aléatoires utilisés pour l'analyse. On définit une **fonction de densité de probabilité** (ci-après FDP) $f(x)$ d'un processus (aléatoire ou non) produisant une séquence de candidats x_n , $n = 1, 2, 3...$ où $f(x)\mathrm{d}x$ donne la probabilité que la valeur numérique du candidat se retrouve entre x et $x + \mathrm{d}x$. Par exemple, dans le cas du générateur uniforme de la section précédente, on a

$$f(x) = \begin{cases} 1 & 0 < x \leq 1 \\ 0 & \text{sinon} \end{cases} \tag{5.1}$$

Une autre FDP avec laquelle vous avez probablement déjà fait connaissance est la distribution dite normale, ou Gaussienne:

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{-x^2}{2\sigma^2}\right) \ . \tag{5.2}$$

Notons que, par leur définition même, les FDP sont des distributions normalisées, dans le sens que

$$\int f(x)\mathrm{d}x = 1 \ . \tag{5.3}$$

Une fois la FDP connue, elle peut servir à calculer une multitude de caractéristiques globales de l'enemble de valeurs de x; la moyenne, par exemple, est donnée par

$$\langle x \rangle = \int f(x)\, x\, \mathrm{d}x \ . \tag{5.4}$$

Le hic est que la distribution statistique d'une séquence de N valeurs discrètes de x produites par un générateur basé sur de telles FDP deviendra identique à ces FDP seulement dans la limite $N \to \infty$. Comme en pratique on travaille toujours avec un N fini, il est important de comprendre, et de quantifier, jusqu'à quel point les deux distributions dévient l'une de l'autre.

Travaillons avec la distribution uniforme associée à la FDP (5.1), et supposons que l'on a produit N valeurs numériques $r_n \in [0, 1]$ à l'aide notre générateur de nombre aléatoire uniforme. Définissons d'abord la **fonction histogramme** associée à cet ensemble de nombre aléatoires. Ces fonctions sont construites comme suit: on définit une séquence de M intervalles d'échantillonnage contigus et de largeur fixe b:

$$[x_m, x_m + b] \ , \qquad x_m + b = x_{m+1} \ , \qquad m = 1, ..., M \tag{5.5}$$

avec ici $x_1 = 0$ et $x_M + b = 1$. Puisque les intervalles sont contigus, on a évidemment $b = 1/M$. On fait ensuite le décompte N_m de toutes les valeurs de r tombant dans chaque intervalle:

$$N_m = N_m + \begin{cases} 1 & \text{si } x_m < r_n \leq x_m + b \\ 0 & \text{sinon} \end{cases} \quad , \qquad m = 1, ..., M \ , \qquad n = 1, ..., N \tag{5.6}$$

et la fonction histogramme h_m est alors définie comme

$$h_m(r_n; b) = \frac{N_m}{bN} \ , \qquad m = 1, ..., M \ . \tag{5.7}$$

Ici la notation "$(r_n; b)$" indique que la fonction histogramme est à la fois une fonction des données du problème, soit les N valeurs de r_n, ainsi que du choix de la largeur de l'intervalle d'échantillonnage b, qui n'est pas une donnée du problème mais plutôt un choix de l'utilisateur (vous!); d'où l'utilisation du point-virgule pour distinguer ces deux types de "variables" définissant la fonction histogramme. Notons que la fonction histogramme satisfait à l'équivalent discret de la contrainte de normalisation (5.3):

$$\sum_{m=1}^{M} h_m b = 1 \ . \tag{5.8}$$

De plus, dans la limite $N \to \infty$, on s'attendrait à ce que $N_m = N/M$ $(= Nb)$ valeurs tombent dans chaque "compartiment", donc on aura

$$\lim_{N \to \infty} h_m(r_n; b) = 1 \ , \qquad m = 1, ..., M \ , \tag{5.9}$$

démontrant bel et bien que la fonction histogramme est l'équivalent discret de notre FDP uniforme (5.1).

La Figure 5.3 montre des fonctions histogrammes construites à partir de séquences de nombres aléatoires produits par notre générateur uniforme, pour diverses valeurs de N. Dans tous les cas on a compartimenté ici l'intervalle $[0, 1]$ en $M = 20$ intervalles d'échantillonnage. On voit immédiatement que plus N est grand, plus les déviations par rapport à la valeur attendue $h_m = 1$ sont petites. Changer l'initialisation du générateur de nombre aléatoires changerait le *détail* de ces fonctions histogramme (i.e., quels h_m se retrouve au dessus ou au dessous de la valeur attendue), mais pas leur allure générale.

Essayons de quantifier un peu ce dernier énoncé. L'importance des déviations par rapport à la valeur attendue est souvent mesurée par la **déviation quadratique moyenne**, définie comme:

$$\sigma^2 = \frac{1}{M} \sum_{m=1}^{M} (h_m - \langle h \rangle)^2 \ , \tag{5.10}$$

où la valeur attendue $\langle h \rangle$ correspond ici à la moyenne des h_m:

$$\langle h \rangle = \frac{1}{M} \sum_{m=1}^{M} h_m \ . \tag{5.11}$$

La quantité σ comme telle est habituellement appelée "déviation RMS", pour "Root-Mean-Squared". Sa variation avec N est illustrée à la Figure 5.5, toujours pour nos nombres aléatoires uniformes. On remarque que la déviation RMS y varie en $1/\sqrt{N}$. Cette situation n'est pas particulière à une FDP uniforme, mais caractérise en fait tous les processus statistiques sans effets de mémoire, dans la mesure où, comme quand on roule un dé ou on tire à pile ou face, chaque mesure tirée de la distribution est complètement indépendante des précédentes[1].

[1] Comment réconciliez-vous cet énoncé avec le fait que notre générateur de nombre aléatoire est complètement déterministe, dans le sens qu'une fois initialisé, la valeur du nombre aléatoire r_n est complètement déterminée par celle du nombre r_{n-1}?

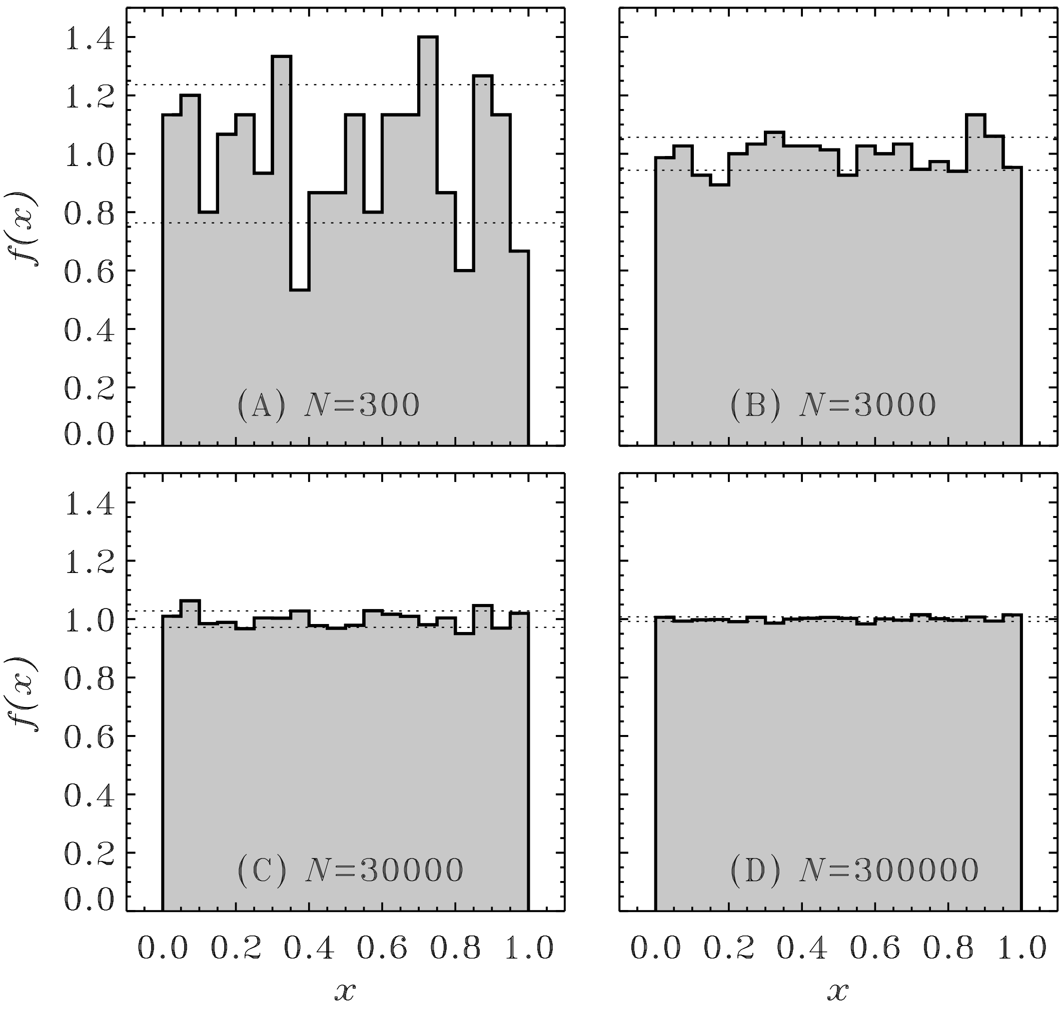


Figure 5.3: Fonction de distribution des valeurs de séquences de N nombres aléatoires $r_n \in [0, 1]$, pour quatres valeurs de N croissant par des facteurs 10 successifs. Les traits pointillés horizontaux indiquent l'intervalle correspondant à ± 1 déviations rms. Dans tous les cas la valeur attendue est $f(x) = 1$.

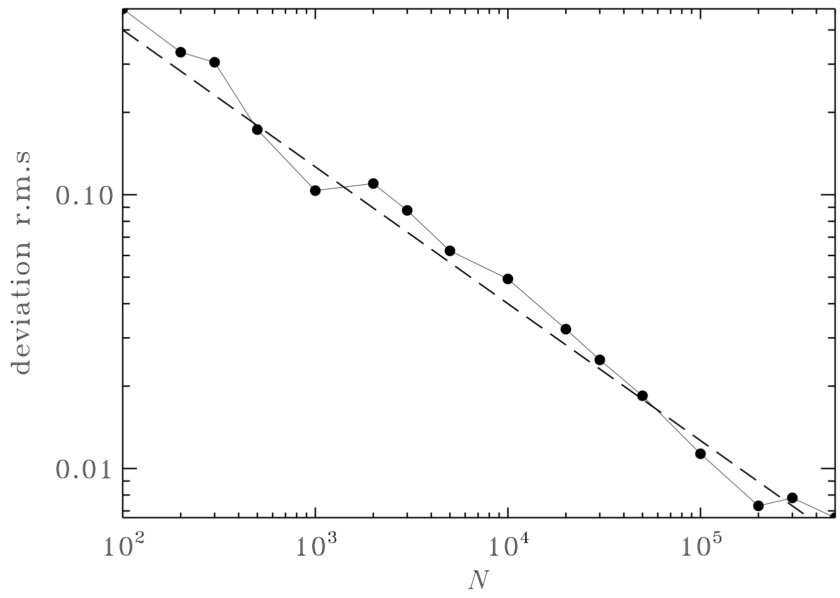


Figure 5.4: Variation de l'écart rms (éq. (5.10)) sur la valeur attendue $f(x) = 1$, en fonction du du nombre N de candidats r produits. La droite en tirets indique une décroissance en $1/\sqrt{N}$.

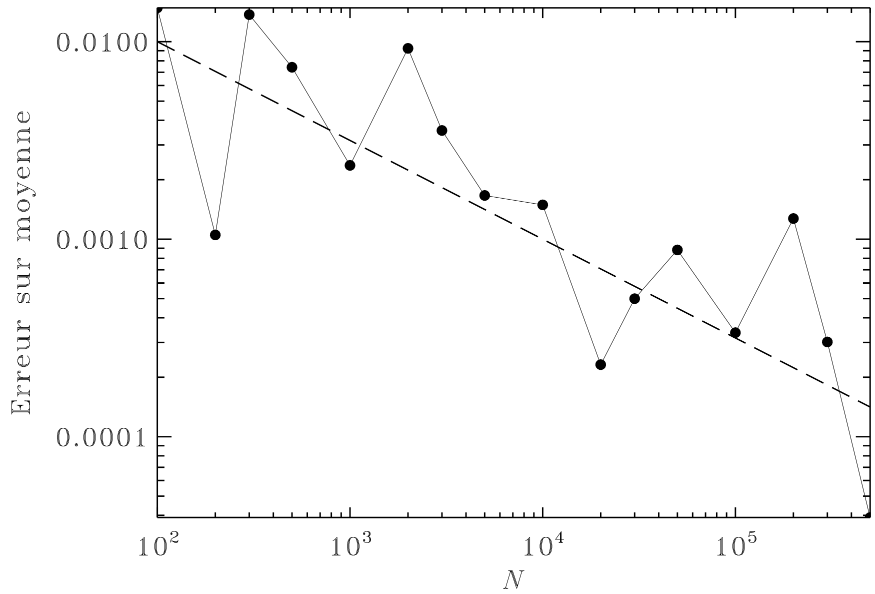


Figure 5.5: Variation de l'erreur sur la moyenne de la distribution en fonction du nombre N de candidats r produits. L'erreur est définie ici selon l'éq. (5.14). La droite en tirets indique une décroissance en $1/\sqrt{N}$.

Une autre mesure souvent intéressante (et importante) de la grandeur des fluctuation dans les distributions construites pour un N fini est la valeur moyenne de la distribution même:

$$\langle r\rangle = \frac{1}{N}\sum_{n=1}^{N} r_n \ . \tag{5.12}$$

Pour notre distribution uniforme dans $[0,1]$, on devrait évidemment avoir

$$\lim_{N\to\infty}\langle r\rangle = \frac{1}{2} \ , \tag{5.13}$$

Donc on peut se définir ici une mesure d'erreur comme:

$$\varepsilon(N) = \left|\langle r\rangle - \frac{1}{2}\right| \ . \tag{5.14}$$

La variation de cette quantité avec N est portée en graphique à la Figure 5.5; encore une fois, on constate que l'erreur décroit *grosso modo* comme $1/\sqrt{N}$.

5.3 Distributions non-uniformes

Dans bien des situations il est nécessaire de produire une séquence de nombres aléatoires extraits d'une distribution qui ne soit pas uniforme. Il existe des techniques générales permettant de transformer n'importe quelle FDP en une autre; nous n'entrerons pas là-dedans ici (mais voir la bibliographie en fin de chapitre), et nous nous limiterons plutôt à donner quelques "recettes" pour produire des distributions souvent utilisées en simulations physiques. Ceux ou celles qui désirent approfondir ce genre truc, ou en examiner les bases théoriques, trouveront de quoi s'occuper dans la bibliographie en fin de chapitre.

La **méthode de transformation** est basée sur l'idée que la distribution *cumulative* associée à une FDP $f(x)$ non-uniforme peut être interprétée comme un nombre aléatoire r extrait d'une distribution uniforme dans $[0,1]$; L'idée générale est illustrée à la Figure 5.6, qui montre comment une distribution uniforme en y peut, via une fonction non-linéaire (ici une exponentielle décroissante), produire une distribution non-uniforme en x. Mathématiquement cette idée s'exprime selon la relation:

$$\int_{-\infty}^{x} f(x')\mathrm{d}x' = r \ , \qquad r\in[0,1] \ . \tag{5.15}$$

Si $f(x)$ peut être intégré analytiquement, ceci offre une manière simple et efficace de produire des nombres aléatoires extraits de distributions statistiques non-uniforme. Voyons comment ca marche avec un exemple simple mais utile, soit la distribution exponentielle.

5.3.1 Distributions exponentielles

Dans plusieurs situations physiques (et au Labo 8...), il est nécessaire de simuler des séquences d'événements se produisant de manière aléatoire dans le temps et/ou l'espace, sans effet de mémoire (i.e., l'événement b est complètement indépendant de l'événement a l'ayant précédé). Un tel système obéit à la statistique dite de Poisson, et on peut montrer que la distribution des temps d'attentes (ou distances) entre deux événements successifs prend une forme exponentielle:

$$f(x) = \begin{cases} \lambda^{-1}\exp(-x/\lambda) \ , & 0\leq x\leq\infty, \\ 0 & x<0 \end{cases} \ , \tag{5.16}$$

où λ est le facteur d'échelle contrôlant la décroissance de la distribution avec x. L'équation (5.15) donne alors:

$$\lambda^{-1}\int_{-\infty}^{x}\exp(-x'/\lambda)\mathrm{d}x' = \lambda^{-1}\int_{0}^{x}\exp(-x'/\lambda)\mathrm{d}x' =$$

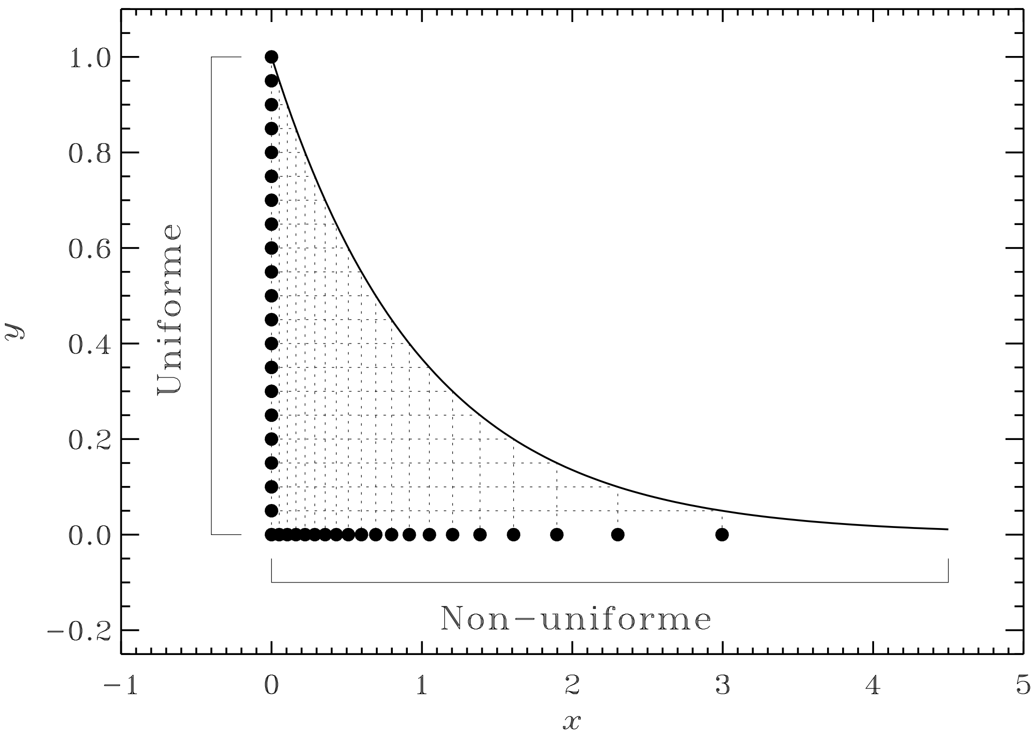


Figure 5.6: Illustration de la production d'une distribution non-uniforme en x à partir d'une distribution uniforme en y, via une transformation nonlinéaire (voir texte). Si les y étaient des nombres aléatoires $\in [0,1]$ produits par un générateur du genre `ran0` alors la distribution des x correspondants serait ici plus concentrée aux petites valeurs de x.

$$= [-\exp(-x'/\lambda)]_0^x = 1 - \exp(-x/\lambda) = r \ , \qquad r \in [0,1] \ , \tag{5.17}$$

d'où on déduirait

$$x = -\lambda \ln(1-r), \qquad r \in [0,1] \ , \qquad x \in [0,\infty] \ . \tag{5.18}$$

mais comme la variable aléatoire $1-r$ se distribue dans l'intervalle $[1,0]$ identiquement à la manière dont r se distribue dans $[0,1]$, on peut tout aussi bien écrire:

$$x = -\lambda \ln r \ , \qquad r \in [0,1] \ , \qquad x \in [0,\infty] \ . \tag{5.19}$$

5.3.2 Distributions gaussiennes

La distribution dite gaussienne (éq. (5.2) ci-dessus) est fort utile pour bien des applications, en particulier pour les analyses ou simulations des erreurs expérimentales. La Figure 5.8 en montre un exemple (traits pleins). Le paramètre clef ici est la variance σ, qui contrôle l'étendue en x de la distribution (la largeur à mi-hauteur $= 1.176\sigma$). Si des nombres aléatoires sont extraits de cette distributions, 68.3% seront situés à l'intérieur de $\pm\sigma$, mais il demeure probable de produire occasionnellement des nombres de taille passablement plus élevée. Cependant, l'intégrale d'une gaussienne ne peut être calculée analytiquement via l'éq. (5.17). Comment produire une séquence de nombre aléatoires distribués de manière Gaussienne? Il existe plusieurs techniques, mais nous ne considérerons ici que celle basée sur la transformation dite de Box-Muller: ayant

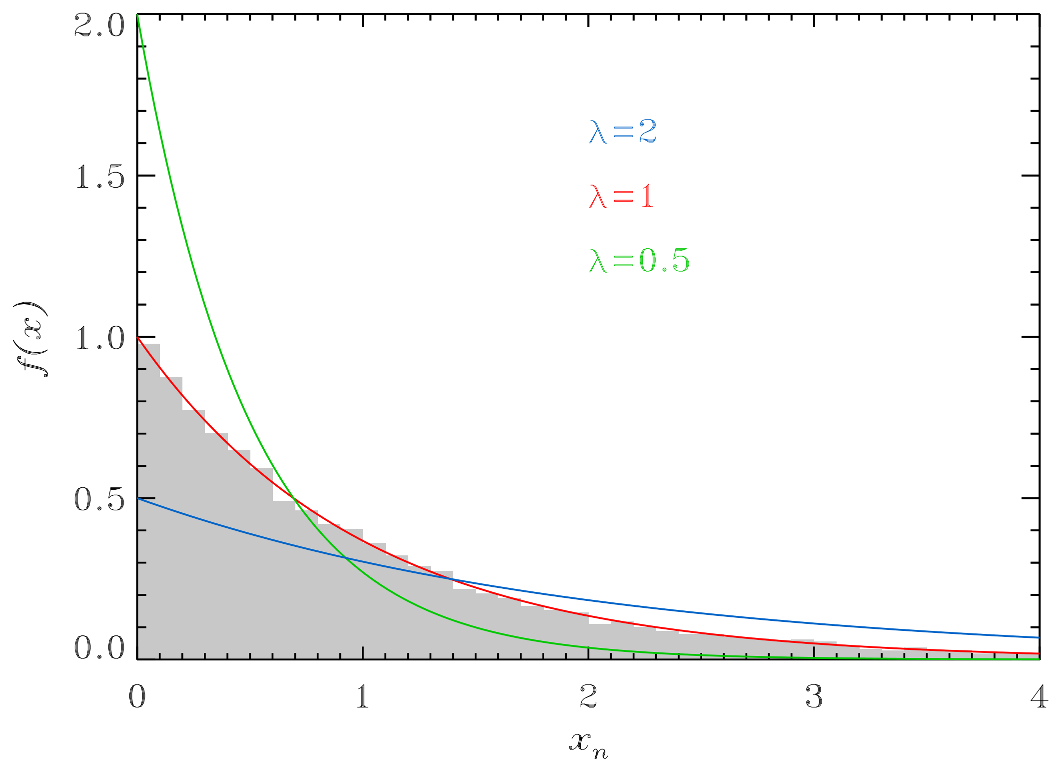


Figure 5.7: Fonction de distribution de probabilité de forme exponentielle, telle que décrite par l'éq. (5.16), pour trois valeurs du facteur d'échelle (traits plein). L'histogramme est la distribution de 10^4 nombres aléatoires produits à l'aide de l'éq. (5.19) avec $\lambda = 1$.

en main deux nombre aléatoires r_1, r_2 extraits d'une distribution uniforme dans l'intervalle $[0, 1]$, on peut produire deux nombres aléatoires g_1, g_2 distribués de manière Gaussienne via les relations:

$$g_1 = \sqrt{-2 \ln r_1} \cos(2\pi r_2) \ , \tag{5.20}$$

$$g_2 = \sqrt{-2 \ln r_1} \sin(2\pi r_2) \ , \tag{5.21}$$

où $g_1, g_2 \in [-\infty, \infty]$. La Figure 5.8 en montre un exemple de la distribution produite ainsi, pour 10^4 nombres aléatoires calculés utilisant les relations ci-dessus. Cette procédure fonctionne, mais elle est loin d'êre optimale au niveau du temps de calcul requis; on peut en fait faire beaucoup mieux, mais je vous laisse ça comme "lecture supplémentaire".

5.4 Évaluation d'intégrales par Monte Carlo

Appliquons maintenant l'approche Monte Carlo au calcul d'une intégrale définie d'une fonction $f(x)$ d'une variable; nous choisirons de nouveau un polynôme quartique:

$$f(x) = 1 + x^4 \ , \tag{5.22}$$

et, comme intégrale à évaluer:

$$I = \int_0^1 f(x) \mathrm{d}x \qquad \left(= \left[x + \frac{x^5}{5} \right]_0^1 = \frac{6}{5} \right) \ . \tag{5.23}$$

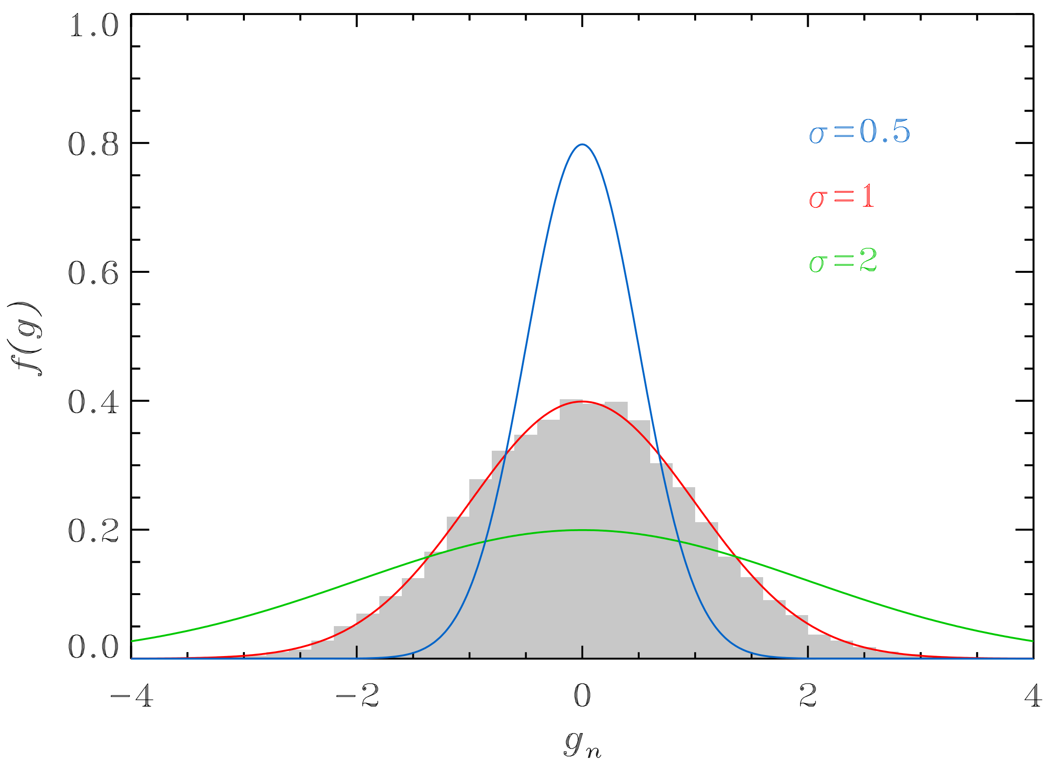


Figure 5.8: Fonction de distribution de probabilité de forme Gaussienne, telle que décrite par l'éq. (5.2), pour trois valeurs de la largeur à mi-hauteur (traits plein). L'histogramme est la distribution de 10^4 nombres aléatoires produits à l'aide des éqs. (5.20)—(5.21).

Si on utilise la méthode du trapèze (§2.4) avec un maillage constitué de N points équidistants en x, on aura:

$$I = \sum_{n=1}^{N-1} \frac{1}{2}(f_{n+1} + f_n) \underbrace{(x_{n+1} - x_n)}_{\equiv h}$$

$$= \left[\frac{f_1 + f_2}{2} + \frac{f_2 + f_3}{2} + \frac{f_3 + f_4}{2} + ... + \frac{f_{N-2} + f_{N-1}}{2} + \frac{f_{N-1} + f_N}{2} \right] \times h$$

$$= \left(\frac{f_1 + f_N}{2} + \sum_{n=2}^{N-1} f_n \right) \times h \ , \tag{5.24}$$

où ici le pas de maille $h = 1/(N-1)$. Notons, et retenons, que le nombre d'évaluations de f requis ici est égal à N. L'expression ci-dessus ressemble dangereusement au calcul d'une valeur moyenne de $f(x)$ échantillonnée uniformément sur l'ensemble de l'intervalle, et c'est bien là une interprétation habituelle de l'intégrale comme une aire sous la courbe (valeur moyenne de f dans l'intervalle fois la longueur de l'intervalle, ici un).

Il y aurait d'autres façons de calculer cette valeur moyenne. Plutôt que d'échantillonner uniformément en x, on aurait pu choisir d'échantillonner f à N points positionnés aléatoirement (mais de manière statistiquement uniforme) en x dans l'intervalle $[0, 1]$. Vous commencez j'espère à percevoir le lien avec l'exemple de l'étang... La valeur de l'intégrale serait alors

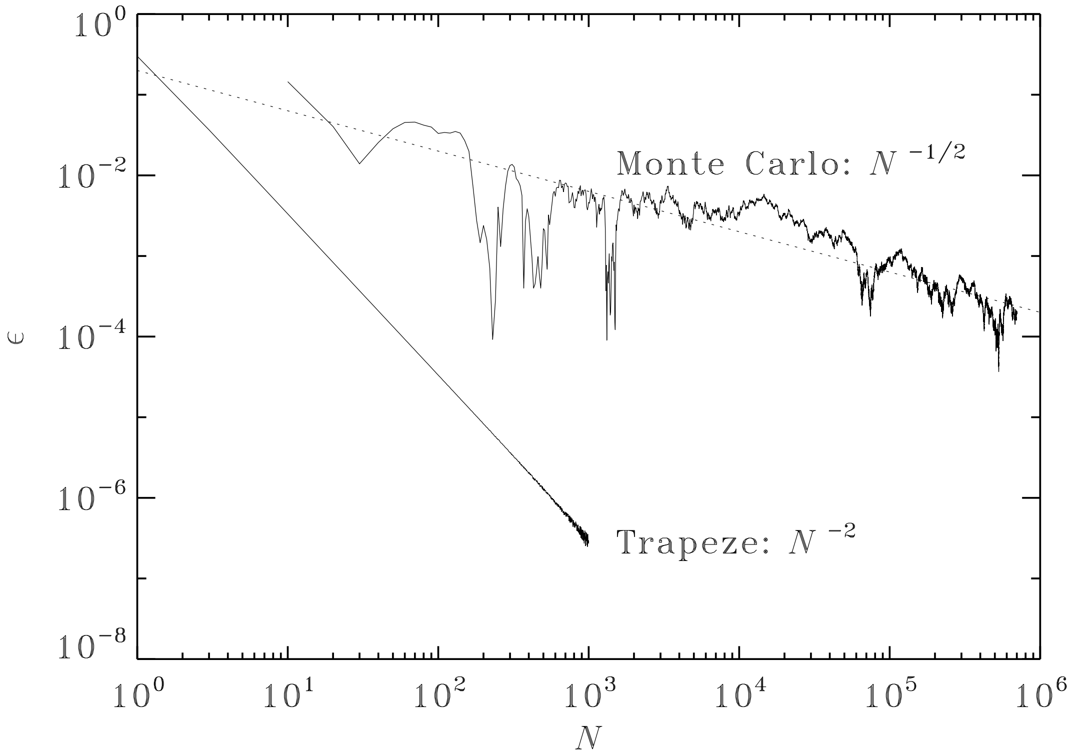


Figure 5.9: Diminution de l'erreur dans le calcul de l'intégrale définie par les éqs. (5.22) et (5.23) en fonction du nombre N d'évaluation de $f(x)$, pour la méthode du trapèze et l'intégration Monte Carlo, tel qu'indiqué. La droite pointillée correspond à une variation en $N^{-1/2}$.

donnée par

$$I = \frac{1}{N}\sum_{n=1}^{N} f(r_n) , \qquad r_n \in [0,1] , \tag{5.25}$$

où r_n est un nombre aléatoire extrait d'une distribution uniforme couvrant l'intervalle $[0,1]$. Pourquoi pas?

La Figure 5.9 montre la variation de l'erreur, définie ici comme

$$\varepsilon(N) = |1.2 - I(N)| , \tag{5.26}$$

pour l'évaluation de notre intégrale par la méthode du trapèze (éq. (5.24)), et par la méthode Monte Carlo (éq. (5.25)). La première converge en $1/N^2$, comme on s'y attendrait d'une méthode $O(h^2)$; la seconde converge de manière beaucoup plus irrégulière, mais en moyenne comme un pitoyable $1/\sqrt{N}$ (droite pointillée). Ici, la méthode Monte Carlo requiert en moyenne $\sim 10^5$ évaluations de f pour en arriver à $\varepsilon = 10^{-3}$, tandis que la méthode du trapèze atteint ce niveau d'erreur en seulement 17 évaluations de f. Suis-je en train de vous faire perdre votre temps avec toutes ces histoires de Monte Carlo?

Non. C'est certainement le cas qu'en une dimension spatiale, le Monte Carlo n'est jamais compétitif. Mais en plusieurs dimensions spatiales, c'est une toute autre histoire. On peut montrer que l'équivalent à D dimensions spatiales de la méthode du trapèze, pour un échantillonage uniforme avec pas $h = 1/N$ dans chaque dimension, converge comme $1/N^{2/D}$ (retournez voir la Figure 5.1 en pensez-y à tête reposée.). Cependant, l'intégration Monte Carlo converge toujours comme $1/\sqrt{N}$, et ce quelle que soit la dimension D !! Le taux de convergence des deux

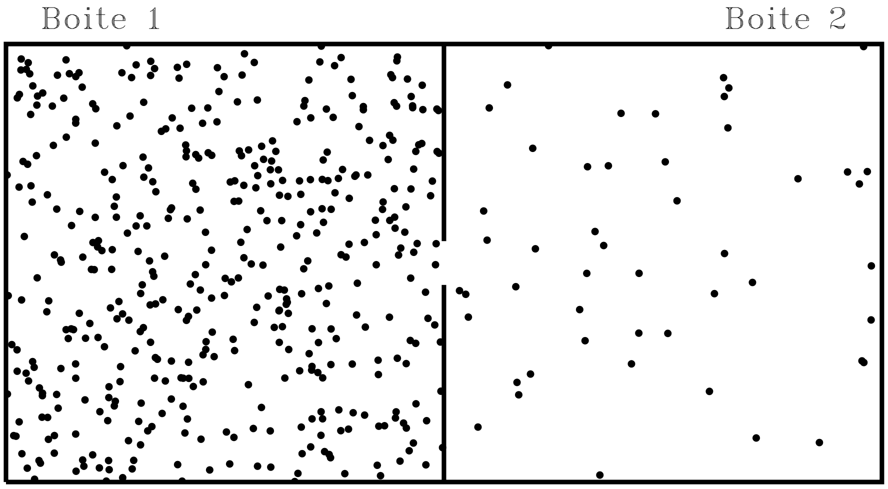


Figure 5.10: Deux boites communiquant par un petit trou contiennent un total de N particules, mais l'une (ici celle de gauche) en contient un nombre N_1 plus grand que l'autre, soit $N_2 < N_1$. Les collisions contre les parois et interparticules feront que le nombre de particules traversant de la gauche vers la droite sera plus grand que dans le sens inverse, et ce tant et aussi longtemps que $N_1 > N_2$. La solution d'équilibre est ici $N_1 = N_2 = N/2$ (voir texte).

méthodes devient donc le même pour une intégrale en 4 dimensions, et aucune méthode de type Newton-Cotes ne demeure le moindrement compétitive en plus de 10 dimensions spatiales.

Vous pourriez penser qu'une intégrale en plus de trois dimensions, ca n'a pas d'allure de toute façon, mais détrompez-vous, vous allez en rencontrer en physique statistique qui montent beaucoup plus haut; par exemple, l'espace de phase d'un pendule à tige rigide pouvant osciller librement dans toutes les directions est de dimension 4; et celui des deux atomes d'hydrogène et de l'atome d'oxygène formant une molécule de H_2O est à 18 dimensions (3 coordonnées spatiales et trois composantes de la vitesse par atome).

5.5 L'approche à l'équilibre

Nous allons maintenant étudier un problème classique dit **d'approche à l'équilibre**, dans le contexte du problème de mélange, illustré sur la Figure 5.10. Deux boites contigues communiquent via un petit trou percé dans leur paroi commune. Chaque boite contient un nombre N_1, N_2 de particules genre boules de billiard. Ces particules vont continuellement se frapper entre elles, ou rebondir sur les parois. De temps en temps, ces collisions vont faire qu'une particule passera de la boite 1 à la boite 2, ou l'inverse. Nous tenterons maintenant de répondre aux trois questions suivantes:

1. Existe-t-il un **état d'équilibre** pour lequel le nombre de particules dans chaque boite demeure constant avec le temps;
2. Si oui, quel est cet état;
3. Toujours si oui, quel est le temps requis pour atteindre cet état d'équilibre à partir d'une condition initiale arbitraire.

Intuitivement, on peut s'attendre à ce que si l'on attend suffisamment longtemps (ici dans le sens d'un intervalle de temps beaucoup plus grand que le temps entre deux collisions successives), chaque particule a ultimement la même probabilité de traverser le trou. Le nombre total de particules δN_{12} traversant de la boite 1 à la boite 2 durant un intervalle de temps Δt devrait donc être donné par quelque chose du genre:

$$\delta N_{12} = \alpha N_1 \Delta t \ , \tag{5.27}$$

où α est la probabilité de passage par particule par unité de temps. C'est une quantité qui dépend de la taille du trou, de la vitesse des particules, de leur taux de collision, *mais pas de leur nombre*, cette dernière dépendance étant déjà explicitée au membre de droite. Et tintin évidemment pour le nombre de particules passant de 2 à 1:

$$\delta N_{21} = \alpha N_2 \Delta t \ , \tag{5.28}$$

avec la même valeur de α qu'à l'éq. (5.27). Pour avoir une situation stationnaire, il faudra qu'il n'y ait plus de transfer *net* de particules d'une boite à l'autre (sinon N_1 et N_2 varieront en conséquence, et donc nous n'avons plus une situation stationnaire). Exiger que $\delta N_{12} = \delta N_{21}$ nous donne alors immédiatement la condition d'équilibre:

$$N_1 = N_2 \ , \qquad \text{[equilibre]} \ . \tag{5.29}$$

Ce résultat concorde avec l'intuition. Supposons que les deux boites sont en fait deux classes remplies d'étudiant(e)s, et que les particules sont en fait des macromolécules nausébondes d'aftershave Irish Brut 45 Magnum dont un de vos petits camarades s'est trop généreusement aspergé le matin même, ayant manqué de temps pour la douche. L'expérience montre que si l'on attend assez longtemps, ca puera uniformément dans les deux classes, en d'autres mots le système a atteint un état d'équilibre où le nombre de molécules puantes par unité de volume est le même partout. Voyons maintenant à quelle vitesse cet état d'équilibre est atteint.

5.5.1 Formulation en équation différentielle

Considérant notre définition de α comme étant la probabilité qu'une particule de la boite passe à la boite 2 par intervalle de temps Δt, le nombre de particules dans les boites 1 et 2 au temps $t + \Delta t$ doit être donnée par quelque chose comme:

$$N_1(t+\Delta t) = N_1(t) + \alpha \Delta t N_2(t) - \alpha \Delta t N_1(t) \ , \tag{5.30}$$

$$N_2(t+\Delta t) = N_2(t) + \alpha \Delta t N_1(t) - \alpha \Delta t N_2(t) \ . \tag{5.31}$$

Le premier terme au membre de droite est le nombre de particules déjà présentes dans chaque boite au temps t, le second terme correspond aux particules entrant dans la boite, et le troisième aux particules sortant de la boite. Je vous laisse vérifier que la forme de ces expressions garantit que le nombre total de particules est une quantité conservée, i.e., $N_1(t) + N_2(t) = N$. Un peu d'algèbre simple permet de ramène ces expression à la forme suivante:

$$\frac{N_1(t+\Delta t) - N_1(t)}{\Delta t} = \alpha(N_2(t) - N_1(t)) \ , \tag{5.32}$$

$$\frac{N_2(t+\Delta t) - N_2(t)}{\Delta t} = \alpha(N_1(t) - N_2(t)) \ . \tag{5.33}$$

Examinez bien les membres de gauche de ces deux expressions. Vous réaliserez j'espère qu'ils ressemblent fort à une différence finie d'ordre $O(\Delta t)$ pour les dérivées temporelles de N_1 et N_2. On peut fonc écrire:

$$\frac{\mathrm{d}N_1}{\mathrm{d}t} = \alpha(N_2 - N_1) \ , \tag{5.34}$$

$$\frac{\mathrm{d}N_2}{\mathrm{d}t} = \alpha(N_1 - N_2) \ . \tag{5.35}$$

Il s'avérera pratique d'exprimer le nombre de particules dans chaque boite en terme de la fraction du nombre total N de particules:

$$f_1 = \frac{N_1}{N} \ , \qquad f_2 = \frac{N_2}{N} \ . \tag{5.36}$$

Notre contrainte de conservation $N_1 + N_2 = N$ devient simplement

$$f_1 + f_2 = 1 \ , \tag{5.37}$$

ce qui permet d'exprimer les éq. (5.34)–(5.35) sous la forme:

$$\frac{\mathrm{d}f_1}{\mathrm{d}t} = \alpha(1 - 2f_1) \ , \tag{5.38}$$

$$\frac{\mathrm{d}f_2}{\mathrm{d}t} = \alpha(1 - 2f_2) \ . \tag{5.39}$$

Remarquez que l'utilisation de la contrainte de conservation a effectivement découplé nos deux équations différentielles, ce qui fait que nous n'avons plus qu'à en solutionner une, et la solution de l'autre s'obtient trivialement via l'éq. (5.37). Remarquez également que si l'on pose $f_1 = f_2 = 0.5$, on obtient bien $\mathrm{d}f_1/\mathrm{d}r = \mathrm{d}f_2/\mathrm{d}t = 0$, comme on s'y attend pour une solution d'équilibre.

L'équation (5.38) est une équation différentielle ordinaire linéaire mais **inhomogène**, du à la présence d'un terme constant α au membre de droite. Un simple changement de variable permet cependant de convertir ça en une équation différentielle homogène qui se solutionne facilement. Introduisons une nouvelle fonction $\xi(t)$ définie selon

$$\xi(t) = f_1(t) - \frac{1}{2} \ ; \tag{5.40}$$

comme ξ et f_1 sont reliées l'une à l'autre par une relation linéaire, on aura

$$\frac{\mathrm{d}\xi}{\mathrm{d}t} = \frac{\mathrm{d}f_1}{\mathrm{d}t} \ , \tag{5.41}$$

ce qui permet de réexprimer l'éq. (5.38) sous la forme

$$\frac{\mathrm{d}\xi}{\mathrm{d}t} = -2\alpha\xi \ , \tag{5.42}$$

qui est l'équation homogène promise. Ceci a évidemment comme solution:

$$\xi(t) = \xi_0 \exp\left(-2\alpha t\right) \ , \tag{5.43}$$

où ξ_0 correspond à la valeur de ξ à $t = 0$. Si l'on choisit comme condition initiale qu'à $t = 0$ toutes les particules sont dans la boite 1, alors $f_1(t = 0) = 1$, et on aura $\xi_0 = 1/2$. On utilise finalement l'éq. (5.40) pour exprimer ce résultat en terme de notre fonction primaire f_1:

$$f_1(t) = \frac{1}{2}\left(1 + \exp\left(-2\alpha t\right)\right) \ . \tag{5.44}$$

La solution asymptotique est bien

$$\lim_{t\to\infty} f_1(t) = \frac{1}{2} \ , \tag{5.45}$$

soit une répartition égale des particules dans les boites 1 et 2, atteinte exponentiellement avec une constante de temps $\tau = 1/(2\alpha)$; se rappelant que α est une mesure de la taille du trou, on voit bien que plus le trou est petit, plus l'approche à l'équilibre est lente, ce qui est tout à fait logique puisque les particules ont plus de difficulté à passer d'une boite à l'autre.

5.5.2 Formulation Monte Carlo

Maintenant, utilisons une approche Monte Carlo à notre problème d'approche à l'équilibre. Nous avons déjà fait tout le travail théorique requis en écrivant les équations (5.30)–(5.31). Le bout de code C présenté à la Figure 5.11 est tout ce dont vous avez besoin. Notez bien comment on "décide" de faire traverser ou non une particule, via l'instruction:

```
if ( 1.*rand()/RAND_MAX <= alpha ) { de1a2=de1a2+1 ; }
```

L'expression `1.*rand()/RAND_MAX`, introduite à la §5.1, produit un nombre aléatoire uniformément distribué dans l'intervalle $[0, 1]$. À chaque appel, si le nombre aléatoire produit est *inférieur* à la probabilité α (avec $0 \leq \alpha \leq 1$), alors seulement la particule passe, ou, plus précisément, le compteur comptabilisant le nombre de particules qui traversent est incrémenté de un. Cette opération est répétée pour chaque particule contenue dans chaque boite, via les deux boucles inconditionnelles `for`. Vous pouvez facilement voir que poser $\alpha = 0$ ferait qu'aucune particule ne traverserait (test ci-dessus jamais satisfait), et $\alpha = 1$ au contraire conduirait à un passage de toutes les particules en un pas de temps, puisque le test probabiliste serait alors toujours satisfait quelle que soit la valeur produite par `ran0`.

La Figure 5.12 montre le résultat de trois intégrations Monte Carlo de l'approche à l'équilibre, pour des systèmes de 10^3, 10^4, et 10^5 particules, toutes initialement plaçées dans la boite 1. Les entiers N_1 et N_2 ont été convertis en fraction f_1 et f_2 selon l'éq. (5.36), mais ici la nature discrète de la variable est bien évidente, en particulier pour la solution à $N = 10^3$ (en rouge).

La forme exponentielle de l'approche à l'équilibre est facilement visible ici, mais on remarque aussi des déviations irrégulières par rapport à une exponentielle "pure". Le détail de ces variations est différent pour une initialisation différente du générateur de nombre aléatoire, mais elles sont toujours présentes. On remarque également sur la Figure 5.12 que plus N est grand, plus ces déviations sont petites. La Figure 5.13 montre la variation de la déviation RMS évaluée à quatre temps différents durant l'évolution, à partir d'un ensemble de 50 simulations statistiquement distinctes (i.e., utilisant une initialisation différente du générateur de nombres aléatoires). Ici encore, la fluctuation par rapport à la valeur moyenne décroit selon $1/\sqrt{N}$, à n'importe quel temps durant l'approche à l'équilibre.

5.6 Réalisme physique et irréversibilité

Nous allons clore ce chapitre avec quelques observations et questions qui vont, je l'espère, vous faire réfléchir un peu sur ce qu'on pourrait appeler la philosophie de la modélisation, et peut-être sur la nature de la réalité physique (Mamma mia...).

5.6.1 Comment choisir les alternatives en modélisation?

Nous avons donc développé deux approches de modélisation fondamentalement distinctes à notre problème d'échange de particules entre deux boites: une approche déterministe basée sur la solution d'une équation différentielle, et une approche stochastique de type Monte Carlo. Laquelle est la "meilleure" ici? La solution exponentielle décrite par l'éq. (5.44) est "propre" et mathématiquement concise; on serait tenté de lui donner préférence. Mais réfléchissez-y un peu plus profondément; le système physique étudié ici est formé d'un nombre fini (quoique possiblement très grand) de particules. L'approche déterministe de la §5.5.1 convertit ceci en un problème continu pour les fractions f_1 et f_2, en remplaçant les membres de gauche des équations (5.32)–(5.33) par des dérivées temporelles, qui à strictement parler ne sont définies que si N_1 et N_2 sont des variables réelles, pas des entiers! En passant des éqs. (5.32)–(5.33) aux éqs. (5.34)–(5.35), l'approche déterministe introduit une approximation qui n'est pas faite par l'approche Monte Carlo, qui travaille directement à partir des équations (5.30)–(5.31). Quelle est l'approche la plus physiquement raisonnable ici? Fondamentalement, ce serait le Monte Carlo! Cependant, si N est vraiment très grand (genre, le nombre de macromolécules d'Irish

```
#include <stdio.h>
#include <stdlib.h>
/* Simule melange de particules entre deux boites contigues       */
int main(void)
{
/* Declarations  ------------------------------------------------ */
  long niter=200 ;       /* Nombre d'iterations "temporelles"     */
  long n=1000 ;          /* Nombre total de particules            */
  long n1, n2, de1a2, de2a1, iter, k ;
  float alpha=0.01 ;     /* Prob. de traverser a chaque iteration */
/* Executable --------------------------------------------------- */
  n1=n ;               /* Toutes les particules debutent a gauche */
  n2=0 ;
  for (iter=1 ; iter<=niter ; iter++ )  /* Boucle "temporelle"    */
    {
      de1a2=0 ;        /* compteur part. traversant de 1 a 2      */
      de2a1=0 ;        /* compteur part. traversant de 2 a 1      */
      for (k=1 ; k<= n1 ; k++)      /* On teste particules dans 1 */
        {
          if ( 1.*rand()/RAND_MAX <= alpha ) { de1a2=de1a2+1 ; }
        }
      for (k=1 ; k<= n2 ; k++)      /* On teste particules dans 2 */
        {
          if ( 1.*rand()/RAND_MAX <= alpha ) { de2a1=de2a1+1 ; }
        }
      n1 = n1-de1a2+de2a1 ;     /* Nouveau N de particules dans 1 */
      n2 = n2+de1a2-de2a1 ;     /* Nouveau N de particules dans 2 */
      printf ("Iter = %d ,N_1= %d ,N_2= %d\n",iter,n1,n2) ;
    }
}
```

Figure 5.11: Code C pour le calcul de mélange entre deux boites, par simulation Monte Carlo. Notez que Le “transfert” de particules d'une boite à l'autre se fait seulement une fois que toutes les particules ont été testées, les nombres à transférer de la boite 1 vers la 2 et de la 2 vers la 1 étant accumulées dans des compteurs (ici les variables `de1a2` et `de2a1`), qui doivent être réinitialisées à zéro au début de chaque itération de la boucle extérieure pseudo-temporelle. Si on transférait immédiatement les particules quand le test probabiliste était satisfait (`1.*rand()/RAND_MAX <= alpha`), alors l'ordre dans lequel on teste les boites (ici la 1 avant la 2) aurait une influence sur l'évolution. Notez encore une fois la prémultiplication de la fonction `rand()` par le réel `1.`, essentielle afin d'éviter que la division de l'entier renvoyé par `rand()` divisé par `RAND_MAX` ne donne toujours zéro.

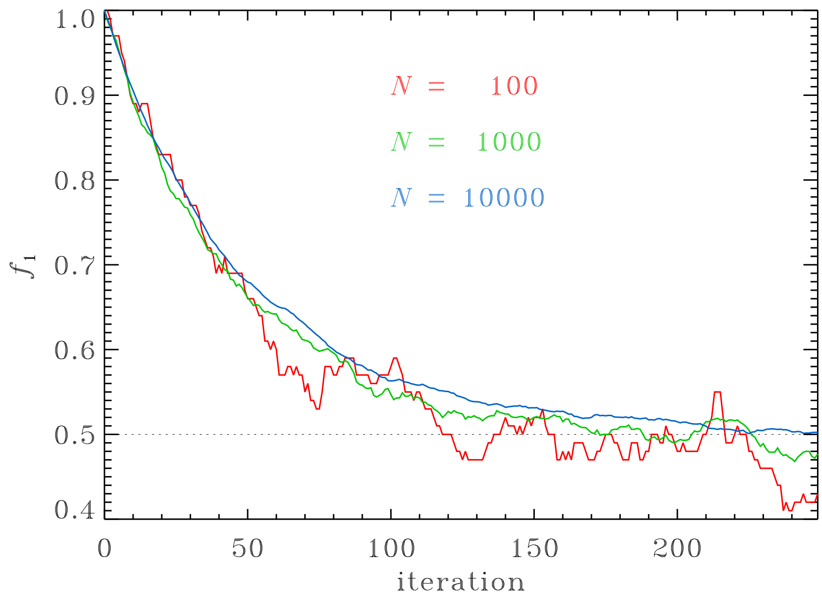


Figure 5.12: Variation temporelle de la fraction de particules dans la boite 1, pour un nombre total de particules $N = 10^2$ (rouge), 10^3 (vert) et 10^4 (bleu). La variation correspondante de f_2 est obtenue via la contrainte de conservation $f_1 + f_2 = 1$, et correspond à une réflexion par rapport à la droite $f_1 = 0.5$. La décroissance est exponentielle dans les trois cas, mais les fluctuations diminuent rapidement à mesure que N augmente (voir texte).

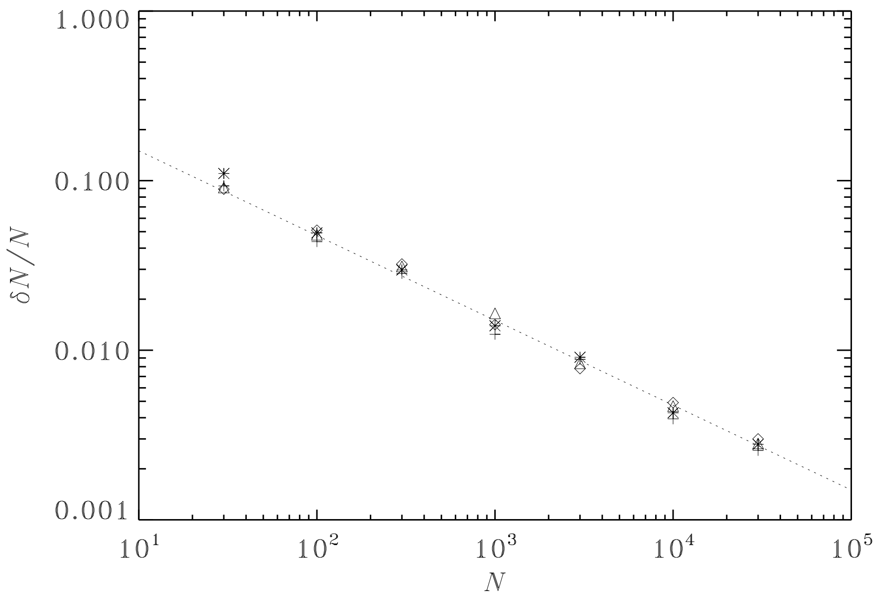


Figure 5.13: Variation de l'écart relatif moyen en fonction du nombre total de particules N, associé à des ensembles de 50 solutions statistiquement distinctes au problème de l'approche à l'équilibre. Les différents symboles correspondent à des écarts mesurés à des temps différents, soit $t = 50$ ($*$), $t = 100$ ($+$), $t = 150$ ($\triangle$) et $t = 200$ ($\diamond$), avec t mesuré ici comme le nombre d'itérations du processus Monte Carlo. Le trait pointillé correpond à une décroissance en $1/\sqrt{N}$.

Magnum 45 dans une salle de classe), l'approche Monte Carlo peut devenir très lente, pour produire en bout de ligne un résultat qui ne pourra être distingué de l'exponentielle décroissante au niveau de précision de votre ordinateur.

Si vous avez suivi le raisonnement ici, vous aurez anticipé que l'approche optimale consisterait à utiliser Monte Carlo quand N est petit, et l'approche déterministe quand N est grand. Mais, comment alors choisir la valeur de N à laquelle on change d'approche? Il n'y a absolument pas de réponse facile à cette question; la réponse dépend du type d'information que vous cherchez à extraire de votre modèle. Notez cependant qu'ici l'approche Monte Carlo fournit "naturellement", en prime, une mesure statistique des fluctuations pouvant être attendues dans le système. C'est là une information souvent précieuse.

5.6.2 L'irréversibilité

L'intuition physique, et nos modélisations mathématique et statistique du problème, indiquent toutes que dans la situation illustrée à la Figure 5.10 le transfert net de particules se fera de la boite 1 vers la boite 2, donc que si l'on photographiait ce système à un temps ultérieur le nombre de particules dans la boite 1 aura diminué, et celui dans la boite 2 aura augmenté par la même quantité. Vous seriez fort surpris si la photographie révélait que la boite 2 s'était vidée de ses quelques particules au profit de la boite 1, et vous seriez fort tenté de conclure qu'un petit comique a inversé l'ordre des photos.

Un phénomène comme celui que nous avons modélisé ici, pour lequel il est possible de distinguer "intuitivement" l'ordre temporel d'une séquence de photos du processus en action, est dit **irréversible**. Notre procédure d'échange stochastique de particules entre deux boites communiquantes est une représentation très simple d'un processus de **diffusion**, phénomène sur lequel nous reviendrons au chapitre 7. On y verra que le mouvements des particules les faisant éventuellement traverser le trou est un ensemble de mouvements rectilinéaires entrecoupés de collisions (élastiques) avec les parois ou avec d'autres particules. Comme vous l'avez déjà vu au CEGEP ou même au secondaire, une collision élastique entre deux corps est un processus réversible; si vous filmez la collision, que vous rejouez le film à l'endroit ou à l'envers, rien ne vous permet de distinguer la direction temporelle de l'expérience originale.

Nous avons donc la situation suivante; à l'échelle "macroscopique" du système (les boites), l'évolution temporelle est irréversible. À l'échelle "microscopique" du système, le phénomène qui en gouverne la dynamique (les collisions) est lui parfaitement réversible. Comment une séquence d'événements réversibles peut-elle produire une évolution globale du système qui soit irréversible? Ceci est une des questions les plus fondamentales à laquelle vous aurez à faire face durant vos étude en physique, et elle vous tiendra fort occupé(e)s pendant quelques cours entiers. Celà pourrait donc vous être utile de commencer à y cogiter dès maintenant...

Exercices:

1. Écrivez un code C utilisant l'approche Monte Carlo au calcul de la surface de l'étang (Fig. 5.1) pour calculer le volume d'une "hypersphère" de rayon R en D dimensions; une telle hypersphère est définie par la relation:
$$x_1^2 + x_2^2 + x_3^2 + ... + x_D^2 = R^2 \ ,$$
où les x_k représentent les axes de coordonnées de votre hyperespace D-dimensionnel. Testez votre code pour la sphère habituelle ($D = 3$), pour laquelle vous savez évidemment que $V = 4\pi R^3/3$.

2. La Loi dite de Gutenberg-Richter, établie empiriquement, nous informe que le nombre N de temblement de Terre de magnitude entre m et $m + \mathrm{d}m$ varie selon la distribution:
$$N(m) = N_0 m^{-1} \ , \qquad 1 \leq m \leq 10 \ .$$
Une telle distribution est appelée une **loi de puissance**.

(a) Déduisez la transformation mathématique qui permettra de produire une telle distribution à l'aide de nombres aléatoires distribués uniformément, selon la procédure décrite à la §5.1;

(b) Générez 1000 "événements" distribués selon cette distribution, et vérifiez graphiquement que leur histogramme prend bien la forme d'une loi de puissance;

(c) Reprenez votre liste de 1000 événements, et calculez en la moyenne cumulative:

$$\langle m \rangle_n = \frac{1}{n} \sum_{k=1}^{n} m_k \ , \qquad n = 2, 3, ..., 10^3 \ .$$

Portez en graphique $\langle m \rangle_n$ versus n; la moyenne est-elle une quantité statistiquement stable ici? Pourquoi?

3. Calculez par Monte Carlo l'intégrale définie suivante:

$$\int_1^2 \int_0^1 \int_{-1}^2 \int_0^{5/2} \int_{-\pi}^{\pi} \int_{1/2}^{5/2} \left(\frac{3}{2} + \frac{(u^2 + \sqrt{x^3 + y^2 + z^2 \sin^2(2\pi v)}) \cos(\sqrt{x^2 + y^2}\pi w)}{(1 - \sin(2v + z + \pi/2) \cos(\pi y))^2 \exp(-(u + v + w)/3)} \right)$$
$$\times (\exp(-z\, w/2) - 1)\, \mathrm{d}u\, \mathrm{d}v\, \mathrm{d}w\, \mathrm{d}x\, \mathrm{d}y\, \mathrm{d}z \ .$$

Le codage en C de l'intégrand devrait à lui seul représenter un excellent exercice... Ensuite,

(a) Portez en graphique le résultat en fonction du nombre d'évaluations Monte Carlo; combien d'évaluation sont requises pour obtenir un résultat stable au troisième chiffre significatif?

(b) Appellons le nombre d'évaluation déterminé ci-dessus N; comme il s'agit ici d'une intégrale en six dimensions, calculer l'intégrale par méthode du trapèze généralisée à 6 dimensions avec le même nombre N d'évaluations de l'intégrand imposerait donc une résolution de $N^{1/6}$ points équidistants par dimension spatiale. Selon vous, considérant la forme de l'intégrand, est-ce suffisant?

Bibliographie:

Le chapitre 7 du *Numerical Recipes* offre une excellente présentation de la génération de nombres aléatoires sur ordinateurs. Je vous recommanderais de consulter d'abord la seconde édition (1992), dont des copies en version C ou FORTRAN trainent un peu partout dans les bureaux de vos profs et de plusieurs de vos TP-istes. Le générateur `ran0` présenté à la §5.1 de ces notes est tiré directement de la version C, et fait partie de la famille des générateurs dits congruentiels. La grande référence classique sur ce sujet est:

Park, S.K., & Miller, K.W., *Comm. of the ACM*, **31**, 1192 (1988).

La troisième édition du Numerical Recipes abandonne effectivement les générateurs congruentiels au profit d'approches plus sophistiquées, mais plus difficile à saisir, autant au niveau de la théorie que de l'implémentation numérique. C'est pourquoi je vous recommande de commencer par la seconde édition.

Tous les ouvrages de statistique le moindrement raisonnables discuterons de la production et des propriétés de diverses FDP non-uniformes; je trouve très pratique et instructive la compilation présentée au chapitre 4 de l'ouvrage:

James, F., *Statistical Methods in Experimental Physics*, 2e éd., World Scientific (2006).

Le chapitre 2 de cet ouvrage offre également une bonne introduction aux probabilités, et à leur distributions. Un autre ouvrage notable dans la même veine est:

Roe, B.P., *Probability and Statistics in Experimental Physics*, Springer (1992).

Au niveau de l'approche Monte Carlo en modélisation physique, l'ouvrage de Gould & Tobochnik cité en bibliographie au chapitre 1 contient d'excellent exemples. À un niveau physique plus avancé, et plus centré sur les applications en physique des matériaux, l'ouvrage suivant me semble pas mal:

Frenkel, D., & Smit, B., *Understanding Molecular Simulations*, 2$^{\mathrm{e}}$ éd., Academic Press (2002).

Chapitre 6

Racines et optimisation

Dans ce chapitre nous allons développer quelques algorithmes numériques permettant de traiter deux classes de problèmes mathématiques à prime abord distincts: la recherche de racines, et celle d'extrema.

Une **racine** r d'une fonction $f(x)$ est définie comme une valeur $x = r$ telle que $f(r) = 0$. Par exemple, une relation linéaire $y = mx + b$ a une seule racine à $r = -b/m$, tandis qu'un polynôme quadratique de la forme

$$y = ax^2 + bx + c = 0 \tag{6.1}$$

a ses deux racines à:

$$r_\pm = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \ . \tag{6.2}$$

Vous n'aurez probablement pas oublié que déjà avec un polynôme cubique, les choses se compliquaient sérieusement. Les premières sections du présent chapitre vous feront faire connaissance avec deux techniques simples permettant le calcul numérique des racines d'une fonction $f(x)$ d'une seule variable, soit la bissection (§6.2) et la méthode de Newton (§6.3). Nous considérerons ensuite la recherche des valeurs extrêmes, ou **extrema**, d'une fonction. Les extrema[1] viennent en deux classes, soit les minima et les maxima; un maxima est une valeur x^* telle que $f(x^*) > f(x)\, \forall x \neq x^*$, et l'inverse pour les minima.

Le lien profond entre la recherche de racine et celle des extrema provient évidemment du fait que pour un extremum on doit avoir $\mathrm{d}f/\mathrm{d}x = 0$, ce qui fait que les problèmes de recherche d'extrema peuvent souvent être convertis en problèmes de recherche de racines. Cependant, comme nous aurons l'occasion de le constater plus loin dans ce chapitre, dès qu'on passe à une fonction de deux variables, il est rarement avantageux d'effectuer cette conversion.

Une des tâches les plus courantes en physique expérimentale est d'ajuster un modèle (habituellement défini en termes d'un ou plusieurs coefficients numériques ajustables) sur un ensemble de données expérimentales ou observations. **L'optimisation** consiste à choisir les valeurs numériques des divers paramètres ajustables de manière à offrir la meilleure représentation possible des données. Ceci est habituellement fait en minimisant une mesure de l'écart entre les prévisions du modèle, et les observations dont il doit en principe offrir une représentation théorique. L'optimisation se retrouve donc souvent formulée comme un problème de minimisation. Comme on le verra plus bas, le choix de l'une ou l'autre technique de minimisation sera souvent dicté par la complexité de la relation mathématico-physique reliant le modèle au données.

[1]extrema: nominatif pluriel du latin extremum; donc SVP pas de “s” à extrema (ou minima, maxima, etc). Et encore moins à extremum, minimum, maximum, etc. Fin de la parenthèse culture générale...

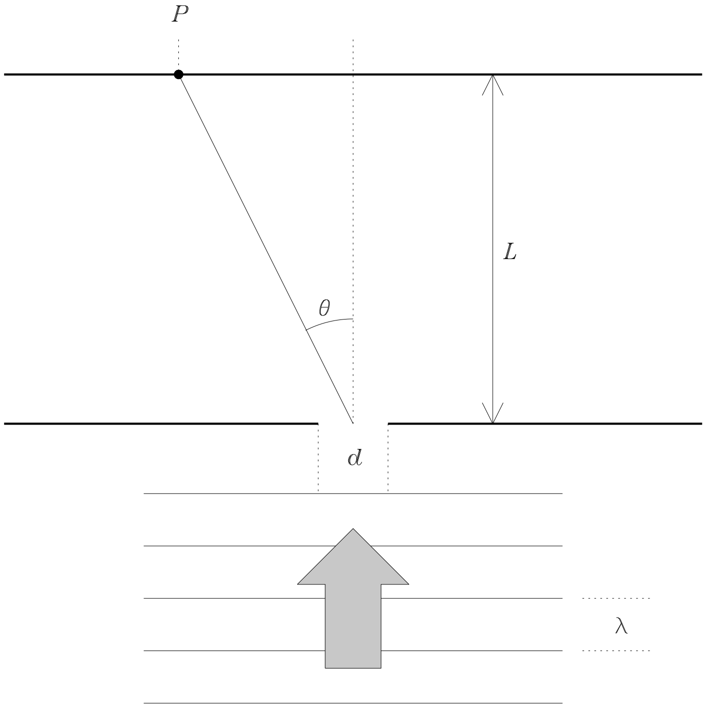


Figure 6.1: Géométrie du problème de diffraction par une fente simple de largeur d illuminée par une onde plane monochromatique de longueur d'onde λ. L'intensité lumineuse est mesurée au point P sur un écran placé derrière la fente.

6.1 La diffraction

Commençons par quelque chose que vous connaissez déjà, c'est toujours bon pour le moral. Vous avez appris au CEGEP que la diffraction de la lumière par un orifice (comme une fente simple) peut être calculée en invoquant le principe de superposition de Huygens, selon lequel les amplitudes des fronts d'onde associés à une rangée de sources ponctuelles contigues couvrant la fente s'additionnent linéairement, mais avec un déphasage relatif correspondant à la différence du chemin optique parcouru. La superposition des fronts d'onde produits par un ensemble de sources ponctuelles situées le long de la fente peut donc conduire à une interférence constructive ou destructive, partielle ou totale. Pour une longue fente de largeur d éclairée par un faisceau lumineux monochromatique de longueur d'onde λ, la variation de l'amplitude $A(\theta)$ de l'onde lumineuse mesurée à un angle θ par rapport à la ligne reliant la fente au centre d'un écran placé parallèlement et à une distance L de celui dans lequel est découpée la fente (voir Figure 6.1) varie selon

$$A(x) = \frac{\sin(x)}{x} \ , \tag{6.3}$$

où la variable x est définie comme:

$$x = \frac{\pi d}{\lambda} \sin\theta \ . \tag{6.4}$$

Ce profil d'amplitude est porté en graphique sur la Figure 6.2 avec une représentation synthétique du profil d'intensité ($\propto A^2$) tel qu'il serait observé sur l'écran. L'alternance régulière

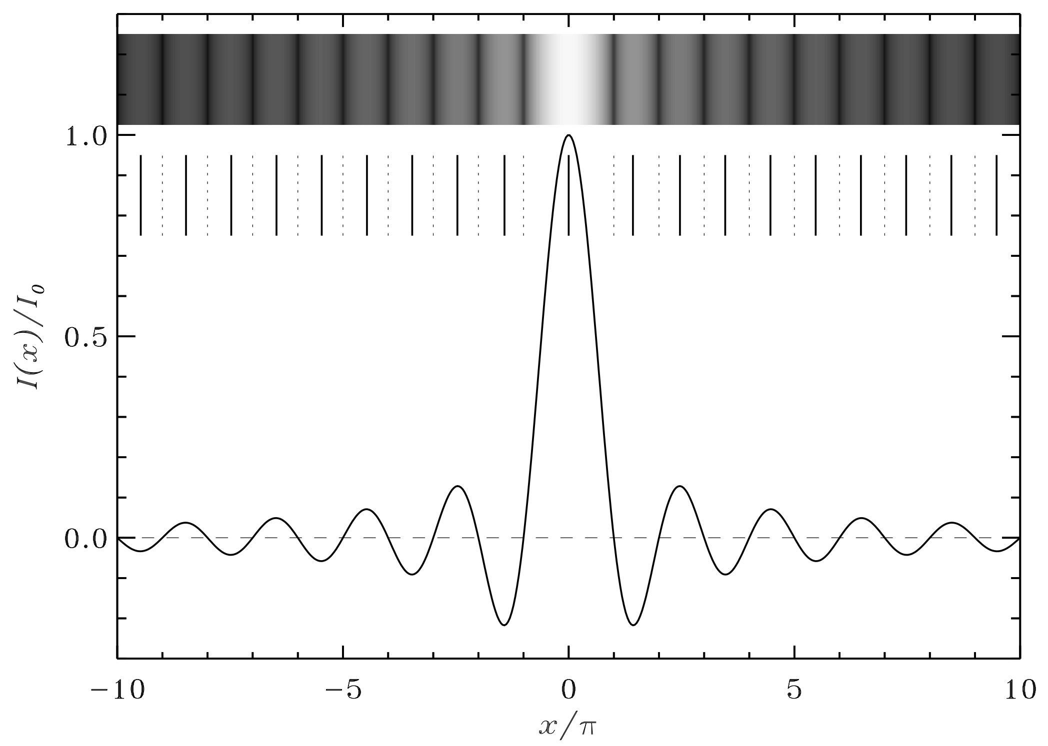


Figure 6.2: Patron de diffraction d'une fente simple. L'intensité lumineuse est proportionnelle au carré de l'amplitude de l'onde, portée en graphique ici. Le paramètre x mesure effectivement la distance à partir du centre du patron de diffraction (voir eq. 6.4). Les tirets pointillés indiquent la position des minima d'intensité, et ceux en trait plein la position des maxima.

de bandes brillantes et sombres est caractéristique du phénomème de diffraction, qui n'est évidemment pas restreint aux fentes rectilignes.

Les minima d'intensité se produisent aux endroits où l'amplitude totale de l'onde plane diffractée est nulle. La dépendance sinusoidale du numérateur de l'éq. (6.3) indique que ces minima apparaitront aux positions telles que

$$x = \pm n\pi \ , \qquad n = 1, 2, 3... \tag{6.5}$$

La question à savoir ce qui se passe à $x = 0$ se règle par la Règle de l'Hospital, qui nous informe que

$$\lim_{x\to 0} \frac{\sin(x)}{x} = \lim_{x\to 0} \cos(x) = 1 \ . \tag{6.6}$$

Pour trouver les positions des pics d'intensité nous n'avons qu'à dériver le membre de droite de l'éq. (6.3) par rapport à x, et exiger que le résultat soit égal à zéro:

$$\frac{\mathrm{d}}{\mathrm{d}x}\left(\frac{\sin x}{x}\right) = \frac{\cos(x)}{x} - \frac{\sin x}{x^2} = 0 \ . \tag{6.7}$$

Sauf que voilà, aucune manipulation algébrique légale connue ne permet d'isoler le x dans cette expression. En fait, l'équation (6.7) définit un problème de recherche de racine pour une fonction $f(x)$ que l'on pourrait écrire comme:

$$f(x) = \tan(x) - x \ . \tag{6.8}$$

On vous a peut-être même déjà fait solutionner ça via une méthode graphique, en mesurant sur du papier quadrillé les points d'intersection des fonctions $y = \tan x$ et $y = x$. On aurait pu aussi ne pas diviser par $\cos\theta$, ce qui aurait conduit à une seconde forme de la fonction ayant les mêmes racines, soit:

$$f(x) = x\cos(x) - \sin(x) \ . \tag{6.9}$$

Au sens strictement mathématique, les équations (6.8) et (6.9) sont évidemment équivalentes. Mais du point de vue d'une recherche numérique de racine, la version définie par l'éq. (6.9) est préférable à (6.8), car cette dernière diverge là où $\cos\theta = 0$, soit à un ensemble discret de valeurs $x = (n + 1/2)\pi$, $n = 0, 1, ...$

6.2 La bissection

Nous cherchons donc les racines de l'éq. (6.9). Supposons que nous pouvons établir *a priori* un intervalle $[x_1, x_2]$ dans lequel nous cherchons une racine r devant être déterminée avec une précision donnée ε. Comme son nom l'indique, la méthode de la **bissection** converge vers la racine par bissection successive de l'intervalle de recherche. L'idée de base ici n'est rien de plus subtil que le fait que $f(x)$ doive changer de signe en passant par un zéro! Etant donné un intervalle de départ $[x_1, x_2]$ et les valeurs correspondante f_1 et f_2 (de signes opposés si x_1 et x_2 encadrent une racine), on calcule un nouveau $f_m \equiv f(x_m)$ à mi-chemin entre x_1 et x_2; si f_m est du même signe que f_2, la racine doit être entre x_1 et x_m, et on reprend donc la bissection avec $x_2 = x_m$; si par contre f_m et f_2 sont de signes opposés, alors la racine est entre x_m et x_2, et on reprend la bissection avec $x_1 = x_m$. La valeur de x_m correspond à l'estimé de la racine, et l'écart $x_2 - x_1$ à la précision absolue de cet estimé. Un programme en C effectuant une recherche de racine par bissection est listé à la Figure 6.3. Notons déjà que la méthode de la bissection ne requiert que la capacité d'évaluer la fonction $f(x)$ à toute valeur de x, mais n'exige aucun calcul ou estimé de la dérivée de cette fonction.

La Figure 6.4 illustre l'application de la bissection dans le cas de notre problème de recherche d'extremum du patron de diffraction d'une fente simple, tel que décrit par l'éq. (6.9). Dans cette situation le choix de l'intervalle de départ est facile, puisque, nous sommes en droit de supposer que l'extremum est situé quelque part entre les premier et second minima, dont nous connaissons les positions: $x = \pi$ et $x = 2\pi$.

La partie du bas de la Figure illustre l'évolution de la taille et position de l'intervalle de recherche, les itérations allant du haut vers le bas. L'algorithme garantit que la largeur de l'intervalle de recherche diminue comme $1/2^n$, où n est le décompte des itérations depuis le début du processus de bissection. Ceci implique que le nombre total d'itérations requis pour atteindre une tolérance ε est donné par

$$n = \log_2\left(\frac{x_2 - x_1}{\varepsilon}\right) \ , \tag{6.10}$$

soit une quarantaine d'itérations pour $\varepsilon = 10^{-12}$ et $x_2 - x_1 \sim 1$.

La bissection offre plusieurs caractéristiques avantageuses, non la moindre étant qu'elle *garantit* la détection d'une racine *si* une racine existe dans l'intervalle spécifié. Remarquez cependant que si plusieurs racines se retrouvent dans l'intervalle initial, la méthode convergera sur l'une d'elles sans donner d'indications que d'autres racines sont présentes. Si le changement de signe entre x_1 et x_2 est du à une discontinuité (comme pour $\tan x$ à $x = (n + 1/2)\pi$, $n = 0, 1, ...$), la bisection vous trouvera la discontinuité sans rechigner. Si aucune racine ne se trouve entre x_1 et x_2, l'algorithme listé plus haut convergera vers une des deux borne initiales (x_1 si tel qu'implémenté ici).

```
#include <stdio.h>
#include <math.h>
/* Ce programme calcule la racine d'une fonction d'une variable,     */
/* entre deux bornes x_1 et x_2.                                     */
int main(void)
{
/* Declarations ---------------------------------------------------- */
  float fonc(float) ;          /* Declaration de la fonction fonc(x) */
  int k=0 ;                    /* Compteur d'iterations */
  int itermax=40 ;             /* Nombre maximal d'iterations */
  float x1, x2 ;               /* bornes de l'intervalle initial */
  float xm ;                   /* Ceci sera notre racine */
  float epsilon=1.e-5  ;       /* Precision requise pour la racine */
  float pi=3.1519926536 ;      /* la valeur de pi */
  float fm, f2, delta;         /* Variables locales */
/* Executable ------------------------------------------------------ */
  x1 = pi+0.1 ;                /* Borne gauche */
  x2 = 2.*pi-0.1 ;             /* Borne droite */
  delta = x2-x1 ;
  while ( delta > epsilon && k < itermax)  /* Boucle bissection */
    {
      xm = 0.5*(x1+x2) ;       /* Point milieu de l'intervalle */
      fm = fonc(xm) ;          /* Valeur de fonc(x) en ce point */
      f2 = fonc(x2)
      if ( fm*f2 > 0. )        /* Le demi-intervalle droit a la racine */
        { x2 = xm ; }          /* L'intervalle devient la moitie droite */
      else                     /* ...sinon c'est le gauche qui a la racine */
        { x1 = xm ; }          /* L'intervalle devient la moitie gauche */
      delta = x2-x1 ;
      k = k + 1 ;              /* Decompte des iterations */
      printf ("racine de la fonction = %f +/- %f\n", xm,delta) ;
    }
  printf ("racine de la fonction = %f +/- %f\n", xm,delta) ;
}
/* Definition de la fonction dont on recherche la racine */
float fonc (float x)
{
  float f ;                    /* Variable locale */
  f = x * cos(x) - sin(x) ;    /* La fonction qu'on calcule */
  return f ;
}
```

Figure 6.3: Programme en C effectuant une recherche de racine par bissection d'un intervalle $[x_1, x_2]$ (avec $x_2 > x_1$) fourni par l'usager, et présumé contenir au moins une racine. Notez la condition composée contrôlant la boucle `while`, avec la partie `k < itermax` assurant que la boucle ne pourra pas tourner plus de `itermax` itérations.

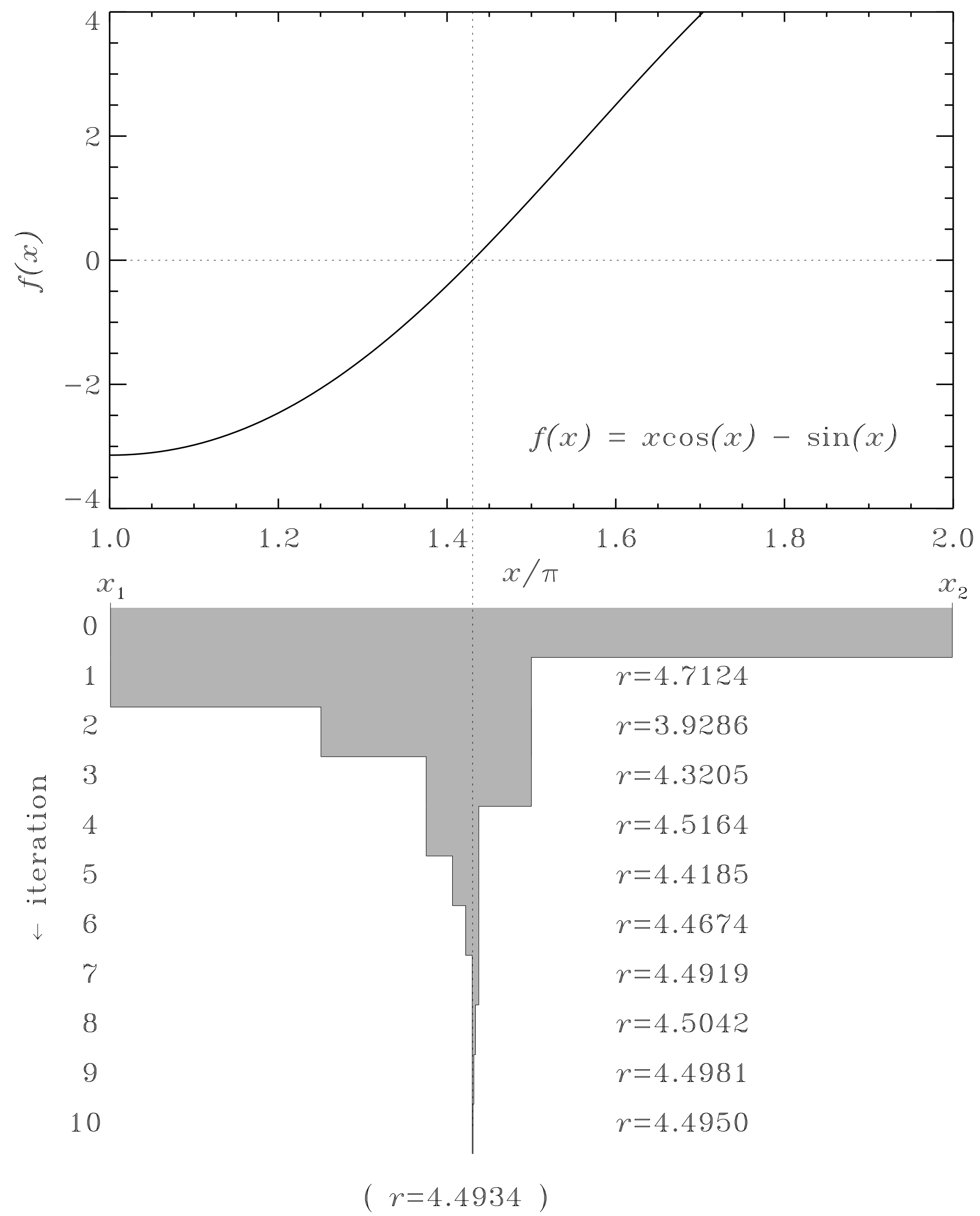


Figure 6.4: Fonctionnement de la méthode de la bissection pour la recherche du premier maxima du patron de diffraction d'une fente simple. La partie du haut montre la variation du $f(x)$ défini par l'éq. (6.9) en fonction de x, et la partie du bas l'évolution de l'intervalle de bissection (en gris) pour le 10 premières itérations de l'algorithme (allant du haut vers le bas)

6.3 La méthode de Newton

Si vous n'avez pas la patience d'attendre une quarantaine d'itérations, il existe toute une gamme de méthodes de recherche de racines qui, *quand tout va bien*, convergent beaucoup plus rapidement que la bissection. La méthode dite de Newton (oui, le même que $F = ma$) est une grande classique parmi ces méthodes, mais elle n'est applicable efficacement qu'aux situations où la dérivée de la fonction dont les racines sont recherchées est calculable analytiquement. Son point de départ est encore une fois notre très fameux développement en série de Taylor d'une fonction $f(x)$ dans le voisinage de $x = x_0$:

$$f(x_0 + \Delta x) = f(x_0) + (\Delta x)\frac{\mathrm{d}f}{\mathrm{d}x}\bigg|_{x_0} + \frac{(\Delta x)^2}{2!}\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}\bigg|_{x_0} + \frac{(\Delta x)^3}{3!}\frac{\mathrm{d}^3 f}{\mathrm{d}x^3}\bigg|_{x_0} + \dots \tag{6.11}$$

Si l'on suppose maintenant que $f(x)$ a une racine à $x = r$, et que cette racine n'est pas trop éloignée de x_0, alors on peut définir un Δx tel que $x_0 + \Delta x = r$, et tronquer le développement au second terme:

$$f(r) = f(x_0) + (r - x_0)\frac{\mathrm{d}f}{\mathrm{d}x}\bigg|_{x_0} \ . \tag{6.12}$$

Mais le membre de gauche est nul si r est une racine, et donc r devrait être donné par

$$r = x_0 - \frac{f(x_0)}{\mathrm{d}f/\mathrm{d}x} \ . \tag{6.13}$$

Évidemment, la racine r ainsi calculée ne sera pas exactement égale à la racine réelle, puisque son calcul résulte de l'utilisation d'un développement tronqué, ce qui introduit automatiquement une erreur dans le calcul de la racine. Mais le processus peut être répété en révéaluant f et sa dérivée à $x = r$, et en réutilisant l'éq. (6.13) pour recalculer un nouvel estimé de r, et ainsi de suite itérativement jusqu'à ce que la racine ait été localisée au niveau de précision numérique désirée. Un programme en C effectuant une recherche de racine par la méthode de Newton est listé à la Figure 6.5, toujours pour notre problème de recherche des maxima du patron de diffraction.

La Figure 6.6 illustre les premières 4 itérations de ce processus, dans le cas de la recherche d'une racine du polynôme $f(x) = x^4 - 1$; ce polynôme compte quatre racines distinctes (± 1, et $\pm i$), et pour une position de départ $x_0 = 1.8$ la méthode de Newton converge très rapidement sur la racine $r = 1$. On notera que la méthode de Newton revient à utiliser la dérivée de $f(x)$ pour construire une extrapolation linéaire vers $f(r) = 0$.

La Figure 6.7 illustre le taux de convergence de la méthode de Newton ($\bullet$), maintenant appliquée à la recherche du premier extremum (non-central) dans le patron de diffraction d'une fente simple. La convergence est ici extrêmement rapide, avec l'erreur sur r chutant à 10^{-7} après seulement 4 itérations (et à $\sim 10^{-14}$ à la cinquième!). La bissection ($\triangle$), en comparaison, converge beaucoup plus lentement, même si aux itérations 2 et 3 les erreurs associées aux deux méthodes sont comparables.

On pourrait être tenté de conclure, après étude de la Figure 6.7, que la méthode de Newton devrait être préférée à la bissection. En fait ce n'est pas le cas. La méthode de Newton est beaucoup plus rapide, mais beaucoup moins robuste, que la bissection. Pour une fonction monotoniquement croissante ou décroissante, sans trop de changement de signe de la dérivée seconde (comme pour le polynôme quartique de la Fig. 6.6), la méthode de Newton est imbattable. Mais imaginez maintenant une fonction de forme plus complexe, et où une itération de la méthode de Newton vous fait atterrir à une valeur de x où $\mathrm{d}f/\mathrm{d}x \simeq 0$; l'éq. (6.13) garantit que votre prochaine estimé de r se retrouvera quelque part dans la galaxie d'Andromède (à moins d'avoir eu la géniale idée d'ajouter un test de divergence à l'algorithme de base listé ici). De plus, la méthode de Newton appliquée à une fonction présentant un profil antisymétrique sur réflexion par rapport à la racine peut se coincer dans un cycle non-convergent. Ces problèmes ne se posent pas avec la bissection.

```
#include <stdio.h>
#include <math.h>
/* Ce programme calcule la racine d'une fonction d'une variable      */
/* par la methode de Newton                                         */
int main(void)
{
/* Declarations --------------------------------------------------- */
  float fonc(float), dfdx(float) ;  /* Fonction et sa derivee */
  int k=0 ;                         /* Compteur d'iterations */
  int itermax=20 ;                  /* Nombre maximal d'iterations */
  float xr ;                        /* Ceci sera notre racine */
  float epsilon=1.e-6 ;             /* Precision pour la racine */
  float pi=3.1415926536 ;           /* Valeur de pi */
  float delta ;                     /* Variable locale */
/* Executable ----------------------------------------------------- */
  xr   = 1.5*pi ;                   /* Point de depart */
  delta= 1. ;                       /* Nombre plus grand qu'epsilon  */
  while (fabs(delta) > epsilon && k <= itermax )    /* Boucle Newton */
    {
      delta = -fonc(xr)/dfdx(xr) ;  /* calcul de l'increment en x */
      xr    = xr+delta ;            /* Nouvelle valeur de x */
      k     = k+1 ;
    }
  printf ("racine de la fonction = %f +/- %f", xr,delta) ;
}
/* Definition de la fonction dont on recherche la racine */
float fonc (float x)
{
  float f ;                  /* Variable locale */
  f = x * cos(x) - sin(x) ;  /* La fonction qu'on calcule */
  return f ;
}
/* Definition de la derivee de la fonction precedente */
float dfdx (float x)
{
  float df ;                 /* Variable locale */
  df = -x * sin(x) ;         /* Derivee de la fonction fonc */
  return df ;
}
```

Figure 6.5: Programme en C effectuant une recherche de racine par la méthode de Newton, à partir d'une valeur de départ x_r fournie par l'usager. Il s'agit ici d'une implémentation minimale; en réalité il faudrait au moins ajouter un test pour limiter la grandeur du pas `delta`, pour se protéger de situations où l'on atterit à une valeur de x pour laquelle $\mathrm{d}f/\mathrm{d}x \simeq 0$. Notez que l'usager doit fournir ici *deux* fonctions C, la première calculant $f(x)$, et la seconde sa dérivée $\mathrm{d}f/\mathrm{d}x$.

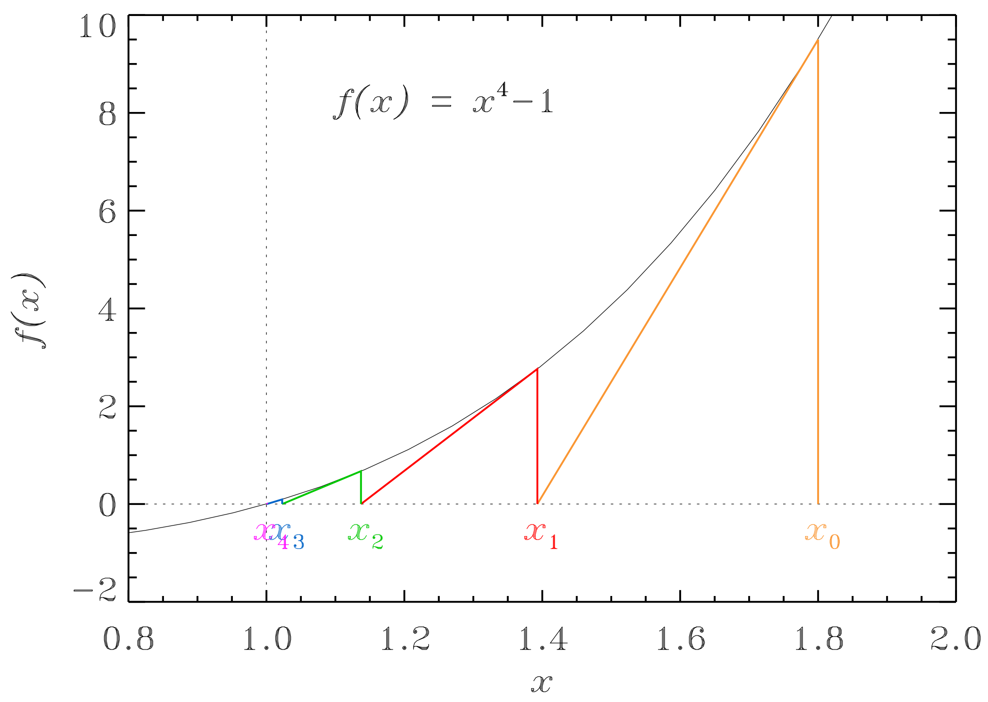


Figure 6.6: Les quatre premières itérations de la méthode de Newton pour la recherche de racines, ici la racine $r = 1$ du polynôme quartique $f(x) = x^4 - 1$.

Une stratégie très judicieuse consiste à débuter la recherche de racine à l'aide de la bissection, et une fois la racine localisée approximativement, de passer à la méthode de Newton afin d'aller chercher deux ou trois chiffres de précision supplémentaires sans trop d'efforts. Ou encore, effectuer une ou deux étapes de bissection entre deux itérations de la méthode de Newton, si la convergence semble trop lente[2]. Ce genre de **méthode hybride** est souvent optimal, et on verra qu'il peut l'être également en optimisation, sujet des prochaine sections.

6.4 Maximisation (et minimisation)

La maximisation d'une fonction $f(\mathbf{x})$ consiste à trouver le (ou les) point(s) $\mathbf{x}^*$ tel que

$$f(\mathbf{x}^*) > f(\mathbf{x}) \ , \qquad \forall \mathbf{x} \neq \mathbf{x}^* \ . \tag{6.14}$$

Vous comprendrez j'espère que trouver le ou les maxima d'une fonction $f(\mathbf{x})$ est absolument équivalent à trouver les minima de $-f(\mathbf{x})$, en conséquence de quoi nous allons nous restreindre aux problèmes de maximisation sans aucune perte de généralité.

Pour une fonction d'une seule variable et exprimable analytiquement, la maximisation est souvent convertie en problème de recherche de racine, comme nous l'avions fait à la §6.1 pour la recherche des maxima d'intensité du patron de diffraction produit par une fente simple. En plus d'une variable cependant, cette approche est rarement utilisée. Considérons par exemple le patron de diffraction du à une ouverture carrée de coté d; l'intensité $I(x, y)$ est alors donnée par une expression du genre:

$$\frac{I(x,y)}{I_0} = \left(\frac{\sin x}{x}\frac{\sin y}{y}\right)^2 \ , \tag{6.15}$$

[2]Vous trouverez un algorithme de ce genre au chapitre 9 du *Numerical Recipes*.

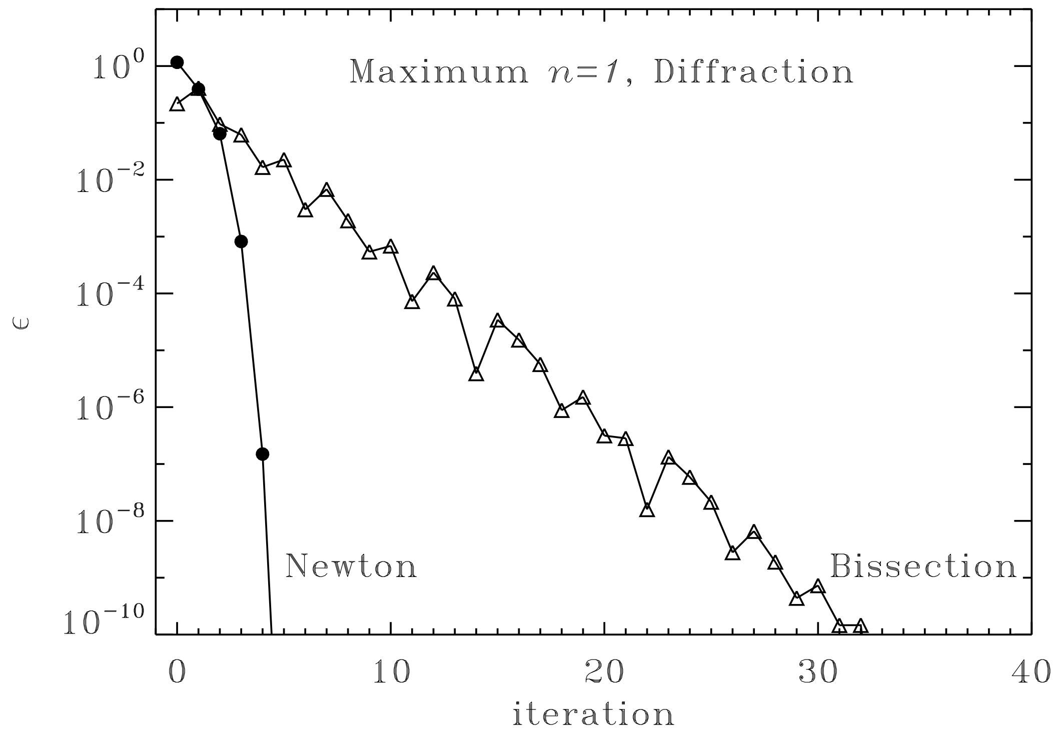


Figure 6.7: Convergence de la méthode de Newton (•) et de la bissection (△) sur la recherche du premier maxima du patron de diffraction d'une fente simple.

avec x et y donnés par des expressions semblables à l'éq. (6.4). La Figure 6.8 illustre la surface correspondant à cette fonction de deux variables. Un maximum dans ce "paysage" bidimensionnel correspond à un point (x^*, y^*) tel que

$$\nabla I(x,y) \equiv \frac{\partial I(x,y)}{\partial x}\hat{\mathbf{e}}_x + \frac{\partial I(x,y)}{\partial y}\hat{\mathbf{e}}_y = 0 \ . \tag{6.16}$$

Comme l'opérateur gradient est un opérateur vectoriel, chacune des deux dérivées doit valoir zéro séparément pour un maximum de $I(x,y)$. La substitution de (6.15) dans (6.16) produit donc un système de deux équations algébriques nonlinéaires couplées, où chacune des équations décrit une courbe ou famille de courbes (souvent topologiquement complexe) dans le plan $[x,y]$, et les maxima correspondent aux intersections de ces familles de courbes. Trouver ca numériquement, c'est du costaud. Il est habituellement préférable de rechercher directement les maxima. Voyons comment.

6.5 Méthodes de grimpe

Les bouquins d'analyse numérique débordent de techniques, habituellement affublées de noms inprononçables, pour la maximisation (ou minimisation) de fonctions de plusieurs variables. Fondamentalement, la vaste majorité de ces méthodes sont basées sur la **grimpe**, et fonctionnent de la manière illustrée allégoriquement à la Figure 6.9, dans le contexte d'une fonction de deux variables. On commence par se placer quelque part à un point (x,y) (possiblement choisi aléatoirement) dans le paysage bidimensionnel (Fig. 6.9A); l'atterissage (en B) n'est habituellement pas problématique numériquement si la fonction est continue en x et y. Une fois remis sur pied, on explore le voisinage local du point (x,y) et on se déplace dans une direction ascendante

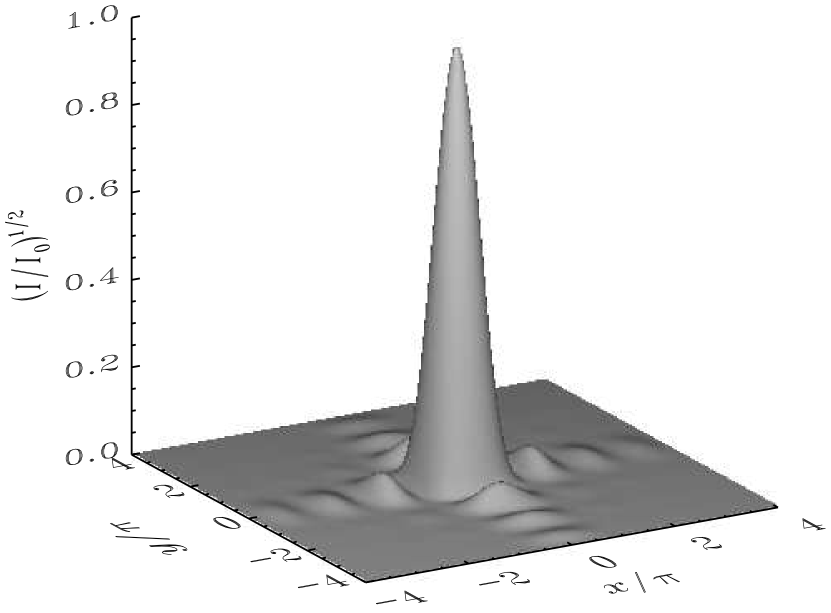


Figure 6.8: Le patron d'intensitée $I(x, y)$ produit par la diffraction d'une onde plane à travers une ouverture carrée, représenté sous la forme d'une surface dans le plan $[x, y]$. Le maximum est à $I(0, 0)/I_0 = 1$.

(en C). Le processus est répété séquentiellement jusqu'à ce qu'on se retrouve à un point (x^*, y^*) où toutes les directions partant de ce point soient descendantes. Le maximum est alors localisé (D). Caramba! Pour traduire la Figure 6.9 en un algorithme exprimable en C, nous devons d'abord faire plusieurs choix pratiques:

1. Comment choisir la position de départ;
2. Comment calculer la direction du déplacement;
3. Comment calculer la grandeur du pas;
4. Comment utiliser l'information accumulée aux pas précédents;
5. Comment décider que le maximum est bel et bien localisé.

Ce sont les différentes réponses possibles à ces questions qui sont responsables de l'effrayante multiplicité d'algorithmes disponibles dans les bouquins. Nous allons ici en développer un qui est simple à coder, relativement robuste, et facilement généralisable à des problèmes de maximisation pour des fonctions de plus de deux variables. Il est basé sur l'utilisation du gradient de la fonction à maximiser pour choisir la direction du déplacement. Le gradient d'une fonction scalaire (comme dans le cas de l'équation (6.16)) indique la direction de pente maximale dans un espace des paramètres ayant la dimensionalité du nombre de variables indépendantes de la fonction à maximiser. Dans les cas de la diffraction de la lumière par une ouverture carrée, le gradient est un vecteur contenu dans le plan $[x, y]$, dont les composantes x et y sont données par $\partial I/\partial x$ et $\partial I/\partial y$, avec $I(x, y)$ donné par l'éq. (6.15). À partir d'une position (x^n, y^n), on

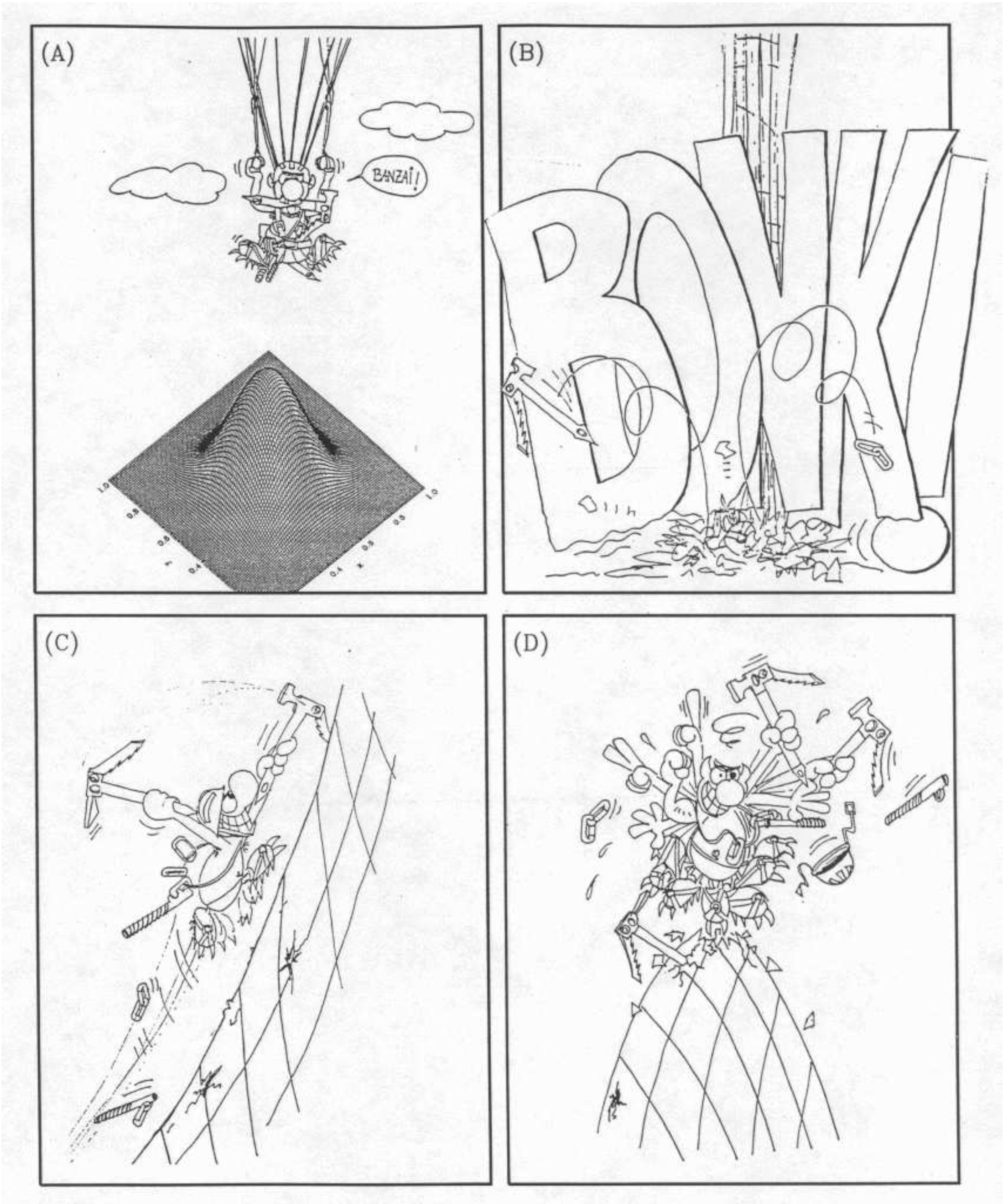


Figure 6.9: Méthode de grimpe (allégorie).

pose simplement

$$x^{n+1} = x^n + \frac{\delta}{|\nabla f|}\frac{\partial f}{\partial x} , \qquad y^{n+1} = y^n + \frac{\delta}{|\nabla f|}\frac{\partial f}{\partial y} , \tag{6.17}$$

où δ mesure la longueur du pas dans le plan $[x, y]$, et

$$|\nabla f| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2} . \tag{6.18}$$

Si vous ne l'avez pas déjà deviné, cet algorithme est l'équivalent d'un double développement en série de Taylor, tronqué aux seconds termes en x et y! Les composantes du gradient peuvent être calculées soit analytiquement, soit par différences finies; par exemple, à l'ordre $O(h^2)$,

$$\frac{\partial f}{\partial x} = \frac{f(x+\delta, y) - f(x-\delta, y)}{2\delta} , \qquad \frac{\partial f}{\partial y} = \frac{f(x, y+\delta) - f(x, y-\delta)}{2\delta} . \tag{6.19}$$

Le bout de code en C présenté à la Figure 6.10 implémente cet algorithme pour la recherche des maxima du patron de diffraction d'une ouverture carrée, avec évaluation des composantes du gradient par différences finies: Étudiez bien ce bout de code, et notez bien qu'un test a été ajouté pour réduire la grandeur du pas δ d'un facteur deux si le pas conduit à une position où $f(x^{n+1}, y^{n+1}) < f(x^n, y^n)$, afin d'empêcher de sauter par dessus le maximum et/ou se coincer dans un cycle non-convergent.

La Figure 6.11 montre les trajectoires conduisant aux maxima, pour une série de points de départ (x^0, y^0) (indiqués par des $\bullet$ colorés) choisis aléatoirement. L'espace des paramètres a été restreint ici à $\pi \leq x, y \leq \pi$, soit le pic central d'intensité du patron de diffraction 2D, et les pseudo-cercles en noir correspondent à des courbes de niveau $I(x, y) =$ constante. Quel que soit le point de départ (x^0, y^0), l'algorithme aboutit à $(0, 0)$, comme il se doit. Dépendant du (x^0, y^0) initial, de 10 à 20 itérations sont requises pour que l'erreur sur l'évaluation de $f(0, 0) = 1$ atteigne 10^{-4}. De toute évidence, ça marche!

Mais il ne fait pas trop vite crier victoire... La Figure 6.12 montre le résultat de notre méthode de grimpe, avec l'intervalle de recherche maintenant élargi à $-2\pi \leq x \leq 2\pi$, $-2\pi \leq y \leq 2\pi$. Il y a maintenant 50 points de départ, tous choisis aléatoirement mais distribués uniformément dans l'intervalle. Dans tous les cas la méthode de grimpe fait son travail et livre la marchandise, dans le sens que les grimpes se terminent toutes sur un maximum, mais dans bien des cas il ne s'agit pas du maximum le plus élevé à $(0, 0)$, mais plutôt de l'un ou l'autre des 8 autres maxima secondaires présents ici dans l'espace des paramètres. Le maximum sur lequel converge la grimpe devient fonction de sa position de départ, une situation embarrassante puisque dans bien des cas nous n'avons pas idée de la position du maximum global, et donc ne sommes pas en mesure de fournir à l'algorithme un "bon" point de départ.

Les Figures 6.11 et 6.12 offrent conjointement une excellente illustration de la distinction entre un problème d'optimisation **locale** versus l'optimisation **globale**. Cette dernière, qui consiste à trouver le maximum absolu dans tout l'espace des paramètres, est beaucoup plus coriace numériquement. À l'heure actuelle il n'existe aucun algorithme qui garantisse la détection d'un maximum global.

La **grimpe itérée** est une approche simple permettant d'aborder l'optimisation globale, et en fait son mode de fonctionnement est déjà illustré sur la Figure 6.12. La grimpe itérée consiste simplement à faire rouler à répétition une méthode de grimpe, chaque fois avec un point de départ différent (habituellement choisi aléatoirement), et on accumule une liste des maxima ainsi détectés, et ce jusqu'à ce qu'on ne trouve plus de nouveau maximum. Décider quand arrêter est évidemment l'aspect crucial de cette approche. Dans bien des situations réelles, le calcul de la fonction f et/ou de ses dérivées peut en soi être une opération très couteuse numériquement, et donc il existe souvent des limites purement pratiques au nombre de grimpes successives qui puissent être effectuées.

On peut imaginer que la grimpe itérée puisse être accélérée si les grimpeurs ont accès à "l'information" accumulée par les grimpeurs antérieurs relative à la structure de l'espace des

```
#include <stdio.h>
#include <math.h>
/* Maximisation d'une fonction de deux variables par la grimpe        */
int main(void)
{
/* Declarations ---------------------------------------------------- */
  float fonc(float,float) ;     /* La fonction a maximiser */
  int k=0 ;itermax=100 ;        /* Compteur et maximum d'iterations  */
  float epsilon=1.e-5  ;        /* Precision requise pour le maximum */
  float f, fn, dx, dy, grad, x, y, xn, yn, pas, del ;
/* Entree/Sortie --------------------------------------------------- */
/* On fournit a l'execution le point de depart (x,y) de la grimpe    */
  printf ("SVP donner position initiale en x :") ; scanf ("%f", &x) ;
  printf ("SVP donner position initiale en y :") ; scanf ("%f", &y) ;
/* Executable ------------------------------------------------------ */
  pas = 0.1 ;                            /* Taille initiale du pas */
  f  = fonc(x,y) ; del=2.*epsilon ;
  printf ("Position initale du grimpeur: x= %f y= %f\n", x,y) ;
  while ( del > epsilon && k < itermax)  /* Boucle de grimpe */
    {
      dx = fonc(x+pas,y)-fonc(x-pas,y) ; /* composante-x du gradient */
      dy = fonc(x,y+pas)-fonc(x,y-pas) ; /* composante-y du gradient */
      grad = sqrt(dx*dx+dy*dy) ;         /* grandeur du gradient     */
      xn = x + pas*dx/grad ;             /* pas dans la direction-x  */
      yn = y + pas*dy/grad ;             /* pas dans la direction-x  */
      fn = fonc(xn,yn) ;
      if ( fn > f ) {                    /* On accepte le pas        */
          x=xn ; y=yn ;                  /* Deplacement du grimpeur  */
          del = fabs( fn-f ) ;           /* Pour le test de sortie   */
          f=fn ;
        }
      else { pas=pas/2. ; }              /* On refuse le pas */
      k += 1 ;
    }
  printf ("Maximum a: x= %f y= %f\n fonction = %f\n apres %d iterations\n",
    x,y,fn,k) ;    /* Sortie: (x,y) et valeur du max, iter. requises */
}
/* Definition de la fonction dont on recherche le maximum */
float fonc (float x, float y)
{
  float f ;                              /* Variable locale */
  f = fabs( sin(x)*sin(y)/(x*y) );       /* La fonction qu'on calcule */
  return f ;
}
```

Figure 6.10: Code C pour une grimpe de base, avec les composantes du gradients calculées par différences finies centrées. Examinez bien, et comprenez, le canevas `if ... else ...` contrôlant l'acceptation ou le rejet du pas en fonction des valeurs de la fonction produite à la nouvelle position.

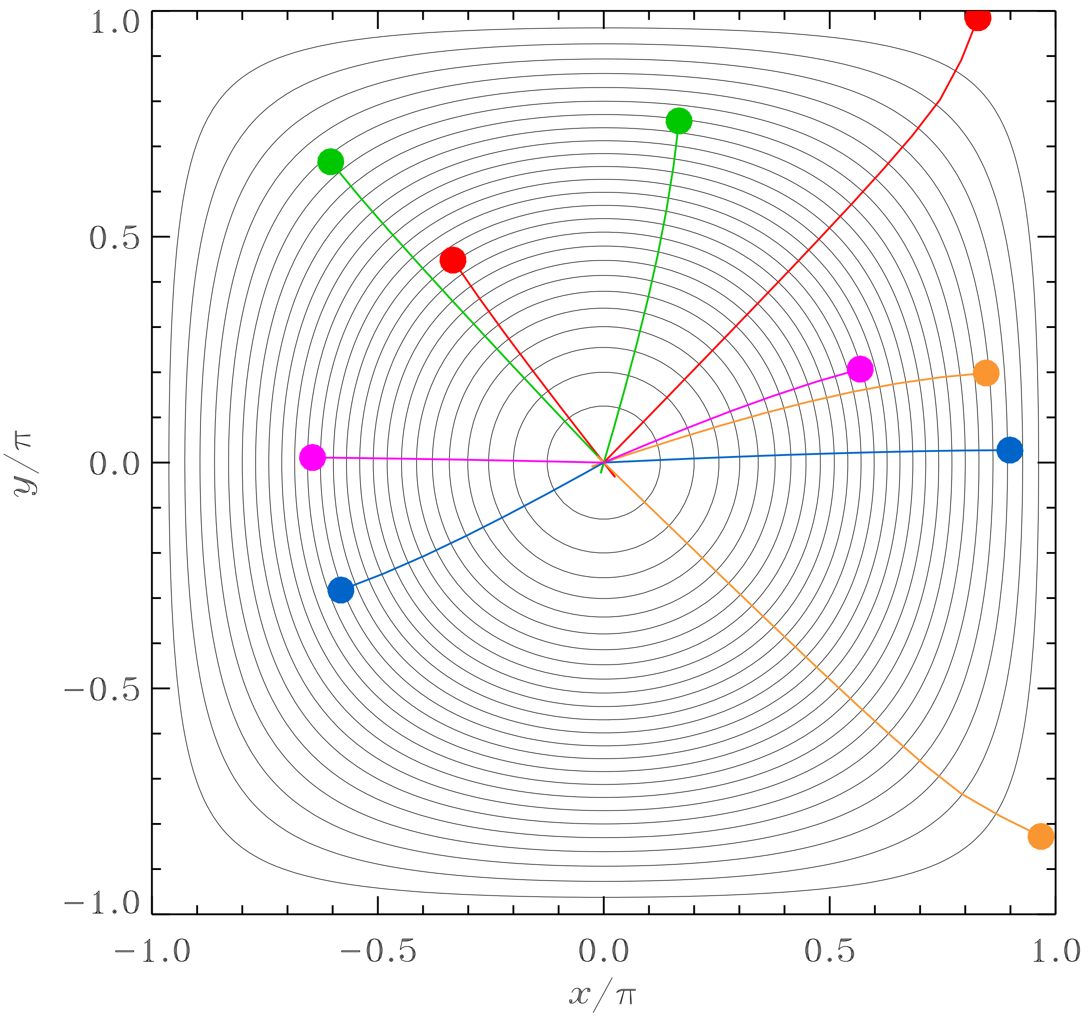


Figure 6.11: Application de notre méthode de grimpe à la recherche du pic central du patron de diffraction d'une ouverture carrée. Les traits noirs sont des courbes de niveau de la fonction à maximiser, soit l'éq. (6.15); son maximum est à $(x^*, y^*) = (0, 0)$, où $I(x, y)/I_0 = 1$. Les points colorés correspondent à 10 positions de départ (x^0, y^0) choisies aléatoirement, et les trajectoires en émanant résultent de l'application répétée des éqs. (6.17), avec ajustement du pas δ (voir texte). Quel que soit le point de départ, l'algorithme localise sans difficulté le maximum recherché.

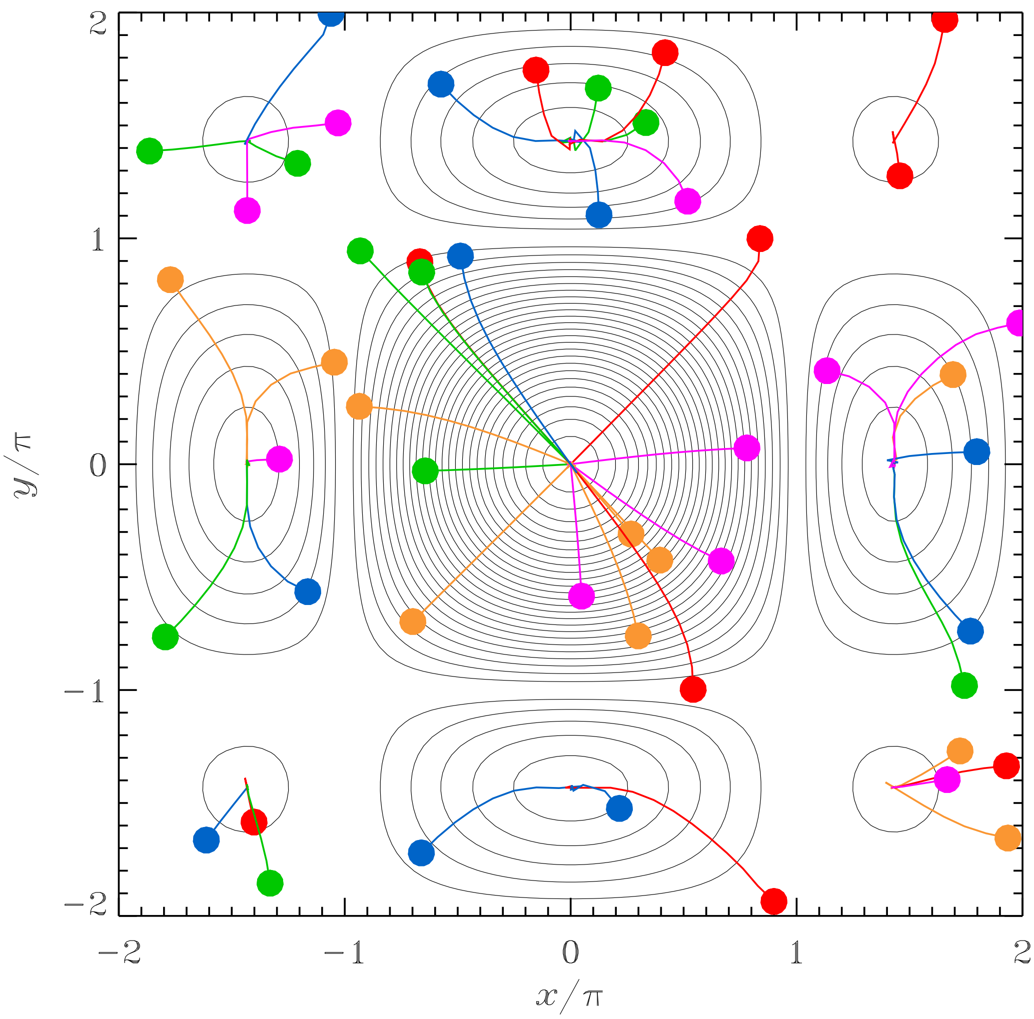


Figure 6.12: Même chose que la Figure 6.11, sauf que l'espace des paramètres couvre maintenant l'intervalle $-2\pi \leq x \leq 2\pi$, $-2\pi \leq y \leq 2\pi$, et 50 points de départ ont été choisis aléatoirement. Plusieurs de ces points de départ convergent vers un maximum secondaire.

paramètres, ou encore si les grimpeurs peuvent communiquer entre eux durant la grimpe. On verra comment au chapitre 10.

6.6 La grimpe stochastique

L'approche Monte Carlo s'applique également à la recherche d'extrema, sous la forme de la **grimpe stochastique**. L'idée générale est la même que pour la méthode de grimpe classique considérée à la §6.5, avec l'importante différence que les direction et grandeur du pas, plutôt que d'être choisies sur la base du gradient de la fonction, le sont de manière aléatoire.

Aléatoire, mais quand même pas n'importe comment. On aimerait produire des pas qui puissent avoir une taille comparable à celle de l'espace des paramètres, de manière à pouvoir "explorer" et se décoincer d'un extremum local; d'un autre coté, une bonne localisation d'un extremum demande un pas d'une taille de l'ordre du niveau de précision requis. En résumé, on veut générer une distribution de pas couvrant à la fois les grandes et les petites tailles. Une distribution ayant cette propriété est la distribution dite gaussienne (éq. (5.2) et Fig. 5.8), que nous utiliserons donc ici.

Évidemment, si les pas de la grimpe sont choisis aléatoirement, plusieurs conduiront à une diminution de la valeur de la fonction f à maximiser (dans le contexte d'une maximisation), donc seul les pas conduisant à une augmentation de f sont "acceptés". Le code C présenté à la Figure 6.13 effectue une telle grimpe stochastique. Examinez bien la série d'instructions à l'intérieur de la boucle extérieure; La première boucle `while` peut générer jusqu'à 20 pas aléatoires, et si (et seulement si!) aucun de ces pas ne conduit à une diminution de la fonction, alors la variance de la distribution gaussienne de taille de pas est diminuée de moitié, et on passe à une nouvelle itération. Mais si un des 20 pas a réduit la valeur de la fonction alors on passe à l'itération suivante sans toucher à σ. Remarquez également qu'on utilise ici la transformation de Box-Muller décrite précédemment pour produire une séquence de nombres aléatoires distribués de manière gaussienne, avec variance σ à partir de la distribution uniforme produite ici par le générateur `ran0` (voir §5.1 et Fig. 5.2). Truc à remarquer, le code de la Figure 6.13 utilise deux séquences distinctes de nombres aléatoires, chacune produite par une valeur de germe différente, pour générer les deux aléatoires uniformes r_1 et r_2 requis par la transformation de Box-Muller. Ceci est préférable afin d'assurer une indépendance statistique complète entre les deux nombres aléatoires gaussiens g_1, g_2 produits par la transformation.

La Figure 6.14 montre le résultat de 10 grimpes stochastiques dans le cadre de notre désormais familier problème de recherche du maximum central du patron de diffraction d'une ouverture carrée. Pour des points de départ choisis aléatoirement, on s'attendrait ici à ce que la grimpe classique produise un taux de succès de 1/9. Avec la grimpe stochastique, on obtient ici un très honorable 8/10. Remarquez comment certaines solutions se "coincent" pendant quelques itérations sur un extremum secondaire, mais finissent par atterrir sur le pic central suite à l'action d'un pas de taille substantiel; on voit ici l'avantage d'avoir utilisé une distribution gaussienne de taille de pas; bien que leur probabilité soit faible, de temps en temps un pas de taille substantiellement plus grand que la variance de la distribution sera produit, ce qui permet de mieux échantillonner l'espace environnant, tout en ne freinant pas trop la convergence vers l'extremum (qu'il soit local ou global).

Pour ce problème d'optimisation de dimension 2, la grimpe stochastique n'est pas particulièrement avantageuse par rapport à la grimpe itérée, du point de vue du nombre requis d'évaluations de $f(x)$ pour en arriver à un niveau de précision donné. Comme dans le cas de l'intégration, l'approche Monte Carlo à l'optimisation ne devient vraiment compétitive que pour des espace de recherche de plus haute dimensionalité. La situation est comparable à celle que nous avions rencontré avec le calcul des intégrales par Monte Carlo; évaluer numériquement les composantes du gradient d'une fonction de N variables via une différence finie centré d'ordre deux requiert 2^N évaluations de la fonction.

```
#include <stdio.h>
#include <math.h>
#define PI 3.1415926536
/* Maximisation d'une fonction f(x,y) par grimpe stochastique       */
int main(void)
{
/* Declarations ----------------------------------------------------- */
  float ran0( long *idum) ;   /* Generateur de nombres aleatoires    */
  float fonc( float, float ) ;/* La fonction a maximiser             */
  int k=0, itermax=200 ;      /* Compteur et maximum de pas          */
  int j=0, trymax=20 ;        /* Compteur et maximum d"essais        */
  float epsilon=1.e-6  ;      /* Precision requise pour le maximum   */
  float r1, r2, g1, g2, f, fn, x, y, xn, yn, pas, del ;
  long germe1, germe2 ;
  long *p_germe1, *p_germe2 ;
/* Executable ------------------------------------------------------- */
  x=3.0 ; y=5.0 ; germe1=123 ; germe2=456 ;       /* Initialisations */
  p_germe1=&germe1 ; p_germe2=&germe2 ;   /* Pointeurs pour germes   */
  pas = 1.0 ;                             /* Taille initiale du pas  */
  f  = fonc(x,y) ; del=2.*epsilon ;
  printf ("Position initale du grimpeur: x= %f y= %f\n", x,y) ;
  fn =f/2. ;
  while ( del > epsilon && k < itermax) { /* Boucle de grimpe        */
    j=0 ;
    while ( fn <= f && j < trymax ) {     /* Boucle d'essais de pas  */
        r1 = ran0(p_germe1) ;             /* Aleatoires entre 0 et 1 */
        r2 = ran0(p_germe2) ;
        g1 = sqrt(-2.*log(r1))*cos(2*PI*r2) ; /* Box-Muller          */
        g2 = sqrt(-2.*log(r1))*sin(2*PI*r2) ; /* eqs. (6.19)-(6.20)  */
        xn = x + pas*g1 ;                 /* pas en x                */
        yn = y + pas*g2 ;                 /* pas en y                */
        fn = fonc(xn,yn) ;
        j += 1 ;
    }
    if ( fn > f ) {                       /* On accepte le pas       */
      x=xn ; y=yn ;                       /* Deplacement du grimpeur */
      del = fabs( fn-f ) ;                /* Pour le test de sortie  */
      f=fn ;
    }
    else { pas=pas/2. ; }                 /* On refuse le pas */
    k += 1 ;
  }
  printf ("Maximum a: x= %f y= %f\n fonction = %f\n apres %d iterations\n",
    x,y,fn,k) ;    /* Sortie: (x,y) et valeur du max, iter. requises */
}
```

Figure 6.13: Code C pour la grimpe stochastique, que vous devriez comparer attentivement au code C de la Figure 6.10 pour la grimpe classique; mis à part quelques initialisations, la seule différence se trouve au niveau du calcul du pas. Notez que ceci implique une seconde boucle `while` qui calcule ici 20 pas stochastiques avant de refuser le pas et en réduire la grandeur. Ce code requiert l'inclusion de la fonction `ran0` (voir Figure 5.2).

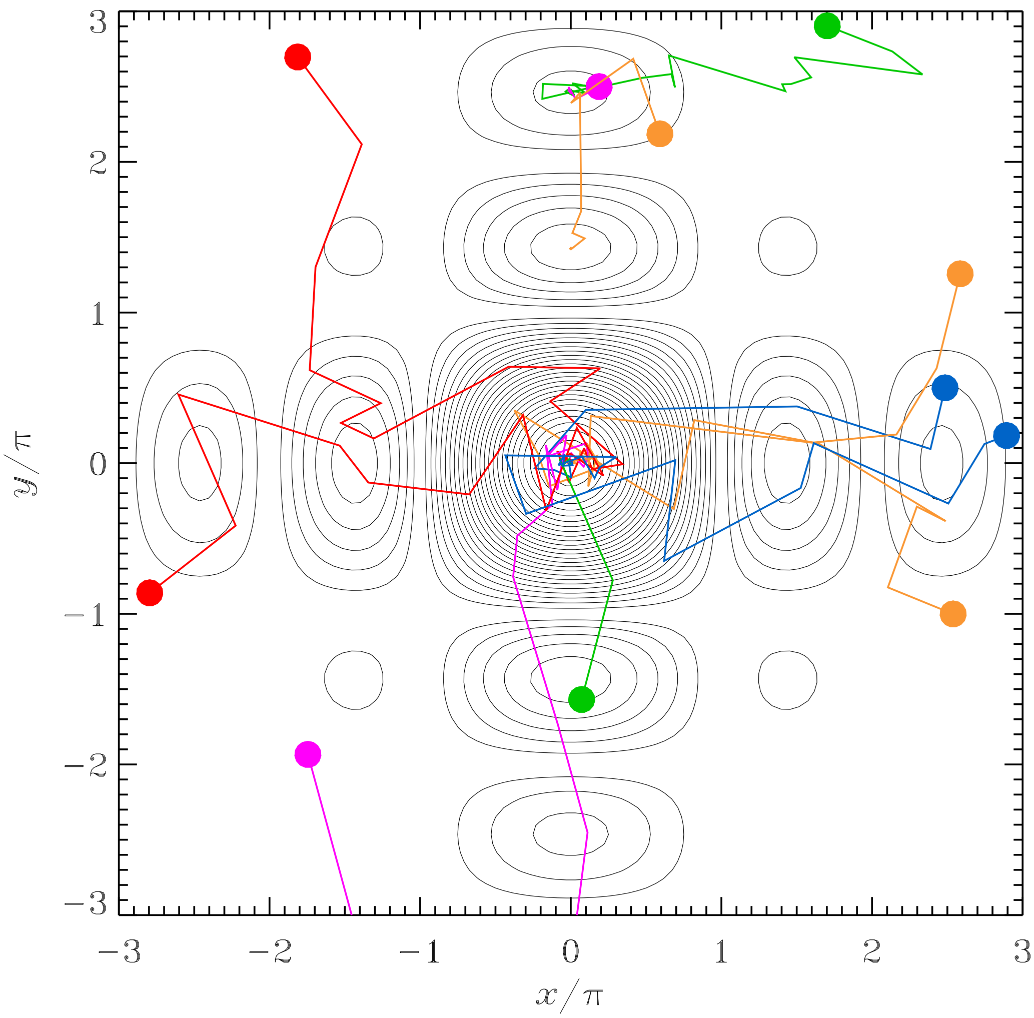


Figure 6.14: Recherche du maximum central du patron de diffraction 2D par grimpe stochastique (ici itérée). Remarquez comment certaines grimpes se coincent temporairement sur un extremum secondaire, mais parviennent à s'en dépêtrer suite à la production d'un pas de longueur substantielle. Le taux de succès est ici de 8/10, ce qui se compare très avantageusement au 15/50 de la grimpe itérée (voir Figure 6.12).

Exercices:

1. Le code C minimal pour la bisection présenté à la §6.2 peut être restructuré de manière telle qu'un seul appel à la fonction `fonc(x)` soit requis par itération. Faites-le.
2. Écrivez un petit bout de code C qui, à partir d'un vecteur `x` contenant N éléments ordonnés de manière croissante, utilise l'idée de la bissection pour localiser la plage d'interpolation associée à une valeur x^*. Démontrez que le nombre d'itération requis pour trouver cette plage varie en moyenne comme $\log_2 N$, dans la limite où N est grand.
3. Écrivez une version du code C pour la recherche de racine par la méthode de Newton qui s'assure que la racine demeure à l'intérieur d'un intervalle $[x_1, x_2]$ spécifié par l'usager (comme pour la bissection).
4. La méthode de grimpe, telle que codée ci-dessus, risque fort de vous péter au nez vers la fin du processuss de convergence, puisque très près du maximum on s'attend à ce que $\nabla f \to 0$, ce qui conduira à des trucs du genre zéro-divisé-par-zéro. Reécrivez l'algorithme en terme d'un *angle* décrivant l'orientation du gradient dans le plan $[x, y]$. Est-ce-que celà fonctionne mieux que l'algorithme original? Pourquoi?
5. Modifiez le code C pour la grimpe stochastique (Figure 6.13) pour en faire un code de grimpe stochastique *itérée*, avec points de départ de chaque grimpe choisis aléatoirement.

Bibliographie:

Sur la recherche de racine et l'optimisation, je ne peux vraiment faire guère mieux que vous recommander les chapitres 9 et 10 de l'ouvrage *Numerical Recipes*. La principale omission au chapitre 10, soit l'usage des algorithme évolutifs pour l'optimisation, sera traité au dernier chapitre de ces notes de cours.

Pour tout ce qui concerne la modélisation de données, les erreurs de mesure, les χ^2, etc, je trouve l'ouvrage suivant absolument magnifique à tous les points de vue, allant de la photo en couverture jusqu'aux aspects plus "philosophique" du sujet:

Taylor, J.R., *An introduction to error analysis*, University Science Books (1982).

Chapitre 7

Marche aléatoire et diffusion

7.1 Les processus stochastiques

Le problème de l'approche à l'équilibre considéré à la §5.5 offre un exemple d'un système physique dont le comportement est contrôlé par un processus dit **stochastique**. Bien qu'en principe les collisions faisant passer les particules par le petit trou puissent se modéliser à l'aide de la mécanique Newtonienne, en pratique le nombre de particules à suivre, et de collisions à calculer, dépasse de vraiment très loin tout ce que vous seriez jamais en mesure de calculer même si l'on mettait tous les processeurs du système ESIBAC à votre disposition pendant une semaine. Ou même un million d'années[1]. Souvent, la seule approche pratique est de simuler le comportement du système pour en extraire des mesures qui, bien que de nature statistique, n'en sont souvent pas moins d'une grande utilité.

Dans ce chapitre, nous allons nous concentrer sur un processus stochastique prototypique, soit la marche aléatoire (§7.2). Nous examinerons ensuite le lien profond existant entre la marche aléatoire et la diffusion (§7.3), un processus physique de grande importance en sciences environnementales. Nous poursuivrons en examinant une variante de la marche aléatoire, soit la marche sur réseau (§7.4), qui nous permettra d'aborder les processus d'agrégation (§7.5), prototypiques de l'émergence de structures cohérentes aux grandes échelles à partir de mécanismes stochastiques opérant aux petites échelles. La caractérisation des ces structures (§7.6) nous entrainera finalement dans le monde déroutant des structures fractales (§§7.7 et 7.8).

7.2 Marche aléatoire

Dans sa forme la plus simple, la marche aléatoire décrit la variation d'un déplacement $\mathbf{D}_n$ suite à une séquence de n pas $\mathbf{s}_n$ de longueur constante s mais orientés aléatoirement dans l'espace. Nous considérerons d'abord ici une version unidimensionnelle de ce processus, où le pas est contraint à $s = \pm 1$ le long d'une droite, chacun des deux pas possibles ($s = +1$ ou $s = -1$) étant équiprobable et aucunement influencé par la valeur du pas précédent. On parle alors d'un **processus stochastique sans mémoire**. Par exemple, il est très tard l'après-midi du jour de l'initiation, vous sortez avec difficulté des toilettes et tentez de vous rendre à Planck, passssséláàkélamooolssondraille...; si les festivités sont suffisamment avancées, un pas dans la direction Est (vers la porte du D-460) est aussi probable qu'un pas dans la direction Ouest (vers la Planck). Dénotons par s_n la valeur (± 1) du n-ième pas, et par D_n le déplacement (attention, pas la distance!) atteint depuis le premier pas. On aura de toute évidence:

$$D_n = D_{n-1} + s_n \ , \qquad n = 1, 2, 3... \tag{7.1}$$

[1]Vous feriez aussi face à d'autres problèmes très fondamentaux, sur certains desquels nous reviendrons au chapitre 9 de ces notes.

(la distance totale parcourue par chaque marcheur est évidemment $n \times s$, pour un pas de longueur s). Le carré du déplacement net au pas n est donnée par:

$$D_n^2 = (D_{n-1} + s_n)^2 = D_{n-1}^2 + s^2 + 2D_{n-1}s_n \ . \tag{7.2}$$

Considérons non pas un, mais un ensemble d'étudiant(e)s tentant laborieusement de rejoindre la Planck, et dénotons par des crochets $\langle ... \rangle$ la moyenne effectuée sur l'ensemble des marcheurs; ceci s'appelle une **moyenne d'ensemble**, et est défini selon

$$\langle x \rangle = \frac{1}{M} \sum_{m=1}^{M} x(m) \ , \tag{7.3}$$

où $x(m)$ dénote la valeur de la mesure x pour le m-ième membre de l'ensemble, ici de taille M. Donc, par exemple, $\langle D_n \rangle$ représenterait le déplacement moyen du groupe sortant des toilettes. Il s'agit ici d'un opérateur linéaire, satisfaisant les propriétés:

$$\langle x + y \rangle = \langle x \rangle + \langle y \rangle \ , \qquad \langle ax \rangle = a \langle x \rangle \ , \tag{7.4}$$

où a est un coefficient numérique quelconque. Appliquons cet opérateur à l'éq. (7.2):

$$\left\langle D_n^2 \right\rangle = \left\langle D_{n-1}^2 + s^2 + 2D_{n-1}s_n \right\rangle = \left\langle D_{n-1}^2 \right\rangle + \left\langle s^2 \right\rangle + 2\langle D_{n-1}s_n \rangle \ , \tag{7.5}$$

en vertu de la linéarité de notre opérateur de moyenne d'ensemble. Si la marche est vraiment aléatoire, et qu'aucune communication n'existe entre les marcheurs leur permettant de synchroniser leurs titubations, chaque marche est statistiquement indépendante des autres; autrement dit, l'ensemble des valeurs ± 1 du pas s_n pour l'ensemble des marcheurs a la forme d'une distribution discrète pour deux valeurs, $+1$ et -1, qui sont équiprobables *et* ne présentent aucune corrélation avec les D_{n-1}, comme l'on s'y attend d'un processus stochastique sans mémoire. Ceci implique donc que

$$\langle D_{n-1}s_n \rangle = \langle D_{n-1} \rangle \langle s_n \rangle = 0 \ . \tag{7.6}$$

Comprenez bien pourquoi, c'est vital pour tout ce qui suit. L'éq. (7.5) devient alors:

$$\left\langle D_n^2 \right\rangle = \left\langle D_{n-1}^2 \right\rangle + s^2 \ . \tag{7.7}$$

Si l'on suppose maintenant que $D_0 = 0$ (i.e., l'origine du système de coordonnées est à la porte des toilettes), alors on aura:

$$\begin{aligned} \left\langle D_1^2 \right\rangle &= s^2 \ , & (7.8) \\ \left\langle D_2^2 \right\rangle &= \left\langle D_1^2 \right\rangle + s^2 = 2s^2 \ , & (7.9) \\ \left\langle D_3^2 \right\rangle &= \left\langle D_2^2 \right\rangle + s^2 = 3s^2 \ , & (7.10) \\ ... &= ... & (7.11) \end{aligned}$$

et donc après n pas:

$$\left\langle D_n^2 \right\rangle = n\, s^2 \ . \tag{7.12}$$

Maintenant, si l'on interprète le décompte n des pas comme une variable temps (genre, un pas chaque trois secondes), cette expression nous indique que le déplacement quadratique moyen augmente linéairement avec le temps, d'où:

$$\sqrt{\langle D_n^2 \rangle} = s\sqrt{n} \ . \tag{7.13}$$

Ceci s'appelle la **moyenne quadratique** du déplacement. Le point crucial de cette petite analyse est évidemment l'éq. (7.6); ceci est valide pour un processus stochastique sans mémoire et seulement dans la limite d'un nombre de marcheurs tendant vers l'infini. Et l'infini, c'est

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define NM   100
#define NPAS 1000
/* Marche aleatoire en 1D: NM marcheurs font NPAS pas de longueur */
/* unitaire, et le deplacement quadratique moyen est calcule      */
int main(void)
{
/* Declarations -------------------------------------------------- */
  int pas, x, j, k, somme=0 ;
/* Executable ---------------------------------------------------- */
  for (j=0 ; j<NM ; j++)              /* boucle sur les marcheurs */
    {
      x=0 ;                    /* Tous les marcheurs partent a x=0 */
      for (k=0 ; k<NPAS ; k++)              /* boucle sur les pas */
        {
          pas=2*floor( 2.*rand()/RAND_MAX )-1 ;   /* pas de +/- 1 */
          x += pas ;                     /* Le marcheur se deplace */
        }
      somme += x*x ; /* on accumule les deplacements quadratiques */
    }
   printf ("Deplacement quadratique moyen:  %f\n", (1.*somme/NM) ) ;
}
```

Figure 7.1: Code C pour la marche aléatoire en 1D. Son seul aspect délicat est la production d'un pas aléatoire prenant les valeurs (entières) de -1 ou $+1$ (voir texte). Le code effectue également une somme cumulative des positions quadratiques x^2 à la sortie de la boucle intérieure, ce qui permet, à la sortie de la boucle extérieure, le calcul du déplacement quadratique moyen pour l'ensemble des `NM` marcheurs après `NPAS` pas.

un gros chiffre; beaucoup plus grand que le nombre d'Avogadro, beaucoup plus grand que le nombre d'atomes d'Hydrogène dans l'univers, beaucoup plus grand que le plus grand chiffre que vous pouvez possiblement imaginer (surtout qu'il ne faut qu'y additionner 1 pour en produire un encore plus grand...). Bref, peut-on vraiment s'attendre à ce que l'éq. (7.13) tienne, et si oui à quel niveau de précision?

Nous tenterons ici de répondre à cette question en *simulant* la marche aléatoire unidimensionnelle, ce qui ne requiert finalement qu'un générateur de nombre aléatoire qu'on force à produire un chiffre égal à soit $+1$ ou -1. Le petit code C listé à la Figure 7.1 montre comment faire à partir de notre bon vieux générateur générique `rand()`. Le canevas global du code est simple: une boucle extérieure sur le nombre `NM` de marcheurs, et une boucle intérieure calculant les `NPAS` de chacun de ces marcheurs. La seule partie corsée est le calcul d'un pas pouvant valoir soit $+1$, soit -1. Ceci est effectué à partir du générateur générique `rand()`. Séquentiellement:

1. `1.*rand()/RAND_MAX` est un aléatoire réel distribué uniformément entre 0 et 1; la valeur `RAND_MAX` est incluse dans la librairie < `stdlib.h` >, qui doit donc être incluse en en-tête au programme;
2. `2.*rand()/RAND_MAX` est donc un aléatoire réel distribué uniformément entre 0 et 2;
3. La fonction C `floor` tronque un réel à l'entier inférieur le plus rapproché. Cette fonction se trouve dans la librairie < `math.h` >, qui doit donc être incluse en en-tête au programme. Donc `floor( 2.*rand()/RAND_MAX )` est un entier qui vaut soit zéro (si

`2.*rand()/RAND_MAX` tombait entre 0 et 1), ou un, (si `2.*rand()/RAND_MAX` tombait entre 1 et 2);

4. En conséquence de quoi, `2*floor( 2.*rand()/RAND_MAX )-1` est un entier qui vaut soit -1, soit $+1$, de manière équiprobable. Voilà!

La Figure 7.2 montre 10 trajectoires de 10^4 pas résultant d'une telle marche aléatoire unidimensionnelle, débutant à $x = 0$. On peut déjà remarquer que les courbes définissent un genre de "cone" plus ou moins symétrique par rapport à $x = 0$ (trait pointillé), et que certains marcheurs réussissent à se retrouver vraiment pas loin de $x = 0$ même après 10^4 pas!

On peut mieux quantifier tout ça en construisant, à chaque pas, la *distribution* des déplacements associés à l'ensemble des marcheurs. Ceci est illustré à la Figure 7.3 (histogrammes) pour quatre temps spécifiques, tel qu'indiqué, les distributions étant construites ici à l'aide de 1000 marcheurs. Les distributions sont concentrées près de $x = 0$ au début et s'étalent en x à mesure que le temps s'écoule. Le fait qu'elles demeurent symétriques par rapport à $x = 0$ indique que *le déplacement moyen $\langle D_n \rangle$ est nul!*. Le fait que les distributions demeurent maximales à $x = 0$ même après un grand nombre de pas indique que *le déplacement le plus probable est toujours zéro!*.

Les distributions de la Figure 7.3 ont une allure qui devrait vous rappeler quelque chose... (sinon voir la Figure 5.8...). Pourrait-il s'agir d'une gaussienne? bin oui; les traits minces représentent une série de fonctions gaussiennes collées aux différents histogrammes en ajustant les amplitude et variance (σ dans l'éq. (5.2)) par minimisation du χ^2 (effectivement). L'ajustement est excellent dans tous les cas! Ayant ainsi extrait σ à différents pas de temps, on peut examiner comment σ varie avec t. Ceci est illustré à la Figure 7.4. Le graphique est ici de type log-log, et dans ce graphique les σ se distribuent parfaitement le long d'une droite de pente 1/2; autrement dit, on peut écrire:

$$\log(\sigma) = b + \frac{1}{2}\log(n) = b + \log(n^{1/2}) \ , \tag{7.14}$$

d'où

$$10^{\log(\sigma)} = 10^{b+\log(n^{1/2})} = 10^b 10^{\log(n^{1/2})} \ , \qquad \rightarrow \qquad \sigma(n) = \sigma_0 n^{1/2} \ , \tag{7.15}$$

avec $\sigma_0 \equiv 10^b$. Ceci implique que, collectivement, les marcheurs composant la distribution s'éloignent de l'origine à une vitesse moyenne $\propto \sqrt{n}$, en accord avec l'éq. (7.13). La marche aléatoire vous offre donc, en quelque sorte, une alternative (coûteuse) à la transformation de Box-Muller pour la génération de nombres aléatoires distribués de manière Gaussienne!

Ces résultats ne sont pas du tout restreints à la marche aléatoire en une dimension spatiale. En plus d'une dimension les déplacement et pas deviennent des quantités vectorielles, et on remplacerait l'éq. (7.1) par:

$$\mathbf{D}_n = \mathbf{D}_{n-1} + \mathbf{s}_n \ , \qquad n = 1, 2, 3... \tag{7.16}$$

où le pas $\mathbf{s}_n$ est de longueur unitaire mais est orienté aléatoirement dans l'espace. Le carré du déplacement au pas n devient:

$$D_n^2 = (\mathbf{D}_{n-1} + \mathbf{s}_n) \cdot (\mathbf{D}_{n-1} + \mathbf{s}_n) = D_{n-1}^2 + s^2 + 2\mathbf{D}_{n-1} \cdot \mathbf{s}_n \ . \tag{7.17}$$

Si le pas est vraiment orienté aléatoirement, alors moyenné sur en ensemble de marches aléatoires on aura $\langle \mathbf{D}_{n-1} \cdot \mathbf{s}_n \rangle = 0$, ce qui conduit de nouveau directement à:

$$\langle D_n^2 \rangle = \langle D_{n-1}^2 \rangle + s^2 \ . \tag{7.18}$$

La suite est exactement comme auparavant, et aboutit de nouveau à l'éq. (7.12). C'est une propriété remarquable de la marche aléatoire: quelle que soit la dimensionalité du problème, le déplacement quadratique moyen varie toujours comme $n^{1/2}$.

Le code C de la Figure 7.1 pour la marche aléatoire en 1D est trivial à modifier pour passer à la marche aléatoire 2D, la boucle intérieure devenant simplement:

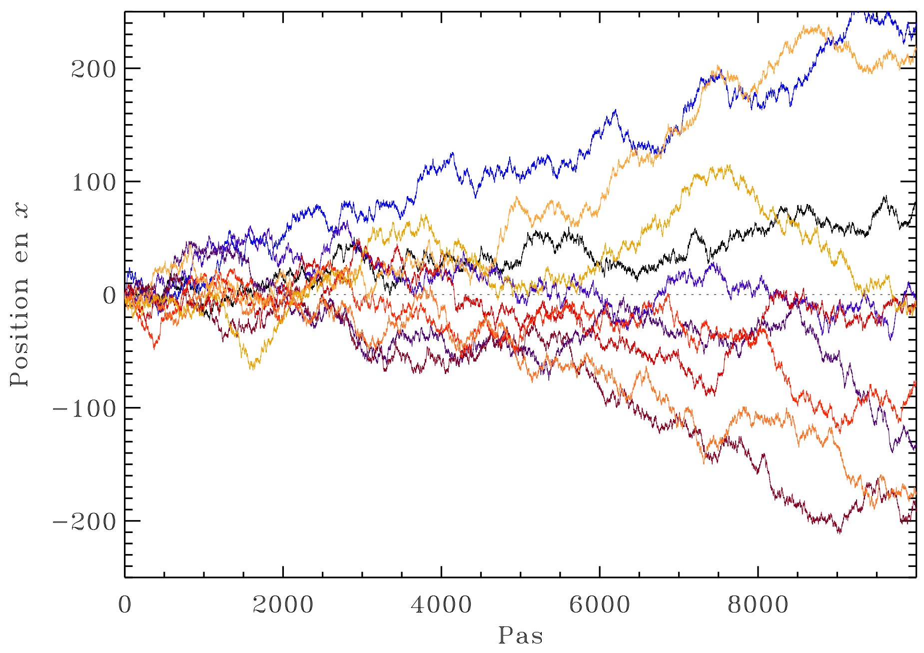


Figure 7.2: Variation en fonction du temps (mesuré en pas n) du déplacement D_n de 10 marcheur aléatoires en 1D, débutant tous à $x = 0$. Bien que plusieurs trajectoires réussissent à s'éloigner de zéro et "progresser" soit dans les directions positive ou négative le long de l'axe de la marche, plusieurs se retrouvent très près de zéro même après 10^4 pas.

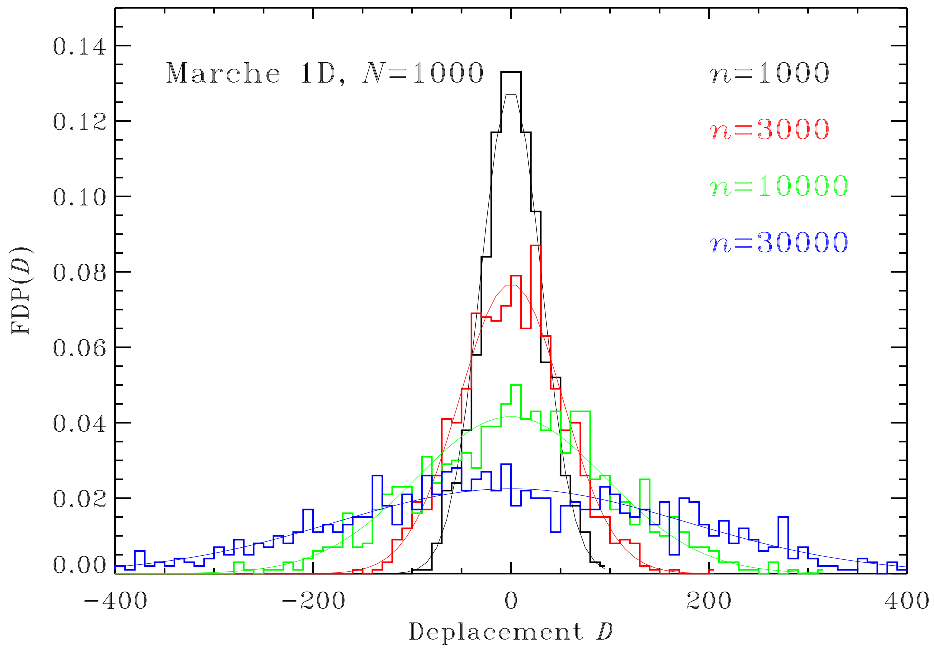


Figure 7.3: Étalement avec le temps de la distribution (histogramme) des déplacements associés à 1000 marcheurs aléatoires en 1D, débutant tous leur marche à $x = 0$. Les traits fins et continus sont des distributions gaussiennes ajustées aux histogrammes correspondants.

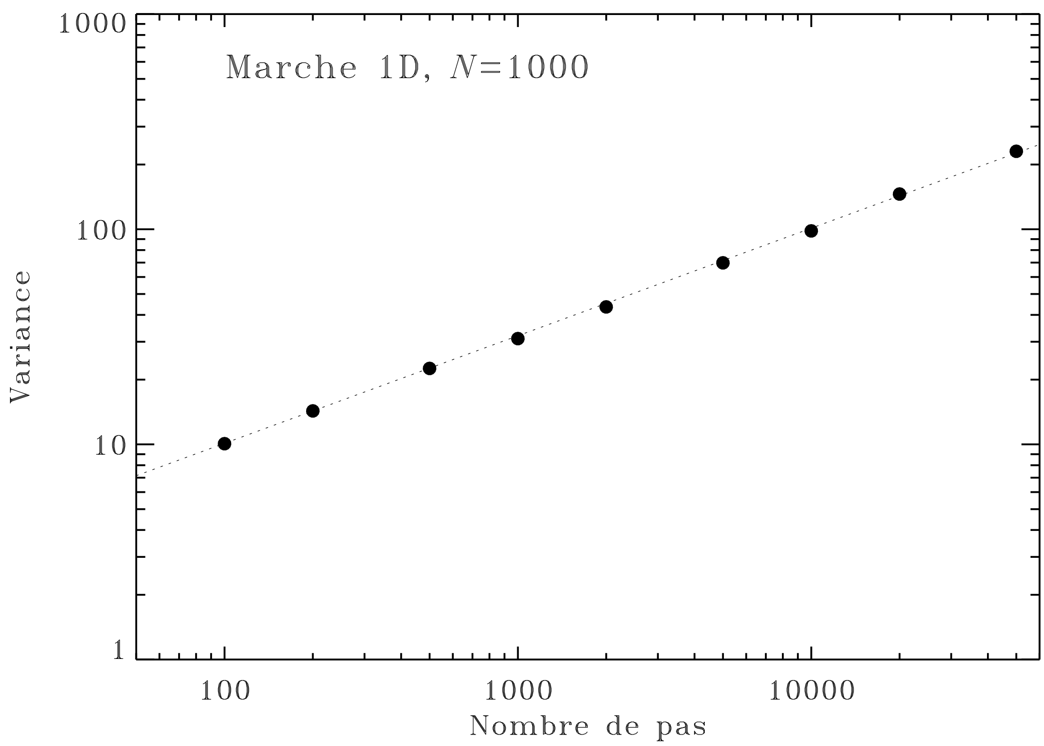


Figure 7.4: Augmentation de la variance de la distribution des déplacements d'un groupe de 1000 marcheurs aléatoires en 1D, en fonction du temps, mesuré ici en nombre de pas n. Le trait pointillé correspond à une dépendance en $\sqrt{n}$ collée à la première mesure de variance à $n = 100$.

```
x=0 ; y=0 ;     /* tous les marcheurs partent a (x,y)=(0,0) */
for (k=0 ; k<NPAS ; k++)          /* boucle sur les pas     */
  {
    ang=2.*PI*rand()/RAND_MAX ;   /* angle entre 0 et 2 pi  */
    x += cos(ang) ;               /* Le marcheur se deplace */
    y += sin(ang) ;
  }
```

(ceci présuppose que la valeur de π a été convenablement définie en en-tête au programme via un `#define PI 3.1415926536`). Évidemment le déplacement quadratique est maintenant donné par `x*x + y*y`.

La Figure 7.5 montre les quelques premiers pas d'une marche aléatoire en 2D, débutant à $(x, y) = (0, 0)$. Le trait en tirets indique le vecteur-déplacement net après le dix-huitième pas. Ce vecteur déplacement sous-tend ici un angle θ_{18} par rapport à un système de coordonnées cartésiennes dont l'origine est à $(x, y) = (0, 0)$. Le dix-neuvième pas de la marche fera atterrir notre marcheur quelque part sur le cercle de rayon unitaire centré sur sa position au dix-huitième pas, à un angle aléatoire dont la valeur mesurée par rapport à un système cartésien local peut être n'importe quoi entre 0 et 2π. Dénotons la valeur de cet angle par α_{19}; l'angle entre le vecteur-déplacement $\mathbf{D_{18}}$ et le pas $\mathbf{s}_{19}$ sera donc donné par $\alpha_{19} - \theta_{18}$, et donc on aura:

$$\mathbf{D}_{18} \cdot \mathbf{s}_{19} = D_{18} s_{19} \cos(\alpha_{19} - \theta_{18}) \; ; \tag{7.19}$$

à ce stade l'angle θ_{18} est fixé, donc une constante entre 0 et 2π; et α_{19} est aléatoire distribué

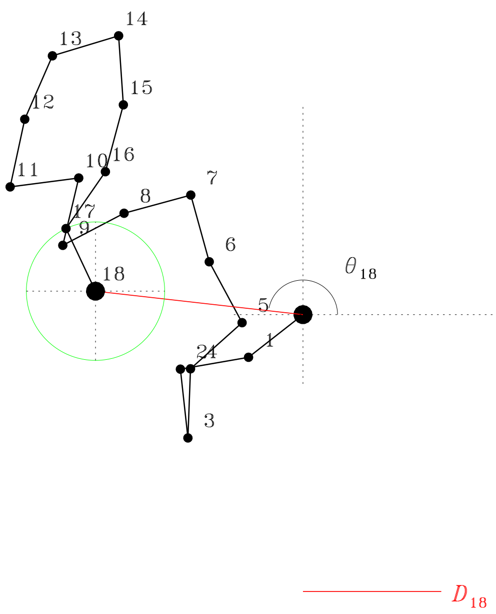


Figure 7.5: Les dix-huit premiers pas d'une marche aléatoie en deux dimensions spatiales et débutant à $(x, y) = (0, 0)$. Le dix-neuvième pas atterrira quelque part sur le cercle de rayon unitaire centré sur la position au pas 18, mais se fera à un angle dont la valeur est complètement aléatoire et indépendante de la grandeur et orientation du vecteur-déplacement au dix-huitième pas (segment de droite en rouge).

uniformément entre 0 et 2π. Donc, et en vertu de la périodicité des fonctions trigonométriques, l'angle $\alpha_{19} - \theta_{18}$ sera aussi uniformément distribué entre 0 et 2π, et son cosinus a donc autant de chance d'être positif que négatif, ce qui assure qu'une moyenne d'ensemble du produit scalaire ci-dessus sera toujours nulle.

La Figure 7.6 illustre quelques marches aléatoires en 2D, chacune de 100 pas. Le cercle tracé a un rayon égal à la taille du déplacement quadratique moyen attendu, soit ici $R = \sqrt{\langle D_{100} \rangle} = 10s$. Ca colle!

7.3 Diffusion = marche aléatoire

Considérons la variation suivante sur le thème de la marche aléatoire en une dimension spatiale: chaque marcheur, comme auparavant, fait un pas $s = \pm 1$, mais la marche est limitée par deux "murs" à $x = 100$ et $x = -100$, et les marcheurs frappant le mur "rebondissent" d'un pas dans la direction opposée. De plus, choisissons une condition initiale où tous les marcheurs débutent à des positions distribuées de manière aléatoire mais statistiquement uniformes dans le sous-intervalle $-100 < x \leq 0$, soit dans la moitiée gauche du domaine. Tous les marcheurs font leur pas de manière synchronisée dans le temps, mais sinon de manière totalement découplée l'un de l'autre.

La Figure 7.7 montre une séquence temporelle d'histogrammes décrivant la distribution des positions des marcheurs à mesure que la marche aléatoire progresse. La distribution, initialement très raide à $x = 0$, s'étale graduellement avec le temps, mesuré en pas de marche aléatoire.

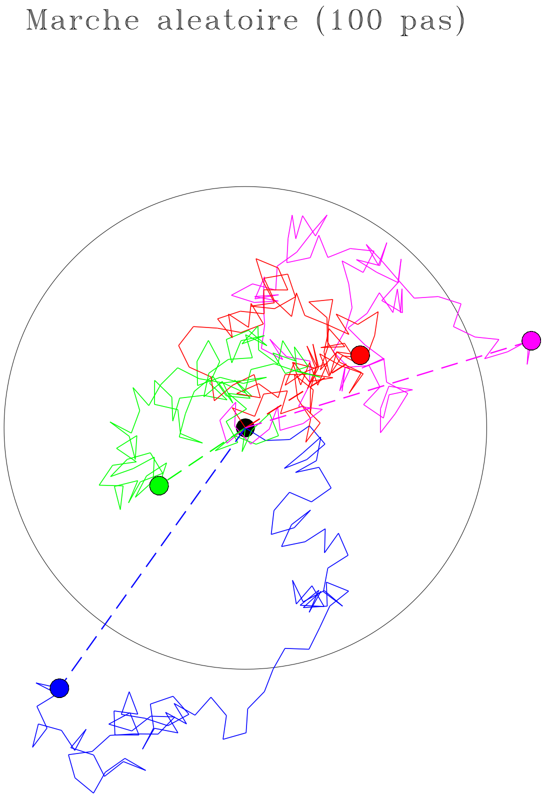


Figure 7.6: Quatre marches aléatoires de $n = 100$ pas, en deux dimensions spatiales. Tous les marcheurs débutent au point noir au centre, et le cercle correspond à un déplacement $D \equiv \sqrt{\mathbf{D} \cdot \mathbf{D}} = \sqrt{n} = 10$.

Cet étalement résulte du fait que plus de marcheurs traversent de la moitié gauche du domaine vers la moitié droite que dans le sens inverse, en raison du simple fait qu'il y a initialement beaucoup plus de marcheurs du coté gauche. Ce n'est que lorsqu'il y aura autant de marcheurs de chaque coté de $x = 0$ que le système pourra devenir statistiquement stationnaire.

La Figure 7.8 montre l'évolution en fonction du temps du nombre de marcheurs situés dans les intervalles $[-100, 0]$ (trait noir) et $[0, +100]$ (trait gris). Au début de la simulation le premier diminue rapidement tandis que le second augmente évidemment tout aussi rapidement puisque leur somme doit demeurer constante (ici 1000 marcheurs). Après quelques dizaines de milliers d'itérations, les nombres de marcheurs de chaque coté de $x = 0$ sont devenus essentiellement égaux, mis à part des petites fluctuations par rapport à la valeur moyenne (500 marcheurs). J'espère que vous commencez à voir une similitude avec la situation considérée à la §5.5 dans le contexte de notre modélisation Monte Carlo de l'approche à l'équilibre... et que vous pouvez conséquemment anticiper que si la simulation était effectuée avec un plus grand nombre de marcheurs, ces fluctuations diminueraient d'un facteur proportionnel à la racine carrée du nombre total de marcheurs...

Cet étalement graduel d'une concentration de “particules” décrivant chacune une marche aléatoire est représentatif des processus dits de **diffusion**. Ce terme a déjà été introduit à la §5.5 dans le contexte de notre discussion de l'approche à l'équilibre d'un système de deux boites contigues communiquant via un petit trou. On y avait mentionné que ce qui fait que des particules réussissent à traverser d'une boite à l'autre, ce sont les fréquentes collisions inter-particules et/ou avec les parois des boites.

Comment réconcilier cet énoncé avec le fait que dans notre discussion de la marche aléatoire à date, les marcheurs se déplacent complétement indépendamment les uns des autres, i.e., sans

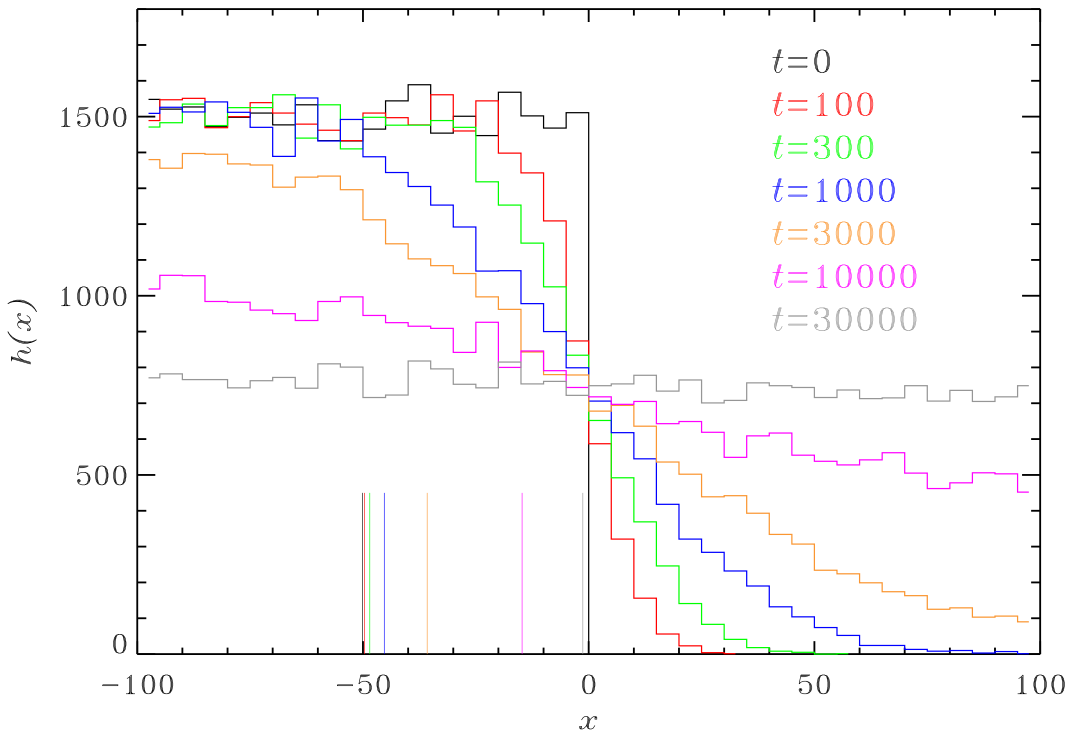


Figure 7.7: Évolution temporelle de la distribution de 30000 marcheurs aléatoires 1D confinés à l'intervalle $[-100 \leq x \leq +100]$, et initialement tous localisés aléatoirement dans le sous-intervalle $[-100 \leq x \leq 0]$. Les lignes verticales le long de l'abcisse indiquent la position moyenne des marcheurs au temps correspondant, tel que codé par la couleur du trait.

collisions ou autres interactions? Il faut revenir à l'image facitieuse des molécules de Brut 45 Magnum répendant leur odeur dans une pièce; si les molécules nauséabondes diffusent dans la pièce, c'est par une forme de marche aléatoire où la direction de chaque pas est déterminée par les aléas des collisions inter-particules, leur longueur par la distance interparticule moyenne dans la pièce, et la durée du pas par le temps moyen s'écoulant entre deux collisions successives. Mais, et c'est là la clef de notre paradoxe apparent, puisque les molécules d'air (essentiellement N_2 et O_2) sont beaucoup plus nombreuses que celles de Brut 45 Magnum (heureusement d'ailleurs), ces dernières feront des collisions presqu'uniquement avec O_2 ou N_2, et presque jamais avec une autre molécule de Brut 45. Donc la marche aléatoire de chaque molécule de Brut est complètement découplée des autres. On dirait ici du Brut 45 Magnum qu'il est un **contaminant trace** diffusant dans l'air (Voir aussi Figure 7.9).

Revenons à notre marche aléatoire en 1D confinée à l'intervalle $-100 \leq x \leq 100$ (Figs. 7.7 et 7.8). Examinons ce qui se passe à une position x_0 quelconque; seules les particules situées dans l'intervalle $x_0 - |s| < x < x_0 + |s|$ ont une chance de traverser x au prochain pas de marche, mais seulement si le pas se fait dans la bonne direction ($s = +1$ pour les particules situées dans $x_0 - |s| < x < x_0$, et $s = -1$ pour celles dans dans $x_0 < x < x_0 + |s|$). Pour des directions équiprobables, la moitié des particules dans chaque intervalle feront ce pas dans la "bonne" direction. Donc le nombre *net* δN de particules traversant la position x_0 vers la droite sera donné par quelque chose comme

$$\delta N(x_0) = \frac{1}{2}N(x_0 - |s|) - \frac{1}{2}N(x_0 + |s|) \ . \tag{7.20}$$

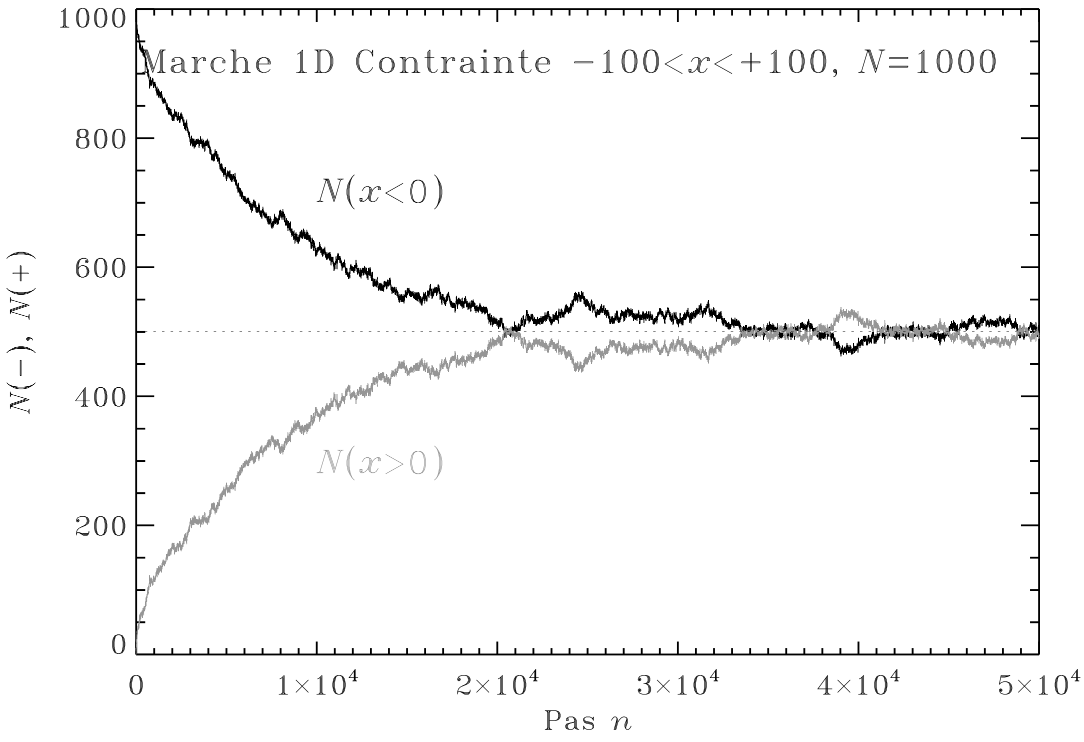


Figure 7.8: Variation temporelle du nombre de marcheurs situés dans la région $x < 0$ (trait noir) et $x > 0$ (trait gris). Pas besoin de l'oeil du lynx pour deviner ici une tendance exponentielle vers un état d'équilibre ou les marcheur sont distribués uniformément en fonction de x, et où leurs nombres de chaque coté de $x = 0$ deviennent égaux, à quelques fluctuations statistiques près (et qui décroitront comme la racine carrée du nombre de marcheur, comme vous l'aurez déjà deviné). Ne manquez pas de comparer ceci à la Fig. 5.12.

Considérons maintenant l'intervalle $x_0 - |s| < x < x_0 + |s|$ dans son ensemble, et dénotons par $N(x_0, t)$ le nombre de particules contenues dans cet intervalle au temps t. Le nombre au temps $t + \Delta t$ sera donné par le nombre au temps t plus ce qui est entré du coté gauche (à $x_0 - |s|$) moins ce qui est sorti du coté droit (à $x_0 + |s|$). Évaluant l'éq. (7.20) à $x_0 - |s|$ et $x_0 + |s|$, on arrive à:

$$\begin{aligned} N(x, t + \Delta t) &= N(x, t) + \delta N(x - s) - \delta N(x + s) \\ &= N(x, t) + \left(\left(\frac{1}{2} N(x - 2s) - \frac{1}{2} N(x) \right) - \left(\frac{1}{2} N(x) - \frac{1}{2} N(x + 2s) \right) \right) \\ &= N(x, t) + \frac{1}{2} \left(N(x + 2s) - 2N(x) + N(x - 2s) \right) \ , \end{aligned} \tag{7.21}$$

où on largué l'indice "0" sur le x et la valeur absolue sur s afin d'alléger la notation. Divisant les membres de gauche et de droite par Δt, on peut reécrire cette expression sous la forme équivalente:

$$\frac{N(x, t + \Delta t) - N(x, t)}{\Delta t} = \frac{1}{2} \left(\frac{(2s)^2}{\Delta t} \right) \times \left(\frac{N(x + 2s) - 2N(x) + N(x - 2s)}{(2s)^2} \right) . \tag{7.22}$$

La seconde parenthèse au membre de droite ressemble à quelque chose que vous avez déjà vu au chapitre 2, soit une formule de différence finie centrée pour la dérivée seconde par rapport à

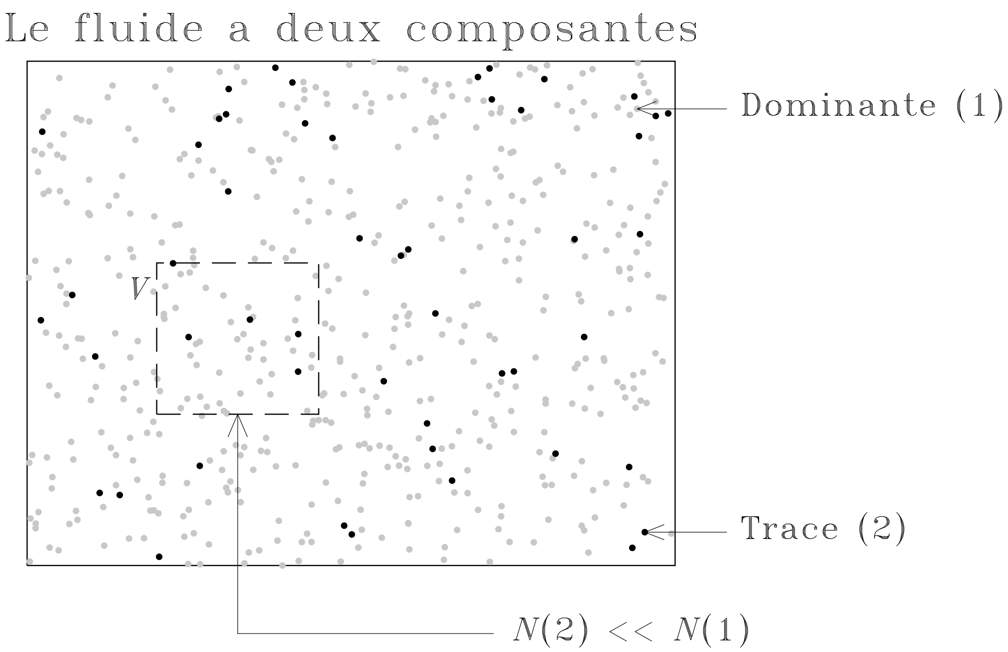


Figure 7.9: Représentation schématique d'un fluide à deux composantes, ici des molécules d'air (composante 1, en gris) et de Brut 45 Magnum (composante 2, en noir), vu ici à l'échelle microscopique. La composante (2) agit comme un contaminant trace en autant que son nombre de particules $N(2)$ est $\ll N(1)$; dans une telle situation, les paramètres thermodynamiques du gaz (densité, température, pression, etc.) sont complètement déterminés par la composante 1, et la diffusion de la composante 2 s'apparente à une marche aléatoire où les collisions avec la composante 1 réorientent aléatoirement la direction de déplacement des constituants microscopiques de la composante 2.

x pour un pas spatial $h \equiv 2s$ (voir l'éq. (2.19)), tandis que le membre de gauche ressemble tout autant à une formule de différence finie avant pour la dérivée temporelle, pour un pas temporel $h \equiv \Delta t$ (voir l'éq. (2.15)). Si l'on accepte ces interprétations, l'éq. (7.22) représenterait alors une discrétisation par différences finies de l'équation différentielle suivante:

$$\frac{\partial N(x,t)}{\partial t} = D\frac{\partial^2 N(x,t)}{\partial x^2} , \tag{7.23}$$

où on a défini le coefficient numérique

$$D = \frac{1}{2}\frac{(2s)^2}{\Delta t} , \tag{7.24}$$

qu'on appelle **coefficient de diffusion**. C'est une quantité qui mesure le taux auquel se disperse le contaminant trace dans le milieu ambiant; remarquez que D varie comme $s^2/\Delta t$, donc un gaz peu dense (grande distance interparticules) conduira à une diffusion plus rapide. De même, dans un gaz chaud les constituants microscopiques se déplacent plus rapidement, et donc (à densité égale) le temps inter-collisions est plus court, ce qui nous ferait prédire que D augmente avec la température. Ces deux attentes s'avèrent conformes à l'expérience. L'équation (7.23) est le prototype d'une **équation de diffusion**, qui s'avère décrire autant le transport conductif de la chaleur que la diffusion de polluants. Ceux et celles qui choisiront de suivre PHY-3140 auront l'occasion de traiter ces sujets en plus de profondeur.

Si l'on accepte que l'éq. (7.23) est une représentation continue valide de notre problème de marche 1D contrainte, alors l'on devrait pouvoir solutionner directement cette équation et retomber sur les mêmes résultats. Vérifions donc ça. L'idée générale est la même qu'au chapitre 3, c'est-à-dire qu'on introduit des formules de différences finies pour approximer les dérivées sur un maillage. On a cependant ici affaire à une **équation aux dérivées partielles**, où la variable dépendante $N(x,t)$ dépend de deux variables indépendantes, ici x et t. On a donc deux maillages distincts à définir, soir un pour la dérivée temporelle au membre de gauche de l'éq. (7.23), et un maillage spatial pour la dérivée seconde en x au membre de droite:

$$t_{m+1} = t_m + \Delta t \ , \qquad m = 0, 1, 2, \dots \ , \tag{7.25}$$

$$x_{k+1} = x_k + \Delta x \ , \qquad k = 0, 1, 2, \dots \ . \tag{7.26}$$

On associe alors à ces maillages la discrétisation

$$N_k^m \equiv N(x_k, t_m) \ , \tag{7.27}$$

où l'indice temporel m est placé en exposant pour bien le distinguer de l'indice spatial. Il ne reste plus qu'à introduire une différence finie avant pour la dérivée temporelle évaluées à x_k, et une différence centrée pour la dérivée spatiale évaluée au temps t_m:

$$\frac{N_k^{m+1} - N_k^m}{\Delta t} = D \frac{N_{k+1}^m - 2N_k^m + N_{k-1}^m}{(\Delta x)^2} \ , \tag{7.28}$$

ce qui conduit immédiatement à l'algorithme explicite:

$$N_k^{m+1} = N_k^m + D\Delta t \left(\frac{N_{k+1}^m - 2N_k^m + N_{k-1}^m}{(\Delta x)^2} \right) \ . \tag{7.29}$$

Cet algorithme est l'équivalent direct de la méthode d'Euler explicite considérée à la §3.2, dans le sens que le membre de droite est évalué à t_k, et est donc supposé "connu", soit du pas de temps précédent soit à partir de la condition initiale.

Considérons maintenant les conditions initiales et limites suivantes:

$$N(x, t = 0) = \begin{cases} 1 & x \leq 0 \\ 0 & x > 0 \end{cases} \tag{7.30}$$

$$\left.\frac{\partial N}{\partial x}\right|_{x=-100} = \left.\frac{\partial N}{\partial x}\right|_{x=+100} = 0 \ , \tag{7.31}$$

qui devraient encore éveiller en vous un certain sentiment de déjà vu... La Figure 7.10 montre une solution numérique à ce problème, avec $\Delta x = 1$, $\Delta t = 0.25$, et $D = 2$. Comparez maintenant ceci à la Figure 7.7; La solution a été tracée aux même temps équivalents, et avec le même code de couleur pour les différents traits. L'accord est excellent! Il est clair que l'éq. (7.23) est une excellente représentation de la marche aléatoire en 1D, dans la limite d'un nombre de marcheurs tendant vers l'infini.

Il faut remarquer comment le coefficient de diffusion a été calculé ici; dans l'éq. (7.24), le pas de temps Δt correspond à un pas de la marche, qui définit en quelque sorte notre unité de temps; et la quantité s mesure effectivement notre unité de longueur. Donc, dans ces unités, le coefficient de diffusion doit valoir $D = 2$. C'est donc la valeur utilisée pour le calcul de la solution de la Figure 7.10. Rendu au stade de la discrétisation numérique de l'éq. (7.23), on doit choisir un pas spatial et temporel pour la discrétisation, qui n'ont aucune raison particulière d'être identiques aux "unités naturelles" de la marche aléatoire. En fait, du point de vue strictement numérique, dans le cas d'une équation aux dérivées partielles de type diffusion, comme l'éq. (7.23), on peut montrer que le pas de discrétisation temporel doit satisfaire au critère:

$$\Delta t \leq (\Delta t)_C \equiv \frac{(\Delta x)^2}{D} \ , \tag{7.32}$$

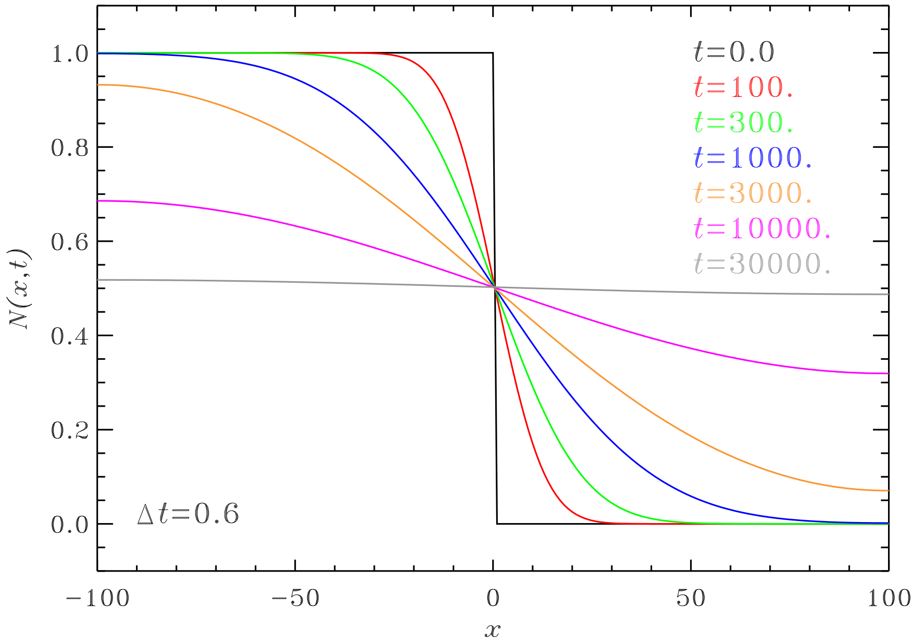


Figure 7.10: Solution numérique de l'équation de diffusion (7.23) avec les conditions initiale et limites données par les équations (7.30)—(7.31), avec $\Delta x = 1$ et $\Delta t = 0.25$.

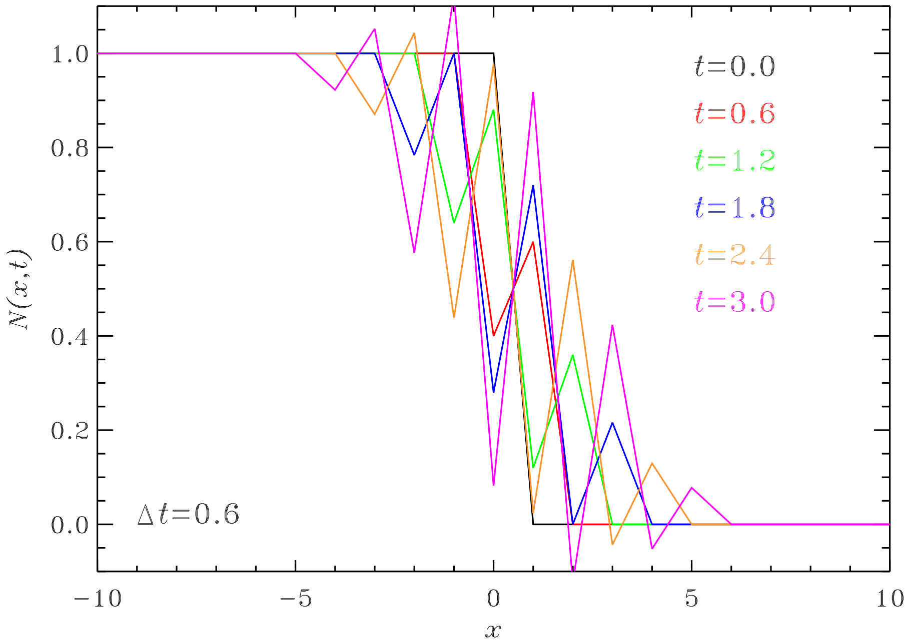


Figure 7.11: Une solution au même problème de diffusion que ci-dessus, mais cette fois pour seulement les cinq premiers pas de temps et en zoom sur la région $[-10 \leq x \leq 10]$ chevauchant la discontinuité dans le profil initial de $N(x,t)$. Cette solution est en tout point identique à celle ci-dessus, sauf pour l'utilisation d'un pas de temps $\Delta t = 0.6$.

afin de garantir la stabilité numérique de l'algorithme d'Euler explicite, pour un pas de discrétisation spatiale Δx. La dérivation de ce critère (appelé **critère de Courant**) est un superbe exercice en analyse numérique, mais qui dépasse les bornes de ce cours, donc prenons simplement pour acquis qu'il doit être satisfait. Ici, avec $\Delta x = 1$ et $D = 2$ on a $(\Delta t)_C = 0.5$, donc la solution de la Figure 7.10, avec $\Delta t = 0.25$ satisfait le critère de Courant par un facteur deux. Prendre un pas de discrétisation spatiale Δx plus grand rend le critère moins contraignant sur la taille du pas de temps, mais on risque alors de perdre au niveau de la précision à laquelle est discrétisée la dérivée seconde au membre de droite de l'éq. (7.23). Comme diraient nos collègues anglophones, "there is no such thing as a free lunch"...

Le critère de Courant est un truc à prendre au sérieux. La Figure 7.11 montre ce qui se passe quand on ne le fait pas. Il s'agit d'une simulation en tout point identique à celle de la Figure 7.10, sauf que cette fois le pas de temps $\Delta t = 0.6$ a été choisi légèrement plus grand que la grandeur limite $(\Delta t)_C = 0.5$ dictée par le critère de Courant. Les profils de $N(x,t)$ portés en graphique couvrent ici seulement les premiers cinq pas de temps. On y note une oscillation noeud-à-noeud se développant rapidement autour de la position de la discontinuité initiale à $x = 0$. Ces oscillations gagnent en amplitude avec le temps, et se propagent dans les directions gauche et droite du domaine, à une "vitesse" d'un point de maille spatiale par pas de temps. Donc, après seulement 100 pas de temps, la solution est complètement polluée par cette instabilité numérique. On pourrait croire que le problème est causé par la discontinuité dans la condition initiale utilisée ici, mais ce n'est pas le cas; un profil raide mais lisse (comme la fonction arctangente, par exemple) deviendra rapidement instable lui aussi. L'instabilité est directement associée au fait que le membre de droite de l'éq. (7.23) est évalué de manière complètement explicite, soit au pas de temps m (cf. l'algorithme (7.29)).

7.4 Marche aléatoire sur réseau

La **marche aléatoire sur réseau** est une variation sur le thème de la marche aléatoire qui permet d'incorporer une forme simple d'interaction entre marcheurs. L'idée de base est illustrée à la Figure 7.12, pour une marche sur réseau bidimensionnel de type cartésien débutant à $(0,0)$. Le marcheur ne peut qu'occuper les sites discrets (x_i, y_j) (aux intersection des traits pointillés sur la Fig. 7.12). Plutôt que de se déplacer dans des directions arbitraires comme dans le cas de la marche aléatoire classique, le marcheur ne peut bouger qu'à un des sites se trouvant dans son voisinage immédiat, haut-bas-droite-gauche, et celui des 4 sites voisins qui sera la cible du déplacement est choisi aléatoirement. Ceci peut sembler une forme extrêmement contraignante de marche aléatoire, mais vous aurez l'occasion de coder et expérimenter avec ce type de marche aléatoire sur réseau 2D au Labo 9, et d'évaluer les similarités et différences avec la marche aléatoire classique en 2D discutée précédemment. En particulier, vous constaterez (si tout va bien) qu'après un grand nombre de pas, l'orientation et la grandeur du vecteur-déplacement $\mathbf{D}_n$ est distribuée de manière essentiellement identique à la marche aléatoire 2D classique.

Tout l'attrait de la marche sur réseau réside en des situations où plusieurs marcheurs se déplacent *simultanément* sur le réseau, tel qu'illustré sur la Figure 7.13. Typiquement, en tout temps, seule une petite fraction des sites est occupée par les marcheurs (points noirs). Ces derniers ne peuvent encore que bouger à un des sites se trouvant dans leur voisinage immédiat, haut-bas-droite-gauche (indiqués sur la Figure par des cercles pour quelques marcheurs), avec le site-cible choisi aléatoirement, *mais le pas n'est effectué que si le site-cible est inoccupé*, sinon le marcheur reste sur son site jusqu'à la prochaine itération temporelle. Quelques exemples de pas permis sont indiqués par un trait vert sur la Figure 7.13.

Le fait qu'un marcheur ne puisse bouger à un site voisin occupé représente une forme très simplifiée de "collision", puisque, statistiquement, deux marcheurs voisins tentant de se croiser se verront stoppés, et tendront même à être "réfléchis" dans des directions opposées. Bien que la restriction géométrique imposée aux déplacements puisse sembler nous éloigner de la "réalité", les pseudo-collisions qu'elle incorpore au processus de marche tend à nous en rapprocher. Lequel des deux types de marche aléatoire est à préférer dépendra souvent du type

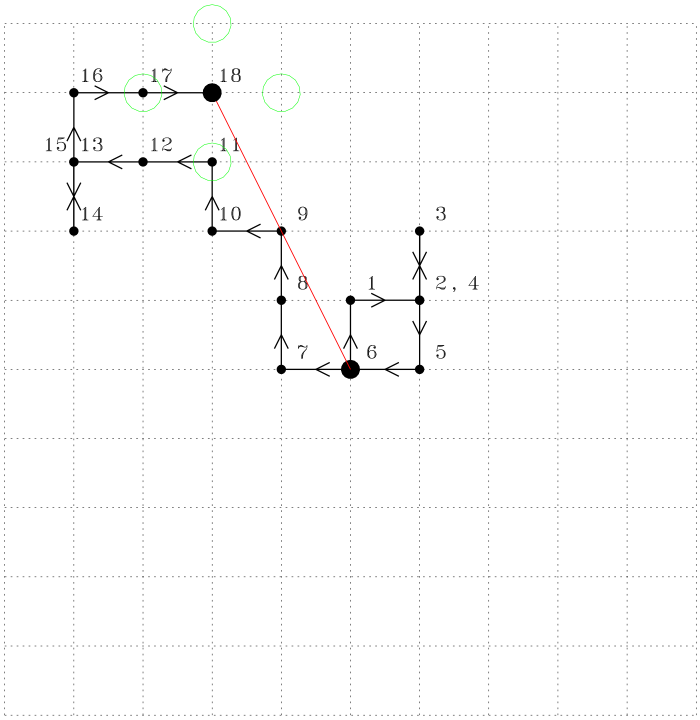


Figure 7.12: Les dix-huit premiers pas d'une marche aléatoire sur réseau en deux dimensions spatiales et débutant à $(x, y) = (0, 0)$. Le dix-neuvième pas atterrira sur l'un des 4 sites indiqués par un cercle, mais le site choisi l'est de manière complètement aléatoire et indépendante de la grandeur et orientation du vecteur-déplacement au dix-huitième pas (voir texte). Comparez cette Figure à la Figure 7.5 pour la marche aléatoire classique en 2D.

de problème physique que vous désirez examiner, comme l'illustre l'exemple discuté à la section qui suit.

7.5 Agrégation

La marche aléatoire sur réseau est particulièrement appropriée pour l'étude des phénomènes d'**agrégation**. On débute avec un grand nombre de particules distribuées de manière aléatoire mais statistiquement uniforme sur le réseau, et effectuant chacune leur petite marche aléatoire comme expliqué à la §7.4. Quelque part dans le réseau, on introduit une ou plusieurs particules fixes spatialement et "collantes", dans le sens que les particules venant à occuper un site voisin stoppent leur mouvement, et deviennent collantes à leur tour. Ceci conduira, au fil du temps, à une agrégation de particules se collant successivement aux particules-germes. Ça, on s'en serait bien douté; mais on n'aurait guère anticipé les *formes* des agrégats résultant de ce processus dit d'**agrégation limitée par la diffusion** ("diffusion-limited aggregation" en anglais, abbrévié DLA).

La Figure 7.14 montre un exemple spéficique, où l'agrégation débute à partir d'une seule particule-germe, située ici au centre d'un réseau cartésien de taille 1024×1024. L'agrégation implique ici 2×10^4 particules initialement distribuées aléatoirement sur le réseau. Les couleurs encodent l'ordre dans lequel les particules ont été capturées par le germe; examinez bien l'échelle de couleur, et constatez qu'à n'importe quel stade de la croissance de la structure, cette croissance s'opère selon une série de captures et branchements successifs se produisant presque

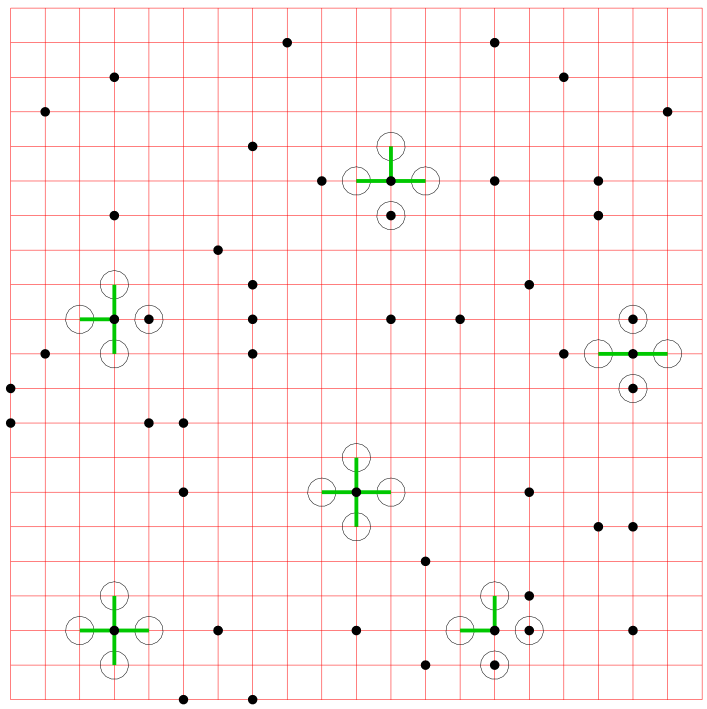


Figure 7.13: Géométrie et règles de déplacement pour la marche aléatoire sur réseau, ici un réseau cartésien régulier de dimensions 20×20 contenant 50 marcheurs distribués aléatoirement. Les déplacements aux sites voisins (cercles, indiqués ici pour un sous-ensemble de 6 particules) ne sont permis (traits verts) que si le site-cible est inoccupé.

toujours aux extrémités de branches déjà existantes.

La formation de structure de type dendritique s'explique par le fait que la moindre aspérité se formant par hasard sur l'agrégat en croissance tend à "capturer" plus de particules que les surfaces planes. Ceci est illustré sur la Figure 7.15. Sur un réseau (cartésien), il n'y qu'une manière de toucher une surface plane (un pas perpendiculaire à et dirigé vers la surface; voir Figure 7.15A), tandis qu'une aspérité peut être habituellement atteinte de plusieurs directions. (voir Figure 7.15B). De plus, une fois deux dendrites plus ou moins parallèles formées, l'espace entre les deux sera difficile à remplir, puisque les particules exécutant une marche aléatoire entre les dendrites auront une plus grande probabilité de se coller à une des dendrites qu'à atteindre leur point de branchement. Dans certains cas, comme sur la Figure 7.15B, les sites voisins du point de branchement peuvent même devenir carrément inaccessibles. Ces effets jouent tous dans le même sens: croissance et branchements successifs des structures dendritiques, plutôt que remplissage homogène du volume.

La structure dendritique résultant du processus d'agrégation est visuellement spectaculaire, et de plus, comme on le verra plus loin, possède des caractéristiques géométriques assez particulières, notamment une **invariance d'échelle**. D'un point de vue strictement visuel, sa forme peut rappeler l'arborescence des dendrites neuronales, les racines de plantes, ou les branches d'un bassin hydrographique, mais il n'y en fait aucun lien physique puisque ces structures naturelles ne se forment pas par agrégation. D'autres systèmes physiques, cependant, résultent de tels phénomènes. Les cristaux de givres se formant sur une surface plane croissent souvent à partir d'une petite irrégularité sur laquelle des molécules de H_2O diffusant sur la surface viennent s'agglutiner. De même, les grains de poussière interplanétaire et interstellaire se for-

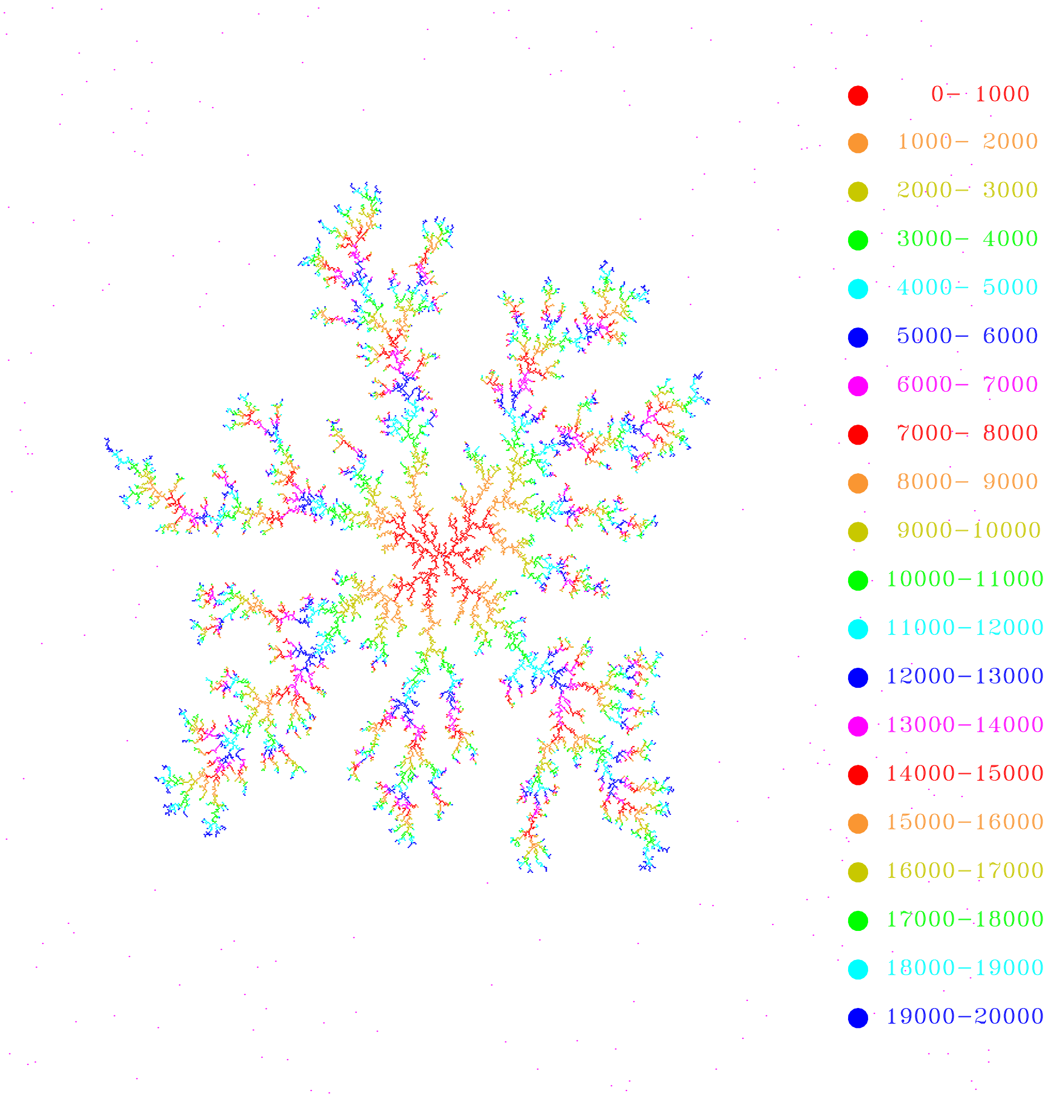


Figure 7.14: Formation d'une structure dentritique à partir d'un germe central, par agrégation limitée par la diffusion. Simulation basée sur la marche aléatoire de 20000 particules sur réseau 2D cartésien (§7.4), ici de dimensions 1024×1024. Les couleurs indiquent l'ordre dans lequel les particules se sont collées sur la structure: le rouge correspond aux premières 10^3 particules, l'orange aux 10^3 suivantes, et ainsi de suite selon le code de couleur détaillé à droite du diagramme.

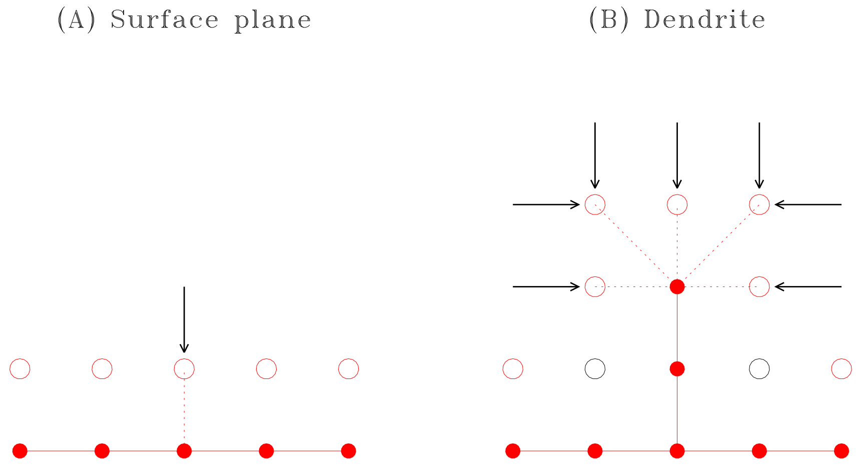


Figure 7.15: Agrégation (A) sur une “surface” plane, et (B) sur une aspérité. Les points rouges reliés par un trait indiquent les particules fixes, et les cercles rouges les sites où pourraient potentiellement se coller une particule mobile. Dans la situation en (A), chaque particule fixe ne contrôle qu'un site de capture (pointillé rouge), qui n'est lui-même accessible que du site situé au dessus (flèche noire). En (B) cependant, la particule située à l'extrémité de l'aspérité contrôle cinq sites de capture, qui collectivement peuvent être atteints par 7 pas distincts. Notons également que les deux sites indiqués par des cercles noirs sont devenus inaccessibles aux particules mobiles, qui se colleraient à la structure avant de pouvoir y parvenir.

ment par agrégation de plus petits grains entrant en collision; leur forme révèle d'ailleurs assez clairement cette origine (voir Fig. 7.16). Et dans un air saturé en vapeur d'eau et subissant un refroidissement, les cristaux de neiges se forment habituellement par agrégation des molécules de H_2O sur un petit grain de poussière... parfois d'origine interplanétaire! (voir Fig. 7.17). Ici la structure moléculaire de H_2O limite l'agrégation d'une molécule sur une autre à des angles très spécifiques, ce qui explique la remarquable régularité géométrique du flocon (comparé à l'agrégat beaucoup plus irrégulier de la Figure 7.14, par exemple), mais le principe est le même. En particulier, la moindre aspérité se formant le long d'une aiguille cristalline devient un site de capture plus probable, et donc tend à croitre.

7.6 L'invariance d'échelle

Revenons à notre agrégat de la Figure 7.14. On a vu (cf. Fig. 7.15) que sa croissance dendritique est favorisée par le fait que les aspérités ont une plus haute probabilité de capturer des particules libres, que les particules fixes formant les troncs des structures dendritiques. Le même raisonnement peut s'appliquer à un tronc dendritique dans son ensemble; plus un tronc contient de branches et sous-branches, plus il est efficace dans la capture de particules libres se déplaçant aléatoirement sur le réseau. Le taux de croissance (et de branchement) d'une dendrite augmente donc avec sa taille, même si à l'échelle locale du réseau, le processus de capture implique toujours une particule mobile pénétrant le voisinage immédiat d'une particule fixe. Cette propriété conduit à une invariance d'échelle dans l'agrégat. Ceci est illustré à la Figure 7.18, qui montre deux zooms successifs, chacun par un facteur 4, de l'agrégat. Examinez bien les deux zooms. Les patterns de branchements, soit la probabilité de branchement en se déplaçant le long d'un tronc, la distribution des longueurs de branches secondaires, bref, l'allure visuelle générale des dendrites, est la même aux trois échelles spatiales représentées ici. Cette invariance

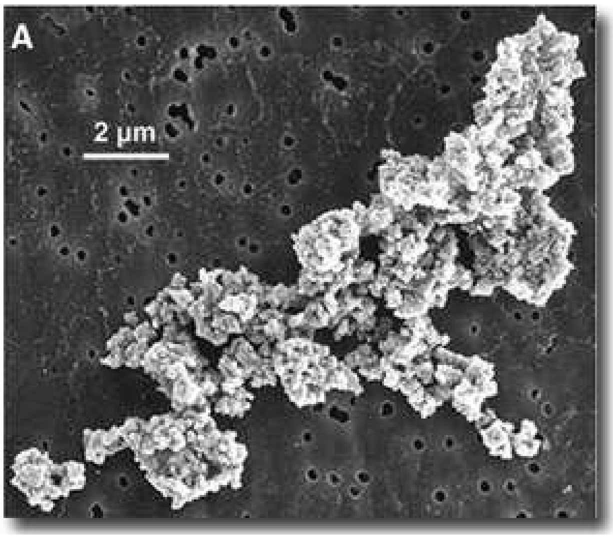


Figure 7.16: Photographies au microscope électronique d'un grain de poussière interplanétaire. Image extraite de `www.physorg.com/news3176.html`.

d'échelle, parfois aussi appelée autosimilarité, est une conséquence directe de l'absence d'échelle caractéristique dans le processus dynamique (ici l'agrégation) étant responsable de la formation de la structure.

7.7 Les fractales et leur mesure

Les formes géométriques présentant une invariance d'échelle sont souvent caractérisées comme étant des **fractales**. Nous allons clore ce chapitre en examinant de plus près —et en formalisant mathématiquement— ce concept.

Retournez lire la citation de Galilée servant d'ouverture au chapitre 1. Si vous êtes convaincu(e) par son savant discours essayez ensuite d'aller me construire une coquille d'escargot en blocs Lego, et repensez-y bien. Un moteur de voiture démonté peut se réduire à une collection de cylindres, plaques d'épaisseur diverses, hexagones troués et autres "figure géométrique" simples ou combinés. Il en va ainsi de la plupart des constructions technologiques. Mais, comme l'a si bien exprimé Benoit Mandelbrot,

> "Les montagnes ne sont pas des cônes, les nuages ne sont pas des sphères, les arbres ne sont pas des cylindres, et la foudre ne voyage pas en ligne droite".

Accrochez vos tuques avec de la broche, vous vous apprêtez à franchir la ligne séparant l'idéalisation physique de la réalité. La nature regorge de structures ayant des formes qui s'avèrent difficile à catégoriser à l'aide de la géométrie Euclidienne si chère à Galilée (et bien d'autres avant et après lui). Comment décrire mathématiquement de telles structures? Notre réponse à cette question débutera par un exemple géométriquement simple (à prime abord) mais mathématiquement plus que surprenant.

Considérons la construction géométrique itérative illustrée à la Figure 7.19. On débute avec un **germe** ayant la forme d'un segment de droite de longueur unitaire ($n = 0$ sur la Figure 7.19). On subdivise maintenant ce segment en trois sections de longueur égales, et l'on élève un triangle équilatétal dans la section centrale; la courbe $n = 1$ montre le résultat de cette opération. Puis, on répète ce processus avec chacun des quatres segments composant la figure $n = 1$, ce qui conduit à la figure $n = 2$; et ainsi de suite pour $n = 3$, $n = 4$, etc, comme sur

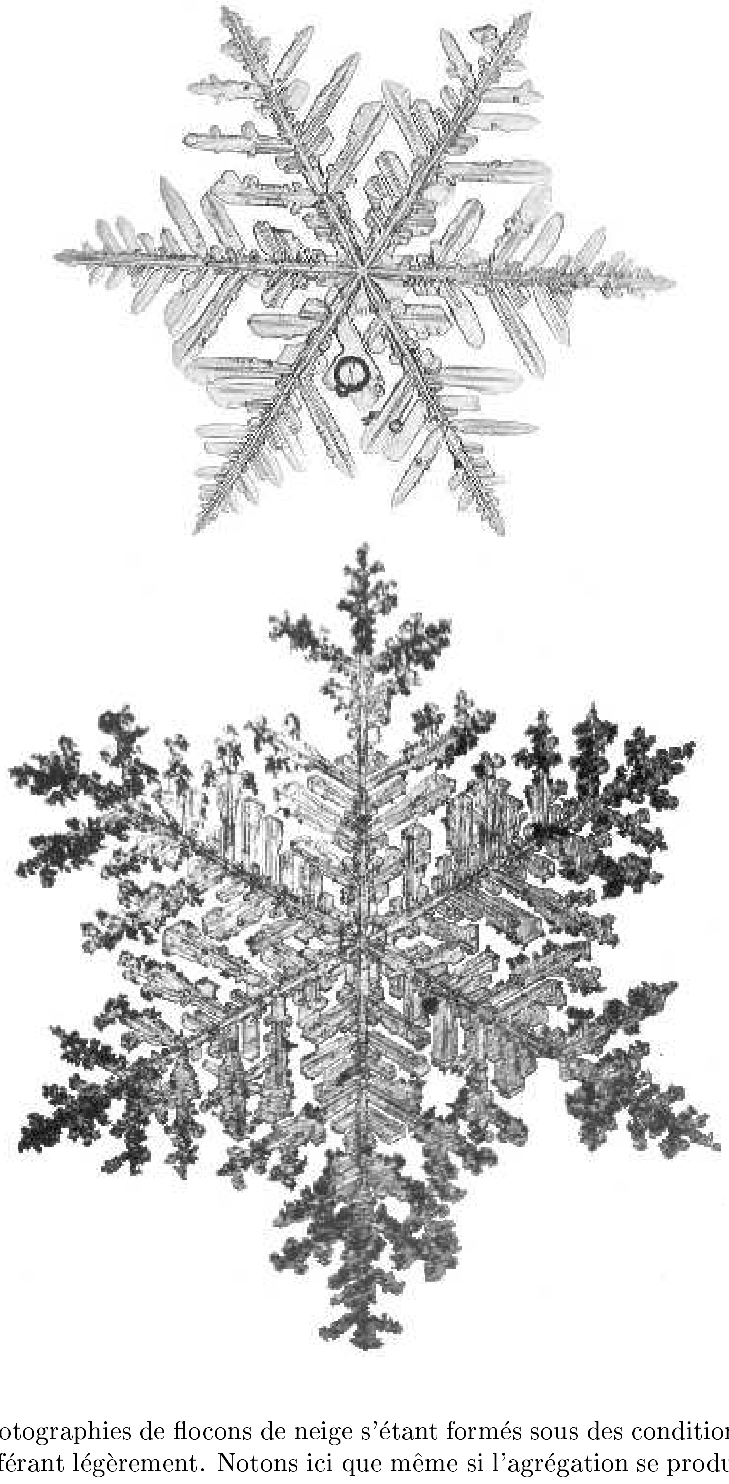


Figure 7.17: Photographies de flocons de neige s'étant formés sous des conditions d'humidité et température différant légèrement. Notons ici que même si l'agrégation se produit dans l'espace à trois dimensions spatiales, l'agrégat produit est une structure bi-dimensionnelle. Ces images sont tirées de la magnifique collection de photographies de flocons de neige qui se trouve sur le site Web `www1.odn.ne.jp/snow-crystals/English_index.html`.

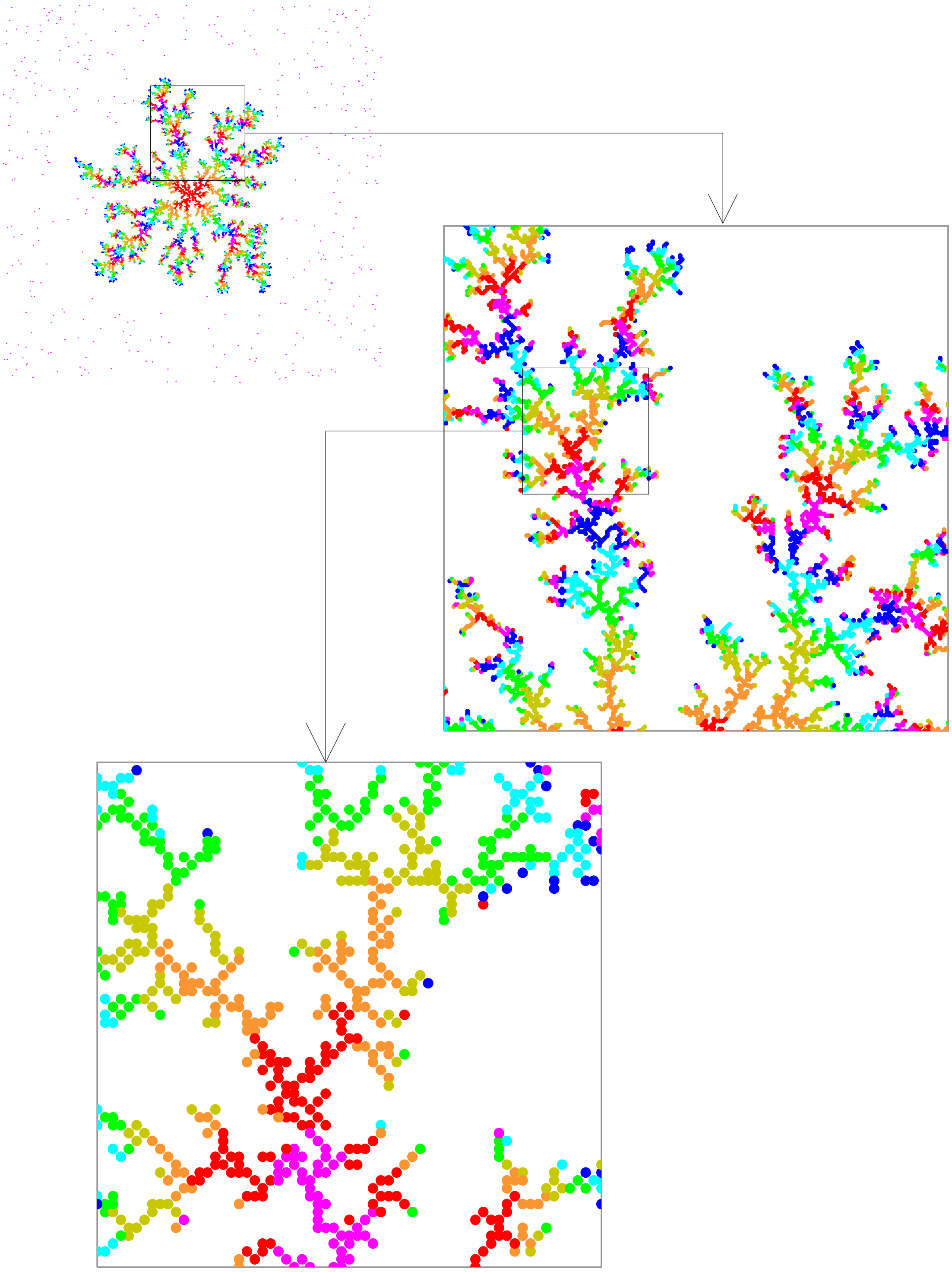


Figure 7.18: Invariance d'échelle dans l'agrégat de la Figure 7.14. Les deux zooms successifs représentent chacun un agrandissement par un facteur 4. Ce n'est ici qu'au second zoom que l'on peut commencer à distinguer que cette simulation est faite sur réseau, via le fait que les particules ne collent qu'horizontalement, verticalement, ou à 45 degrés les unes des autres.

la Fig. 7.19 (qui ne se rend que jusqu'à $n = 6$). La courbe résultant de ce processus dans la limite $n \to \infty$ s'appelle la **fractale de Koch**. Avec un germe ayant la forme d'un triangle équilatéral, on obtient quelque chose qui ressemble fort à un flocon de neige...

La fractale de Koch ne pourrait être qu'une jolie forme géométrique, mais cette courbe s'avère avoir des propriétés mathématiques intrigantes, pour ne pas dire déroutantes. Le processus de génération implique que le nombre de segments augmente d'un facteur 4 à chaque étape du processus. Par contre la longueur de chacun de ces segments diminue d'un facteur 3. Donc, la longueur totale $L(n)$ de la courbe à l'itération n sera donnée par une expression du genre:

$$L(n) = 4^n \times (1/3)^n = \left(\frac{4}{3}\right)^n \ . \tag{7.33}$$

Puisque $4/3 > 1$, on a clairement que la longueur de la courbe tend vers l'infini à mesure que n augmente:

$$\lim_{n\to\infty} L(n) \to \infty \ . \tag{7.34}$$

À examiner les longueurs des 6 premières itérations de la fractale de Koch sur la Figure 7.19, cette augmentation peut paraitre lente, mais je vous laisse vérifier qu'à partir d'un germe de $L = 3$ centimètre, déjà à $n = 100$ la fractale de Koch "dépliée" a une longueur suffisante pour faire le tour de la Terre 4000 fois environ.

Ce qui est remarquable ici est que du point de vue du plan de la feuille, la fractale de Koch, de longueur infinie, a un début et une fin bien définis, soit les deux extrémités du segment de droite définissant le germe. Comment une ligne de longueur infinie peut-elle avoir un début et une fin? et de surcroit s'emboiter sans problème à l'intérieur d'un périmètre de longueur finie, e.g., un rectangle. C'est déjà bizarre, mais il faut de plus noter que cette courbe de longueur infinie ne "remplit" que très peu le rectangle. Quid?

Helge von Koch (1870–1924) était un mathématicien diplomé et respecté, qui s'amusait dans ses temps libres à découvrir ce qu'il appelait lui-même ses "monstres mathématiques". La plupart de ses contemporains ont jugé ses découvertes quelque part entre "amusantes" et "intéressantes", mais disons qu'on en est resté bien loin du prix Nobel... C'est le météorologue Lewis Fry Richardson (1881-1953) qui a le premier montré, initialement par inadvertance et apparamment sans rien connaitre des monstres mathématiques de Koch, que certaines structures naturelles avaient des propriétés semblables. Richardson s'intéressait à la mesure de la longueur des côtes de sa mère-patrie l'Angleterre. Ayant judicieusement déduit qu'il n'avait qu'à mesurer la longueur de la courbe de niveau d'altitude zéro sur une carte topographique, il se lança dans ce processus pour arriver à un résultat en toute apparence raisonnable. Notoirement futé et pionnier de l'approche "think different", Richardson avait cependant très bien réalisé que son résultat sous-estimait la longueur réelle des côtes, puisque à l'échelle de sa carte de toute l'Angleterre plusieurs petites irrégularités, estuaires de petits cours d'eau, etc, n'étaient pas visibles. Il eu donc la géniale idée de reprendre le processus sur une section des côtes à l'aide d'une carte à plus petite échelle, afin de calculer un facteur de correction à appliquer à sa mesure. Ce facteur s'avérant substantiel, Richardson répéta le processus avec une carte d'encore plus petite échelle, pour trouver un facteur de correction encore plus grand, sans aucune indication de "convergence". Tout comme la fractale de Koch, les côtes de l'Angleterre représentent une ligne ici de toute évidence *fermée*, mais néanmoins de longueur infinie *et* pouvant s'emboiter dans un rectangle de taille finie (e.g., une carte topo de l'Europe).

Géométriquement parlant, la fractale de Koch et la courbe de niveau décrivant les côtes de l'Angleterre sont "plus" qu'une ligne, mais "moins" qu'une surface; autrement dit, ce sont des objets géométriques auxquels on devrait assigner une dimension entre un et deux, soit une dimension fractionnaire (d'où l'origine du terme "fractale"). Mais comment quantifier ceci? La solution se trouve dans la *mesure* de l'objet, et suit essentiellement la logique de Richardson.

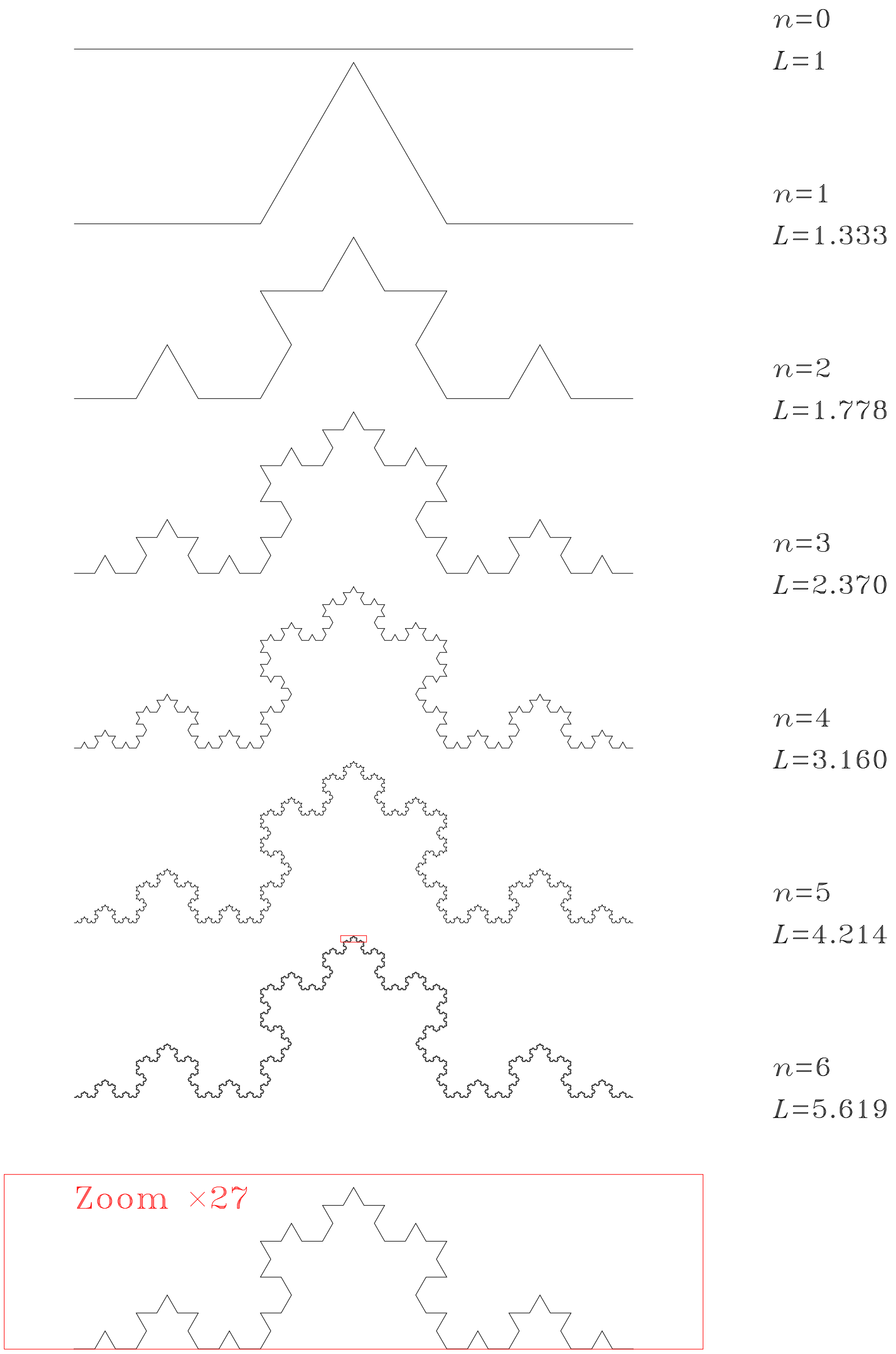


Figure 7.19: Les 6 premières étapes de la génération de la fractale de Koch. Le germe ($n = 0$) est un segment de droite $[0, 1]$, et à chaque étape le tier central est remplacé par un triangle équilatéral pointant vers le coté “extérieur” de la structure. L'image du bas est un zoom par un facteur 27 de la petite région délimitée par le tout petit cadre rouge, à peine visible, au sommet de la courbe de l'itération $n = 6$; notez comment ce zoom est identique à la structure à l'itération $n = 3$. La longueur L à chaque itération est également indiquée.

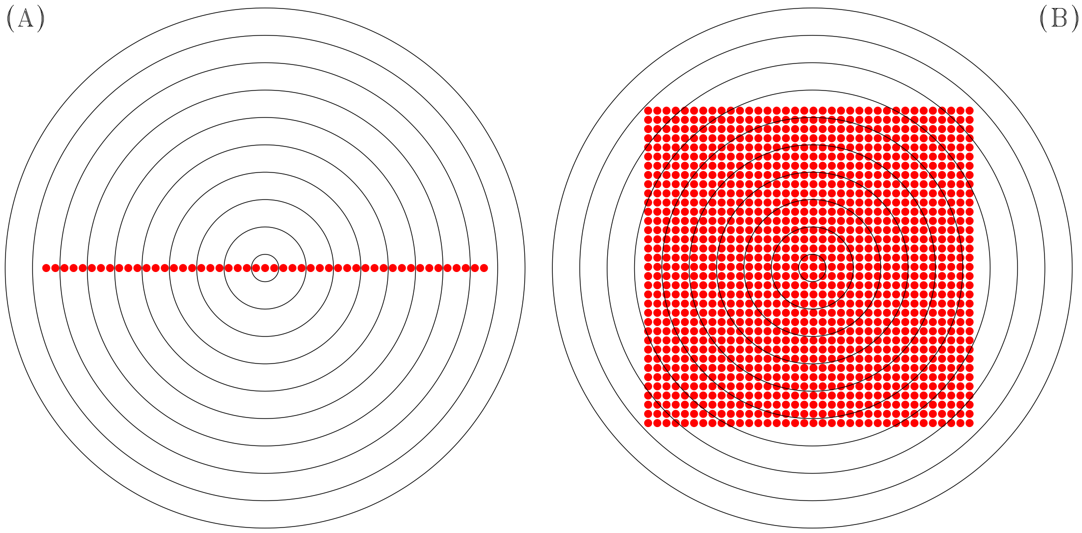


Figure 7.20: Mesure "masse–rayon" de deux objets géométriques simples, tous deux composés au niveau microscopique d'un agrégat de "particules". Le premier est une structure linéaire (i.e., à une dimension spatiale), tandis que le second est une structure à deux dimensions spatiales, ici un carré. La masse $M(R)$ est définie ici comme le nombre de particules contenues à l'intérieur d'un cercle de rayon R centré sur l'objet (voir texte).

7.8 La dimension fractale

L'idée sous-jacente à la mesure géométrique de la dimensionalité d'un objet s'illustre le plus facilement en commencant par un exemple simple, soit les deux "objets" illustrés à la Figure 7.20. En anticipation à ce qui suivra, ces objets sont considérés comme étant formés d'un certain nombre de "particules" ou composantes microscopiques (points rouges). Mais à une échelle spatiale beaucoup plus grande que la taille de ces particules et que la distance-interparticules, l'objet en (A) est une ligne, tandis que celui en (B) est un carré.

Pour identifier la position d'une particule composant l'objet en (A), on n'a besoin que d'une seule coordonnée, par exemple la distance depuis un des bouts de la ligne; et pour celui en (B), on aurait besoin de deux coordonnées, par exemple mesurées dans les directions horizontale et verticale à partir du coin inférieur gauche du carré. On dit donc que la ligne est un objet de dimension un, tandis que le carré est un objet de dimension deux. Mais cette définition "paramétrique" de la dimensionalité est incomplète à bien des points de vues; par exemple, chaque particule composant l'agrégat de la Figure Fig. 7.14 peut être localisée à l'aide de deux coordonnées (cartésiennes ou autres), donc on dirait aussi de cet objet qu'il est de dimension deux. Du point de vue géométrique cependant, cet agrégat est fondamentalement différent d'un carré, et cette différence se doit d'être quantifiée.

Considérons la procédure suivante: à partir du centre géométrique de chacun des deux objets, on trace une série de cercles de rayons R croissants (voir Figure 7.20). Pour chacun de ces cercles, on calcule la "masse" $M(R)$, soit le nombre de particules contenues à l'intérieur de chaque cercle. Il est clair ici que dans le cas de la ligne, ce nombre croitra linéairement avec R, tandis que dans le cas du carré on aura $M \propto R^2$, et ce tant que le diamètre du cercle demeure plus petit que la taille de chaque objet. C'est en effet ce qu'on mesure, comme le montre la Figure 7.21. Ce graphique est tracé avec des axes logarithmiques, donc une droite de pente D indique une relation mathématique entre M et R dite en Loi de puissance, soit:

$$M \propto R^D \ , \qquad D \geq 0 \ . \tag{7.35}$$

Dans le cas de la ligne, on a $D = 1$, mais $D = 2$ pour le carré. On voit bien ici que pour des

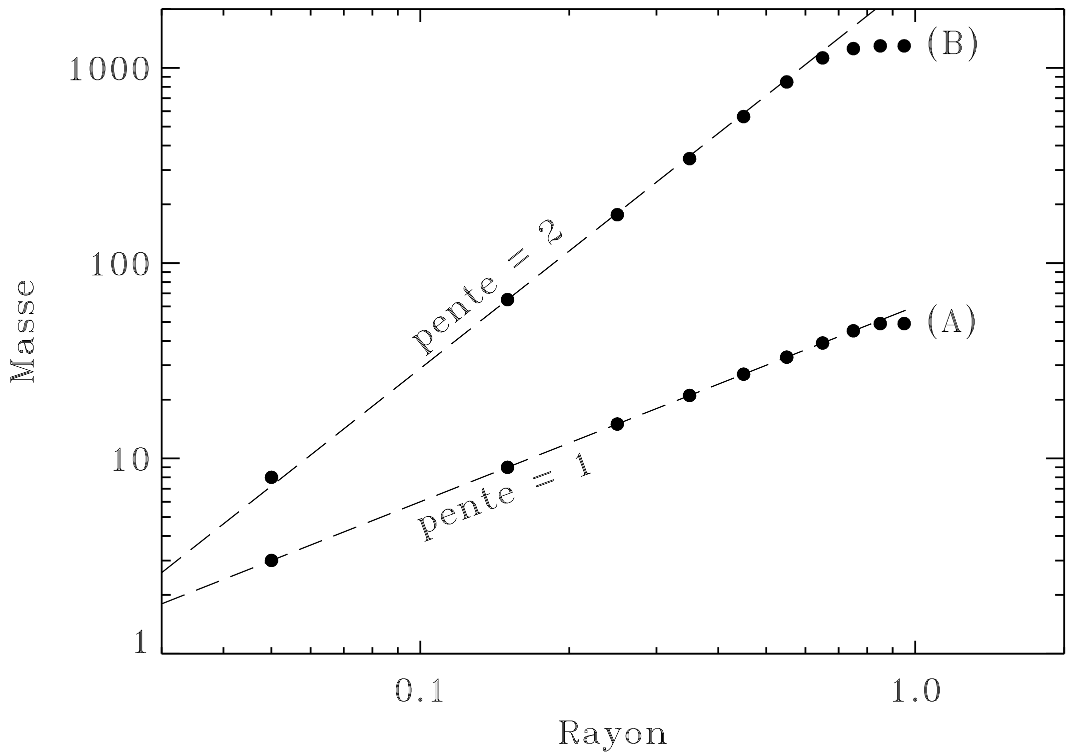


Figure 7.21: Relation masse–rayon pour les deux objets géométriques simples de la Figure 7.20. Une pente logarithmique de +1 correspond ici à une dimension $D = 1$ (objet à une dimension spatiale), tandis qu'une pente de +2 indique $D = 2$ (objet bi-dimensionnel), comme on s'y serait attendu à voir la Figure 7.20. Dans les deux cas les mesures (points noirs) "décrochent" de la droite quand R devient comparable à la grandeur de l'objet.

objets géométriquement simples, D correspond à la dimensionalité habituelle de l'objet.

Il est facile d'appliquer la même procédure de calcul à notre agrégat DLA de la Figure 7.14. La Figure 7.22 en montre le résultat. La masse M varie toujours avec R selon une loi de puissance (éq. (7.35)). Comme auparavant cette loi de puissance ne tient que pour des échelle spatiales plus petites que la taille de l'objet, mais passablement plus grande que la distance inter-particules. Mais cette fois-ci la pente logarithmique est $D = 1.66$; bien que l'agrégat soit défini dans un plan 2D, sa structure a une dimension entre un et deux. L'agrégat est donc "plus" qu'une ligne, mais "moins" qu'une surface. C'est une structure fractale. On doit remarquer que pour obtenir une valeur fiable de D, la structure doit être suffisamment grande pour couvrir au moins un ordre de grandeur en rayon, de manière à ce que la pente logarithmique puisse être déterminée avec une précision suffisante.

La relation masse-rayon devient difficile à appliquer à des objets de forme complexe, et la valeur de D qui en résulte se retrouve à mesurer à la fois la forme globale de l'objet, ainsi que sa structure interne. Une méthode plus robuste de la dimension fractale est basé sur le concept de la *mesure*. Revenons à notre familier agrégat. Étant un objet défini dans un plan 2D, tout comme le désormais célèbre étang du chapitre précédent, la mesure de sa "surface" peut se faire en le recouvrant de petit carrés contigus de surface $M \times M$ connue, et en comptant simplement le nombre N de carrés requis. Ceci s'appelle la méthode du décompte des boites (traduction libre de "box-counting method"). Cette méthode est particulièrement facile à utiliser pour des structures définies sur des réseaux, ou des images pixellisées.

La procédure est illustrée à la Figure 7.23, pour des carrés de coté égal à M =8, 16,

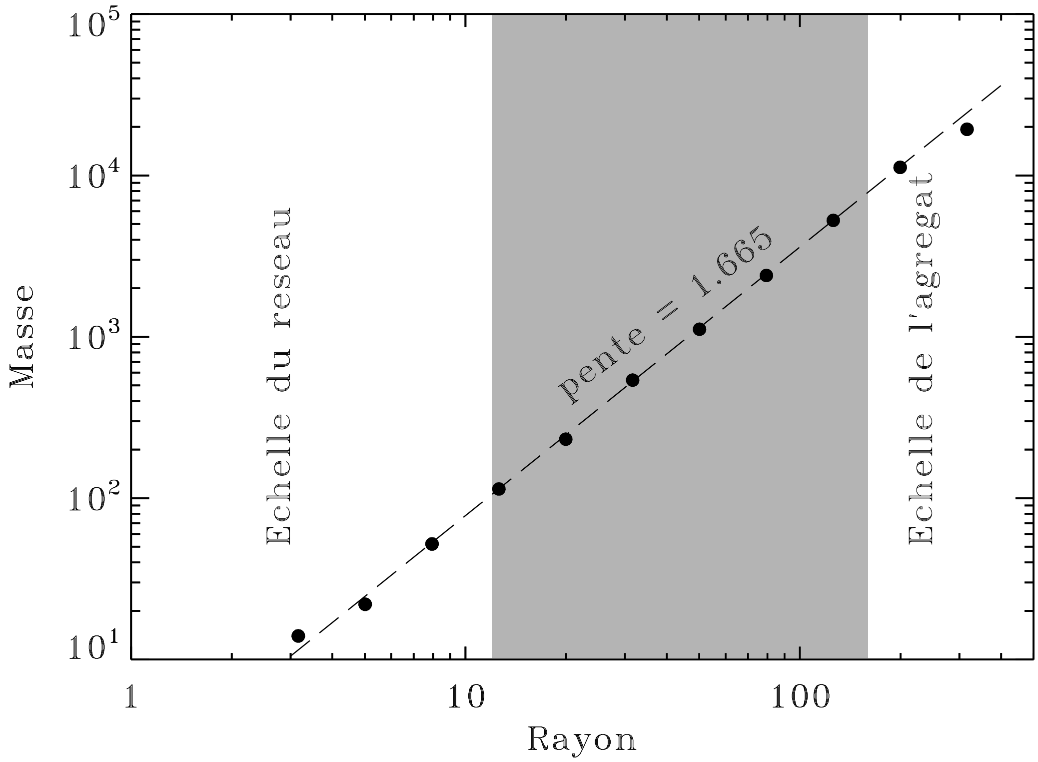


Figure 7.22: Relation masse–rayon pour l'agrégat de la Fig. 7.14. La pente logarithmique vaut maintenant 1.66, ce qui, géométriquement parlant, place l'objet quelque part entre la "ligne" et le "carré", en d'autres mots, entre le unidimensionnel et le bidimensionnel. La région en gris indique l'intervalle utilisé pour calculer la pente (voir texte).

32 et 64. L'unité est ici la distance entre deux noeuds voisins sur le réseau où se fait la marche aléatoire. Remarquez bien qu'il n'importe pas ici qu'une boite ne recouvre qu'une seule particule ou plusieurs, dès qu'elle en couvre une elle contribue "1" au décompte des boites. Encore une fois ici, l'utilisation de boites de taille comparable à (ou plus petite que) la distance inter-particule n'a aucun sens physique et/ou géométrique. De plus, comme l'agrégat a ici un diamètre d'environ 500 unités, tout décompte utiliant des boites plus grande que cette taille produira $N = 1$ indépendamment de la taille de la boite.

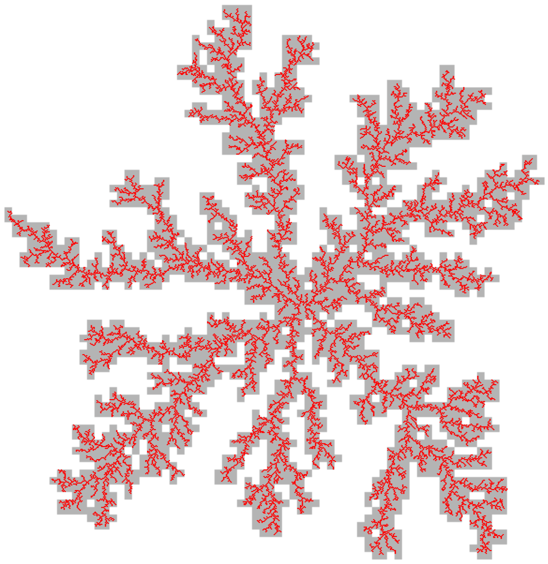
Le résultat de cette procédure de décompte est illustré à la Figure 7.24. On y a porté en graphique le décompte des boites en fonction de la résolution r $(= 1/M)$, définie ici comme l'inverse de l'échelle de mesure M (i.e., haute résolution $\equiv$ petite échelle de mesure), encore une fois selon des axes logarithmiques. On y note de nouveau une loi de puissance de la forme

$$N(r) \propto r^D \ , \qquad D \geq 0 \ . \tag{7.36}$$

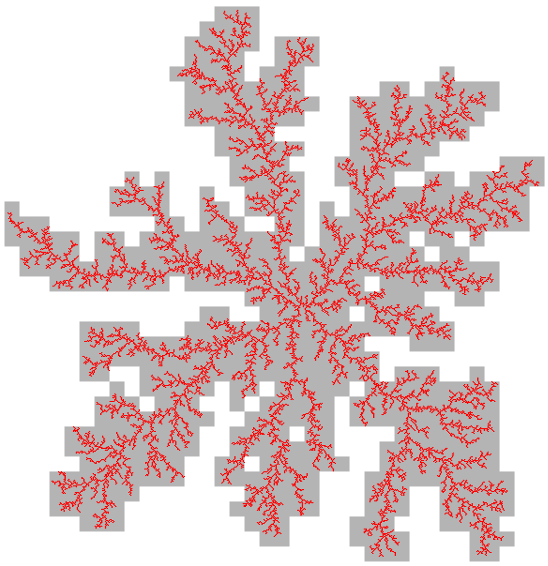
couvrant le domaine des échelles spatiales entre la taille de la structure (r petit) et la structure "microscopique" de l'agrégat (r très élevé). Comme auparavant la pente logarithmique donne directement l'exposant D, qui correspond encore une fois à la dimension fractale, ici $D = 1.59$. La dimension fractale ainsi calculée s'appelle la **dimension de Hausdorff**.

Le fait que les deux valeurs de la dimension fractale de notre agrégat, telles que calculées par la relation masse-rayon et la méthode du décompte des boites, ne soient pas en parfait accord n'est pas trop inquiétant en soi. Quelle que soit la méthode de calcul utilisée pour obtenir une valeur fiable de la dimension fractale de notre agrégat DLA, il faudrait en fait l'appliquer à un grand nombre d'agrégats distincts (i.e., produit selon des réalisations distinctes des marches

$M = 8$, Boxcount = 1940



$M = 16$, Boxcount = 661



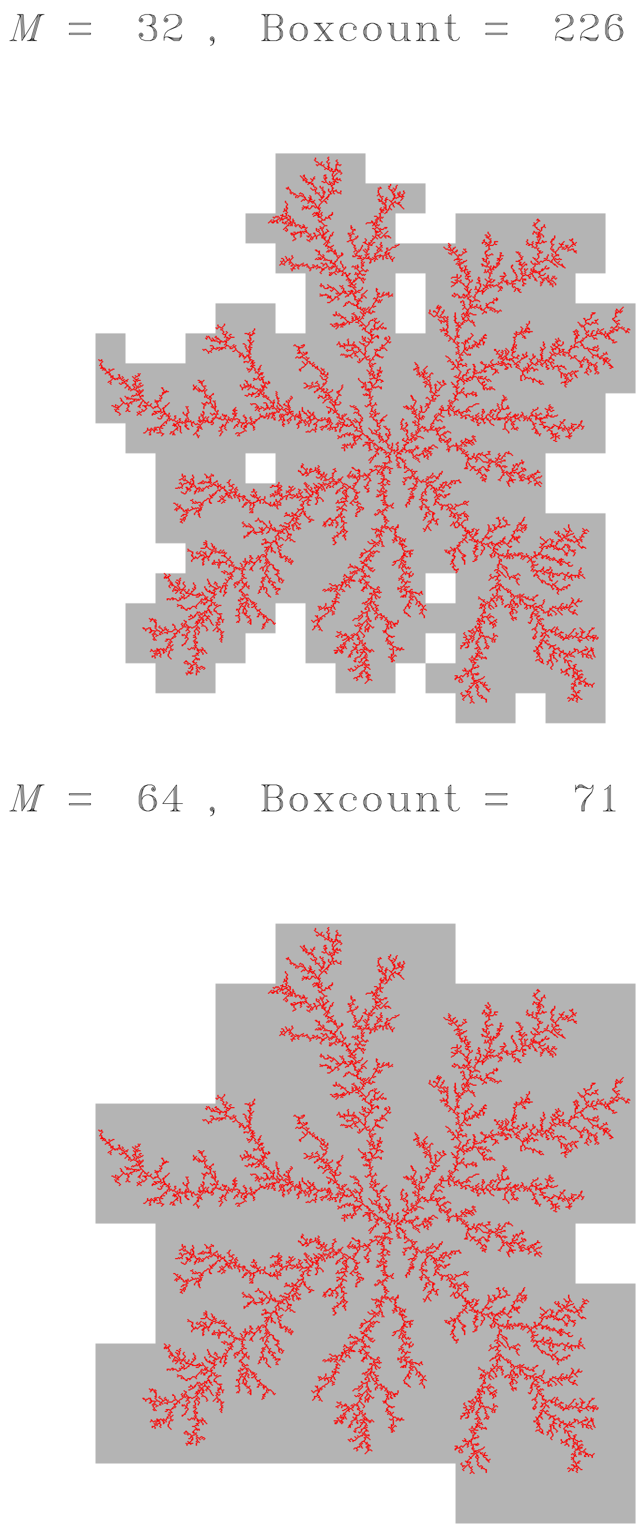


Figure 7.23: [Commence à la page précédente] Quatre itérations de la méthode de décompte des boites, appliquée à l'agrégat de la Figure 7.14. Chaque itération double la dimension linéaire des carrés utilisés pour couvrir l'objet, par rapport à l'itération précédente.

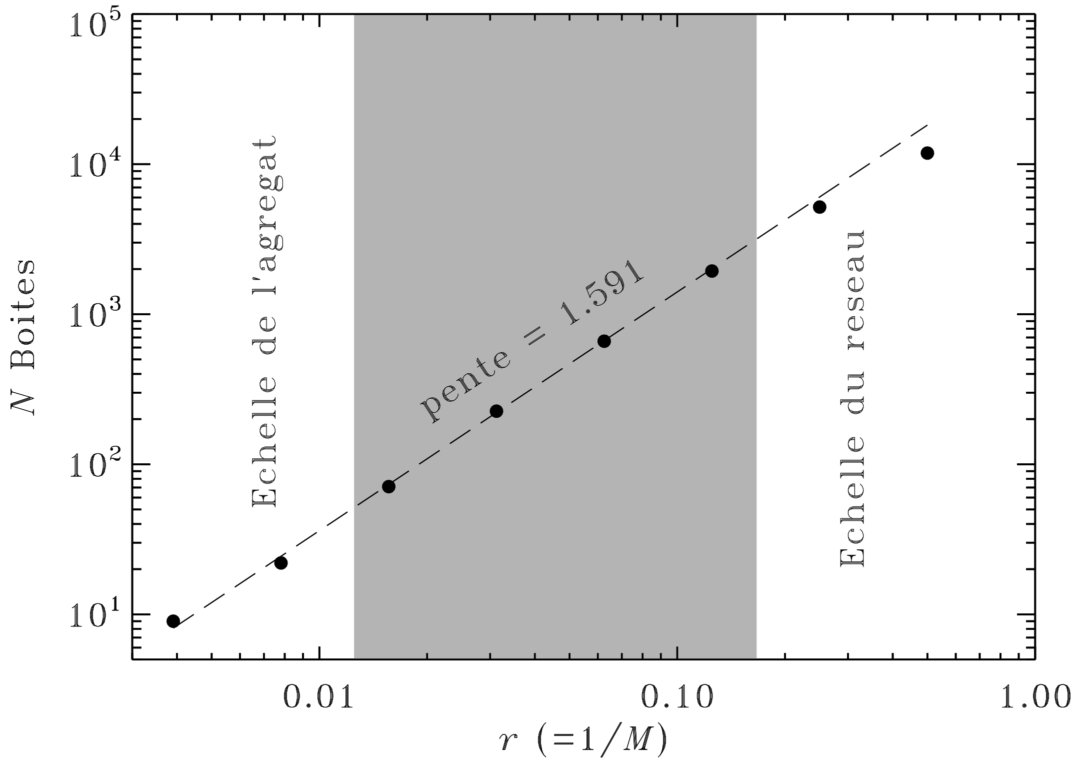


Figure 7.24: Détermination de la dimension fractale de l'agrégat DLA de la Figure 7.14 à l'aide de la méthode du décompte des boites. Comme auparavant, la dimension fractale correspond à la pente logarithmique de N versus r, et est déterminée sur un intervalle de résolution ($r = 1/M$) encadré par les valeurs correspondant à la taille de la structure (r petit) et à la distance interparticule (r grand).

aléatoires), et moyenner les valeurs de D en résultant. Dans le cas d'agrégat de type DLA, on trouve bel et bien $D \simeq 1.6$ indépendamment de la méthode utilisée

Qu'en est-il alors avec la fractale de Koch (Figure 7.19)? Ici l'objet géométrique utilisé pour mesurer la fractale est un segment de droite, plutôt qu'un carré. On a vu qu'à l'itération n, l'échelle M (longueur de chaque segment) est donnée par $(1/3)^n$, donc la résolution $r = 1/M = 3^n$. Le nombre N de segment requis pour couvrir la fractale, quand à lui, est donné par 4^n. On a donc

$$D = \frac{\log N}{\log r} = \frac{\log 4^n}{\log 3^n} = \frac{n \log 4}{n \log 3} = \frac{\log 4}{\log 3} = 1.26186 \ . \tag{7.37}$$

On notera en terminant que la fractale de Koch a une dimension substantiellement inférieure à celle de l'agrégat DLA, ce qui est intuitivement satisfaisant puisque le second semble bel et bien plus "bi-dimensionnel" que le premier.

Exercices:

1. Démontrez que le résultat $\langle D_n^2 \rangle = ns^2$ tient également pour une marche aléatoire en trois dimensions spatiales.

2. Modifiez le code C de la Figure 7.1 pour la marche aléatoire 1D afin d'introduire un biais systématique dans la marche, dans le sens que le pas dans la direction positive de l'axe

devient légèrement plus probable que le pas dans la direction négative (i.e., introduisez des probabilité $p(+)$ et $p(-)$ telles que $p(+) > p(-)$ mais $p(+) + p(-) = 1!$). Ensuite,

(a) Répétez l'expérience numérique décrite à la §7.2, et calculez l'équivalent des Figures 7.2 et 7.3 pour des valeurs de $p(+) = 0.525$, 0.55 et 0.6.

(b) À partir de vos résultats, estimez une "vitesse de dérive" vers la droite.

(c) Revenez maintenant à une simulation classique où les deux directions de pas sont équiprobables, mais ajoutez une composante de vitesse non-aléatoire correspondant à la vitesse de dérive estimée en (b); recalculez l'équivalent des Figures 7.2 et 7.3, et comparez à celles obtenues en (a).

3. Utilisant la même procédure de discrétisation qu'à la §7.3, solutionnez numériquement l'équation de diffusion (7.23) pour une condition initiale donnée par un profil gaussien:

$$N(x, t = 0) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{-x^2}{2\sigma^2}\right) ,$$

avec $\sigma = 0.5$, sur l'intervalle spatial $x \in [-10, 10]$. Ensuite,

(a) Vérifiez que la forme gaussienne du profil initial est préservée au cours de l'évolution;

(b) Examinez comment la déviation quadratique moyenne σ varie avec le temps.

4. Notre procédure de solution numérique de l'équation (7.23) évalue la dérivée seconde au membre de droite au temps t^m, tandis que la dérivée temporelle utilise une formule d'ordre $O(h)$ impliquant t^m et t^{m+1}; opérationnellement, ceci est donc équivalent à la méthode d'Euler explicite (§3.2). On avait vu au chapitre 3 que la précision de la solution pouvait être améliorée en évaluant le membre de droite comme une moyenne entre sa valeur à t^m est une extrapolation linéaire au temps t^{m+1}; c'était la méthode de Heun.

(a) Appliquez cette technique à la solution numérique de (7.23) pour le problème de mélange considéré à la §7.3;

(b) Calculez des solutions numériques pour diverses valeurs du pas de temps Δt, et déterminez si l'utilisation de la méthode de Heun change quelque chose à la stabilité de votre approche numérique.

Bibliographie:

La marche aléatoire est traitée dans tous les ouvrages traitant de processus stochastiques. Le chapitre 12 de l'ouvrage Gould et Tobochnik cité en bibliographie du chapitre 1 offre une bonne discussion du sujet.

De nos jours la diffusion est le plus souvent traitée dans certains ouvrages avancés de génie, dans le contexte de la diffusion de la chaleur ou des effets dynamniques de la viscosité dans les fluides (deux processus décrits par une équation effectivement identique à (7.23)). Dans la littérature plus physique, j'aime bien la discussion au chapitre 2 de:

Guyon, É., Hulin, J.-P., et Petit, L., *Hydrodynamique Physique*, CNRS éditions (2001).

Pour une approche encore plus physique (mais à un niveau passablement avancé), je recommenderais:

Landau, L., et Lifschitz, E., *Mécanique des Fluides*, Éditions MIR (2^e édition 1986).

La dérivation du critère de Courant se retrouve dans la plupart des ouvrages d'analyse numérique traitant de la solution d'équations aux dérivées partielles. La présentation au chapitre 20 du *Numerical Recipes* est tout à fait correcte et accessible.

Il existe une multitude d'ouvrages traitant de géométrie fractale. Le bouquin suivant, un des premiers sur le sujet, vaut encore bien la peine d'être lu:

Mandelbrot, B., *The fractal geometry of Nature*, Freeman (1982).

J'ai appris beaucoup des deux ouvrages suivants, que je me permets donc de citer ici même si je doute qu'ils soient les références optimales sur le sujet:

Prusinkiewicz, P., & Lindenmayer, A., *The algorithmic beauty of plants*, Springer (1990),
Flake, G.W., *The computational beauty of Nature*, MIT Press (1998).

Chapitre 8

Criticalité

En physique statistique, un système est dit **critique** si une petite perturbation localisée peut se retrouver à affecter le système dans son ensemble; l'illustration classique est la **transition de phase**, par exemple quand un tout petit apport de chaleur *quelquepart* dans un chaudron rempli d'eau à 99.9999... degrés Celcius déclenche l'ébullition de *tout* le chaudron. Ceci ne fonctionne cependant que si l'eau est à une température infinitésimalement près de 100°. Si la température est nettement sous 100°, l'apport de chaleur ne fera que réchauffer l'eau un peu plus; et si T dépasse 100° même seulement un tout petit peu, l'eau bouille déjà et l'apport d'une petite quantité de chaleur supplémentaire ne fera qu'accélérer légèrement l'ébullition. Dans le jargon de la physique statistique, on dirait que la longueur de corrélation (i.e., distance d'influence) d'une perturbation (ici l'ajout d'un tout petit peu d'énergie calorique) devient égale à la dimension du système physique (le chaudron). C'est donc à 100 degrés que se passe toute l'action, et le contrôle de la transition de phase demande un contrôle (habituellement externe) très fin de la température.

Ce concept de **criticalité** est d'une applicabilité beaucoup plus vaste que les transitions de phase en thermodynamique, et la physique en offre des réalisations particulièrement spectaculaires. Dans ce chapitre, nous étudierons un processus physique en apparence très simple, la percolation, qui offre un excellent exemple de criticalité, et qui de surcroit est particulièrement facile à simuler sur ordinateur. Nous définirons tout d'abord la percolation à l'aide d'un exemple très intuitif et d'une importance capitale dans l'enseignement universitaire (§8.1). Nous passerons ensuite à la percolation telle que simulée sur ordinateur, d'abord dans le cas spécial d'une dimension spatiale (§8.2), puis plus sérieusement en deux dimensions spatiales (§8.3). Ceci nous conduira à une étude des propriétés géométriques des structures produites par percolation, et nous nous attarderons plus spécifiquement sur l'invariance d'échelle (§8.4). Nous concluerons avec un retour, à la lumière de tous ces résultats, sur le concept de criticalité tel qu'on le rencontre dans diverses branches de la physique (§8.5).

8.1 La percolation

Considérons ce qui se passe lors de la préparation d'un bon café noir, tel qu'illustré schématiquement à la Figure 8.1. Une pompe force le passage d'eau très chaude à travers une masse compacte de fragments de grains de café moulus (en gris). La masse est peut-être compacte, mais elle contient un grand nombre d'interstices (en blanc) à travers desquels l'eau peut s'écouler, capturant au passage —par une forme de diffusion!— l'arôme de café, de telle manière à ce qui sort du bas du dispositif est maintenant le liquide noir que certains d'entre nous apprécient tant. Ce processus d'écoulement d'un fluide à travers un milieu poreux s'appelle **percolation**.

De toute évidence, pour que le percolateur de la Figure 8.1 fonctionne, il doit exister au moins un "passage", soit une suite de vides dans l'amas de fragments de grains de café, qui relie le haut et le bas du système, soit d'où l'eau est injectée jusque là où le café sort. Plus l'amas est dense, plus ce parcour risque d'être tortueux. Les pixels noirs sur la Figure 8.1 représentent un

Figure 8.1: Représentation schématique de la préparation d'un café par percolation. Une eau très chaude est forcée à travers une masse poreuse de fragments de grains de café, produisant la boisson désirée à la sortie. Les "pixels" gris représentent les fragments de grains de café, les blancs des espaces vides, et les noirs l'ensemble des espaces vides formant une connexion entre le haut et le bas du percolateur. Notez l'étroitesse de ce passage à mi-hauteur, ainsi que la présence de nombreux branchements dont certains sont sans issues, mais où d'autres refusionnent plus bas dans le percolateur.

tel parcour, soit le plus grand —et ici le seul— amas de pixels vides contigus reliant le haut au bas du percolateur. La complexité de ce parcour a plusieurs conséquences intéressantes, non la moindre étant le fait qu'elle augmente la surface et le temps de contact entre l'eau et les grains de café durant la percolation. Notez déjà ici qu'à mi-hauteur dans le percolateur, le parcour n'est large que d'un seul pixel; si un seul de ces pixels de ce goulot d'étranglement avait été occupé par un fragment de grain de café, la percolation aurait été bloquée, et si vous insistez trop longtemps vous allez flamber la pompe de votre machine espresso. Alors attention!

Évidemment, plus il y a d'interstices (pixels blancs), plus l'eau percolera facilement. On définit la **porosité** comme le rapport entre le volume occupé par le vide, divisé par celui occupé par les grains. Or voilà le ô combien cruel dilemme pour l'amateur(e) de café: Plus la porosité est faible, plus le café sera fort; mais si la porosité est *trop* faible, alors il n'y aura plus de passage reliant le haut et le bas du percolateur, et rien ne sortira en bas. Il s'agit donc de déterminer le niveau porosité qui sera minimal, tout en permettant au moins un passage ouvert à travers le système. Cette valeur est appelée le **seuil de percolation**, et comme on le verra ci-dessous, en général son calcul est loin d'être simple!

La préparation du café offre un exemple simple et intuitif de la percolation, mais le concept s'applique à bien d'autres branches de la physique. Par exemple, remplacez les pixels gris par une substance non-conductrice, les pixels blancs par un métal conducteur, et on se retrouve avec un modèle de la conductivité électrique pour un alliage; enlevez la substance non-conductrice, et voici un modèle de la conductivité électrique applicable à un milieu granulaire. D'un point de vue plus théorique, la percolation s'avère à représenter un exemple particulièrement simple et facilement simulable d'un système critique.

8.2 Percolation en 1D

Considérons une chaine linéaire de N *sites*, comme sur la Figure 8.2 pour $N = 64$, où chaque site a un voisin à droite et un à sa gauche, sauf évidemment pour les sites aux deux extrémités de la chaine, qui n'ont qu'un seul voisin. Un site peut être soit vide, doit occupé, avec une probabilité d'occupation que nous dénoterons par p (avec $0 \leq p \leq 1$, comme il se doit). La Figure 8.2 montre une séquence de telles chaines, pour une probabilité d'occupation augmentant de 0.1 à 1.0 du bas vers le haut.

Si p est la probabilité qu'un site soit occupé, alors la probabilité qu'il ne le soit pas est $1-p$. Pour un réseau 1D linéaire de composé de N sites, on s'attend à ce que le nombre total de sites occupés devienne égal à $p \times N$ dans la limite $N \to \infty$; et à un N fini, vous ne serez pas surpris d'apprendre (je l'espère) que l'on peut s'attendre à des déviations de nature purement stochastique par rapport à cette valeur attendue. Ceci est bel et bien cohérent avec la Figure 8.2; à $p = 0.6$ et 0.8 on a exactement le nombre attendu de sites occupés, mais à d'autres valeurs de p on observe des déviations, parfois substantielles. Notons en particulier que pour cette réalisation spécifique de la séquence de nombres aléatoires contrôlant l'occupation des sites, le réseau à $p = 0.3$ contient *moins* de sites occupées que le réseau à $p = 0.2$, mais que dans les deux cas ces déviations ne dépassent pas de beaucoup la grandeur anticipée.

Définissons maintenant un **amas** comme un groupe de sites occupés contigus, délimités par un site vide à chaque extrémité. Si la probabilité d'occupation de chaque site est indépendante de l'état des autres sites, alors la probabilité que deux sites contigus soient occupés est p^2, celle que trois sites le soient est p^3, etc; généralisant à s site contigus, on se retrouve avec une probabilité égale à p^s. Maintenant, la probabilité d'avoir au moins un amas de taille s sur le réseau sera égale à cette probabilité p^s, fois la probabilité que le site précédant au coté gauche et suivant au coté droit ne le soient pas. Donc, la probabilité d'avoir au moins un amas de taille s sur le réseau est donnée par:

$$p^s(1-p)^2 \tag{8.1}$$

On constate que cette probabilité tendra toujours vers zéro quand s tend vers l'infini, même dans limite $p \to 1$. En effet, il ne suffit que d'un seul site inoccupé pour segmenter un amas de

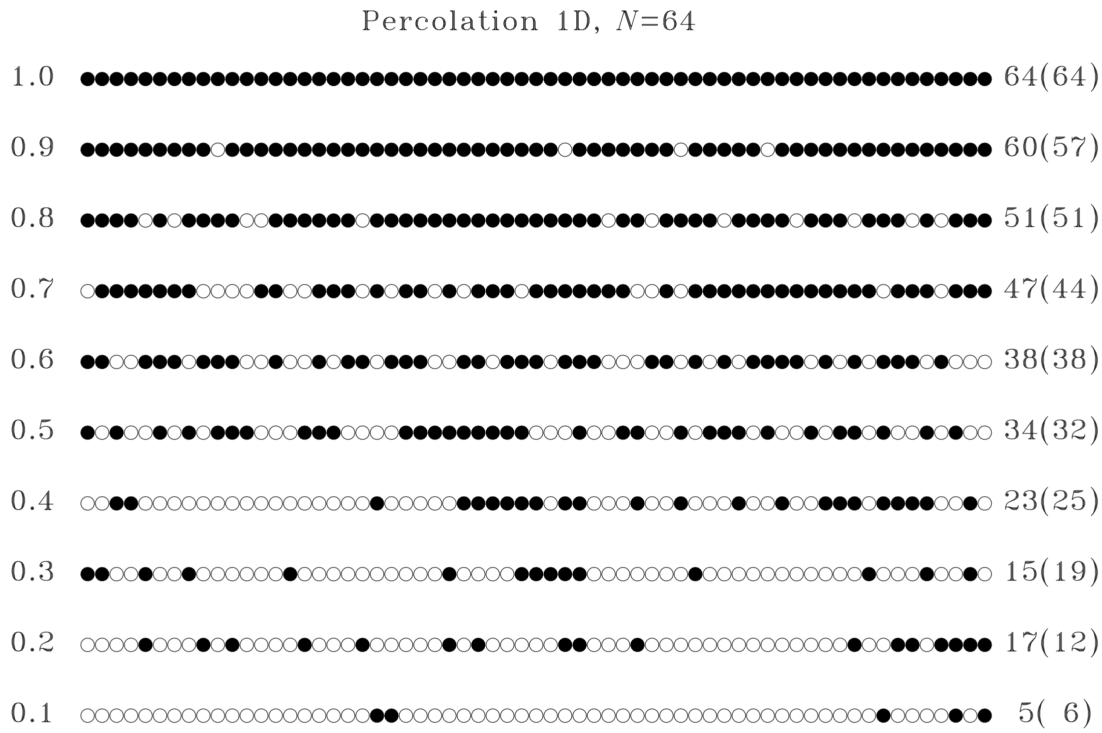


Figure 8.2: Percolation en une dimension spatiale. Chaque ligne représente un réseau 1D de taille $N = 64$, avec les sites occupés en noir et les inoccupés en blanc. La probabilité d'occupation augmente du bas vers le haut de 0.1 jusqu'à 1.0 en sauts de 0.1, tel qu'indiqué dans la colonne de gauche. La colonne de droite donne le nombre de sites occupés, avec la valeur attendue $\texttt{int}(p \times N)$ suivant entre parenthèses.

taille autrement infinie. Cependant il demeure clair qu'à un N donné, la probabilité d'avoir un amas de grande taille augmente avec p, comme on s'y attendrait.

Soit s_k la taille, mesurée en nombre de sites occupés, du k^{e} amas de la chaine, et S la taille du plus grand de ces amas:

$$S = \max(s_k) \ , \qquad k = 1, 2, 3.., K \ . \tag{8.2}$$

Tant que peu de sites sont occupés, on peut s'attendre à ce que la probabilité "d'ajouter" un site à un amas existant augmente linéairement avec p, puisque qu'agrandir un amas implique ajouter un site occupé à une de ses deux extrémités. Les chaines de la Figure 8.2 semblent en accord avec cette attente; S passe de 2 à 4 à 5 à 6 quand p passe de 0.1 à 0.2, 0.3 et 0.4. Mais une fois p suffisamment élevée pour qu'une fraction substantielle des sites soient occupés, un nouveau processus de croissance prend le dessus: la "fusion" de deux amas séparés par un seul site vide, lorsque ce dernier devient occupé. La fusion successive de tels amas, toujours de plus en plus grands, devient de plus en plus fréquente à mesure que p augmente, conduisant à une augmentation très rapide de S en fonction de p, un processus clairement apparent sur la Figure 8.2. Il est en fait possible de démontrer —nous ne le ferons pas ici— que dans la limite $N \to \infty$, la taille du plus grand amas croit avec p selon la relation

$$\lim_{N \to \infty} S = \frac{1+p}{1-p} \ . \tag{8.3}$$

Ceci indique que la taille du plus grand amas devient infinie dans la limite $p \to 1$; autrement dit, en une dimension spatiale le plus grand amas devient de taille égale à la dimension du système. La valeur de p où ceci se produit définit ici le seuil de percolation et sera désormais dénotée p_c. En une dimension spatiale, $p_c = 1$ pour la toute simple raison qu'il ne suffit que d'un seul site vide pour "briser" la chaine. Résultat facile à anticiper, mais dont la version multidimensionnelle devient beaucoup plus subtile... Notons finalement que dans la limite $p \ll 1$, on s'attend à une croissance linéaire de S en fonction de p:

$$\lim_{p \ll 1} \frac{1+p}{1-p} \simeq (1+p)(1+p) = (1+2p+p^2) \simeq 1+2p \ . \tag{8.4}$$

où la première égalité approximative résulte du fait que

$$\frac{1}{1-p} \simeq (1+p) \ , \qquad p \ll 1 \ , \tag{8.5}$$

et la dernière de ce que $p^2 \ll p$ dans la limite $p \ll 1$.

8.3 Percolation en 2D

Passons à deux dimension spatiales, soit du concept de chaine à celui de **réseau**. Un réseau est défini comme une série de site connectés à un certain nombre d'autres sites. Un réseau particulièrement simple est le **réseau cartésien régulier**, où les sites sont répartis en rangées et colonnes équidistantes, un peu comme les cases d'un papier quadrillé ou les pixels du CCD d'une caméra digitale. Nous avons déjà fait connaissance avec un tel type de réseau dans notre étude de la marche aléatoire (§7.4 et Fig. 7.13). Mis à part les sites situés aux bords du réseau, chaque site a quatre **voisin immédiats**, soit les sites situés en bas, en haut, à gauche et à droite; autrement dit, si on dénote par (i, j) le couple (rangée,colonne) d'un site, les voisins immédiats sont

$$(i+1, j) \ , \qquad (i-1, j) \ , \qquad (i, j+1) \ , \qquad (i, j-1) \ . \tag{8.6}$$

Notons que, traditionnellement, les voisins diagonaux

$$(i-1, j-1) \ , \qquad (i-1, j+1) \ , \qquad (i+1, j-1) \ , \qquad (i+1, j+1) \ , \tag{8.7}$$

ne sont pas considérés comme faisant partie des voisins immédiats. Il est évidemment possible de définir des réseaux basés sur des géométries et/ou connectivités différentes, par exemple des réseaux triangulaires à six voisins immédiats (essayez d'en dessiner un), ou un réseau cartésien à huit voisins immédiats (i.e., en incluant les voisins diagonaux). Le réseau cartésien demeure cependant particulièrement populaire, car il s'avère facile à coder, sous la forme d'un tableau à deux dimensions:

```
...
DEFINE N 128            /* taille du reseau */
DEFINE P 0.5            /* probabilite d'occupation */
int main{void}
   ...
   int j, k, reseau[N][N]
   ...
   for (j=0 ; j<N ; j++) {
      for (k=0 ; k<N ; k++) {
         if ( 1.*rand()/2147483647 < P )
            { reseau[j][k]=1 ; }
         else
            { reseau[j][k]=0 ; }
      }
   }
}
```

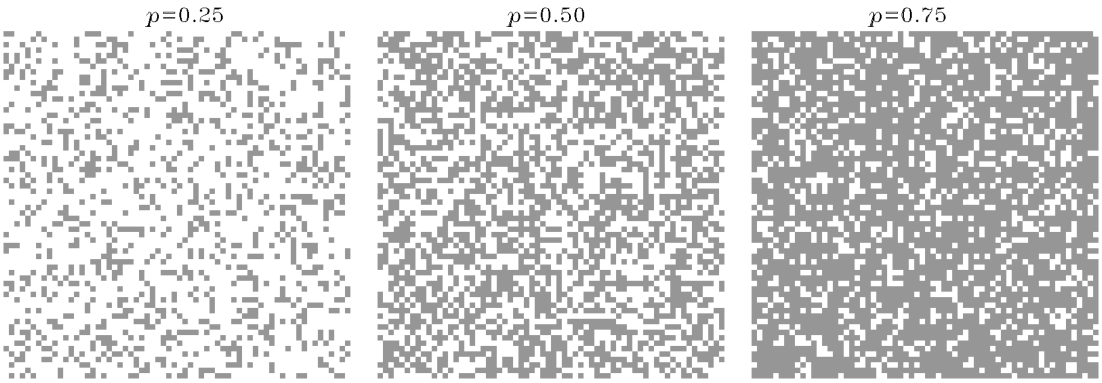


Figure 8.3: Trois réseaux cartésiens réguliers 2D de dimensions $N \times N = 64 \times 64$, pour des probabilités d'occupation $p = 0.25$, 0.5, et 0.75. Les sites occupés sont indiqués en gris, et les sites vides en blanc.

Ce seront les seuls types de réseaux considérés dans tout ce qui suit. Remarquez que dans le fragment de code ci-dessus, on assigne (arbitrairement) à un site vite la valeur (entière) zéro, et à un site occupé la valeur un, et ce sans perte de généralité.

Sur ces réseaux 2D, un amas correspond à un groupe de sites occupés contigus séparés d'autres sites occupés ou amas par des sites vides. Notons que selon la définition de voisinage immédiat introduite ci-dessus, deux sites occupés se touchant par leurs coins ne sont pas considérés comme faisant parti du même amas. Il existe plusieurs types d'algorithmes permettant d'identifier tous les pixels faisant partie du même amas; il n'est pas du tout trivial de faire ça de manière qui soit efficace du point de vue numérique. En fait, nous ferons connaissance avec l'un des plus performant au chapitre suivant. Pour l'instant, nous prendrons simplement pour acquis qu'un tel algorithme existe.

La Figure 8.3 montre trois exemples de réseau cartésiens réguliers en 2D, de dimensions $N \times N = 64 \times 64$ et pour des probabilités d'occupation $p = 0.25$, 0.50 et 0.75. Dans le premier cas, visuellement le réseau prend la forme d'un grand nombre de petits amas ou sites occupés isolés, répartis aléatoirement dans le plan. Il est clair qu'ici aucun amas ne traverse le réseau, et la percolation y serait donc impossible[1]. À $p = 0.75$, on a plutôt l'apparence d'un objet solide contenant plusieurs "trous" de petites dimensions, répartis aléatoirement dans la structure, comme les trous dans un fromage suisse. Ici il existe un très grand amas couvrant l'ensemble du réseau, qui regroupe la quasi-totalité de tous les sites occupés, et où la percolation y serait aisée. Le cas $p = 0.5$ est plus ambigu, du moins visuellement; il est difficile de catégoriser les sites occupés comme un groupe d'amas répartis dans un espace vide, ou un solide très fragmenté. Et il faudrait étudier le réseau avec beaucoup d'attention pour déterminer si la percolation y est possible.

En examinant la Figure 8.3 on voit déjà bien que la taille S du plus grand amas augmente avec la probabilité d'occupation p. La Figure 8.4 illustre cette variation de manière plus quantitative. Chaque point y correspond en fait à la moyenne des tailles du plus gros amas dans chacune de dix simulations indépendantes sur réseau $N \times N = 512 \times 512$, produites avec la même valeur de p mais utilisant un germe différent dans la génération des sites occupés sur le réseau. Les barres d'erreur indiquent la déviation quadratique moyenne associée à l'ensemble de 10 simulations pour chaque valeur moyenne de $S(p)$. On aurait évidemment pu anticiper qu'à mesure que p croit, le plus grand amas du réseau devienne de plus en plus grand, simplement parce que plus de sites occupés sont disponibles pour former un amas. Cependant la variation de S en fonction de p est plutôt particulière. À basse valeur de p, la croissance est encore une fois approximativement linéaire (courbe en tirets), comme en 1D; à partir de

[1]Attention ici; dans notre exemple du café (Fig. 8.1), le liquide s'écoule à travers les sites *inoccupés*, tandis que traditionellement, en *théorie* de la percolation on caractérise le processus en terme d'amas de sites *occupés*.

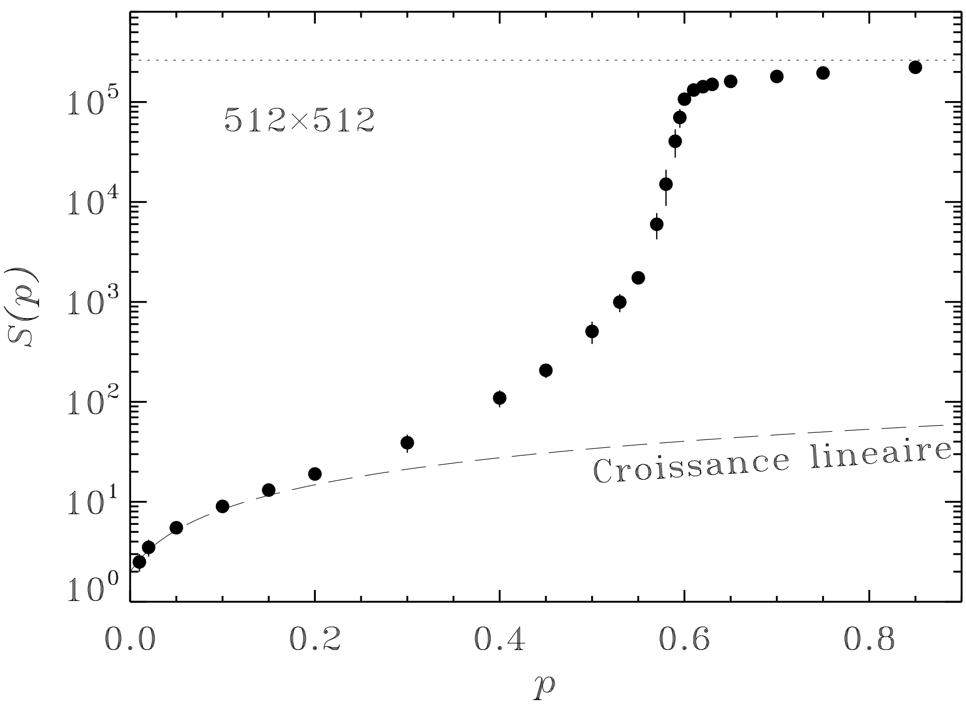


Figure 8.4: Croissance du plus grand amas en fonction de la probabilité d'occupation p, sur un réseau 512×512. Chaque point correpond à la moyenne des tailles du plus grand amas dans un groupe de 10 simulations à la même valeur de p. Le trait en tirets indique une croissance linéaire, qui offre une représentation tolérable des résultats numériques pour $p \lesssim 0.2$. On notera également la croissance super-exponentielle pour $0.2 \lesssim p \lesssim 0.6$. La saturation à $p \gtrsim 0.6$ est une conséquence directe de la taille finie du réseau, qui limite la taille maximale réalisable ici à 512^2 (trait pointillé horizontal).

$p \simeq 0.4$ la croissance devient superexponentielle, puis commence à saturer pour $p \gtrsim 0.6$. La phase de croissance rapide est de nouveau produite par la fusion d'amas distincts mais séparés par un seul site, suite à l'occupation de ce site quand p croit légèrement. La Figure 8.5 illustre cette rapide croissance du plus grand amas, sur un réseau de dimensions 512×512, lorsque la probabilité d'occupation passe de $p = 0.57$ à 0.6 par saut de 0.01. La saturation se produit lorsque le plus grand amas atteint une taille qui devient comparable à celle du réseau, indiquée par la ligne pointillée sur la Figure, qui pose évidemment une limite supérieure absolue à la taille du plus grand amas sur un réseau de taille finie. Notez aussi que quand on approche la saturation, le plus grand amas commence à révéler la forme globale du réseau —ici carrée,— ce qui n'est pas le cas à plus basses valeurs de p.

La forme des amas sur la Figure 8.5 mérite qu'on s'y attarde. Considérons en particulier le cas $p = 0.59$, en bas à gauche. La Figure 8.6 reproduit cet amas, toujours en noir, avec les prochains 660 plus grands amas du réseau, colorés arbitrairement. Le plus grand amas est très filamentaire, et est caractérisé par plusieurs trous qui eux-même contiennent des amas de plus petite taille, avec leurs propres trous contenant également des amas encore plus petits, et ainsi de suite jusqu'à l'échelle des sites individuels du réseau. Tous ça sent l'invariance d'échelle, déjà discutée à la §7.6 dans le contexte de la nature fractale des structures produites par aggrégation. Vous aurez déjà deviné que près du seuil de percolation, les amas sont également des structures de dimensions fractales entre 1 et 2. La Figure 8.7 illustre la fonction de densité de probabilité (FDP) de la taille des amas sur le réseau, i.e. la fonction $f(s)$ telle que $f(s)\mathrm{d}s$ donne la probabilité qu'un amas ait une taille comprise entre s et $s + \mathrm{d}s$, pour des simulations sur réseaux $N \times N = 512 \times 512$ ayant $p = 0.3$, 0.59 et 0.7. Dans le premier cas, la FDP

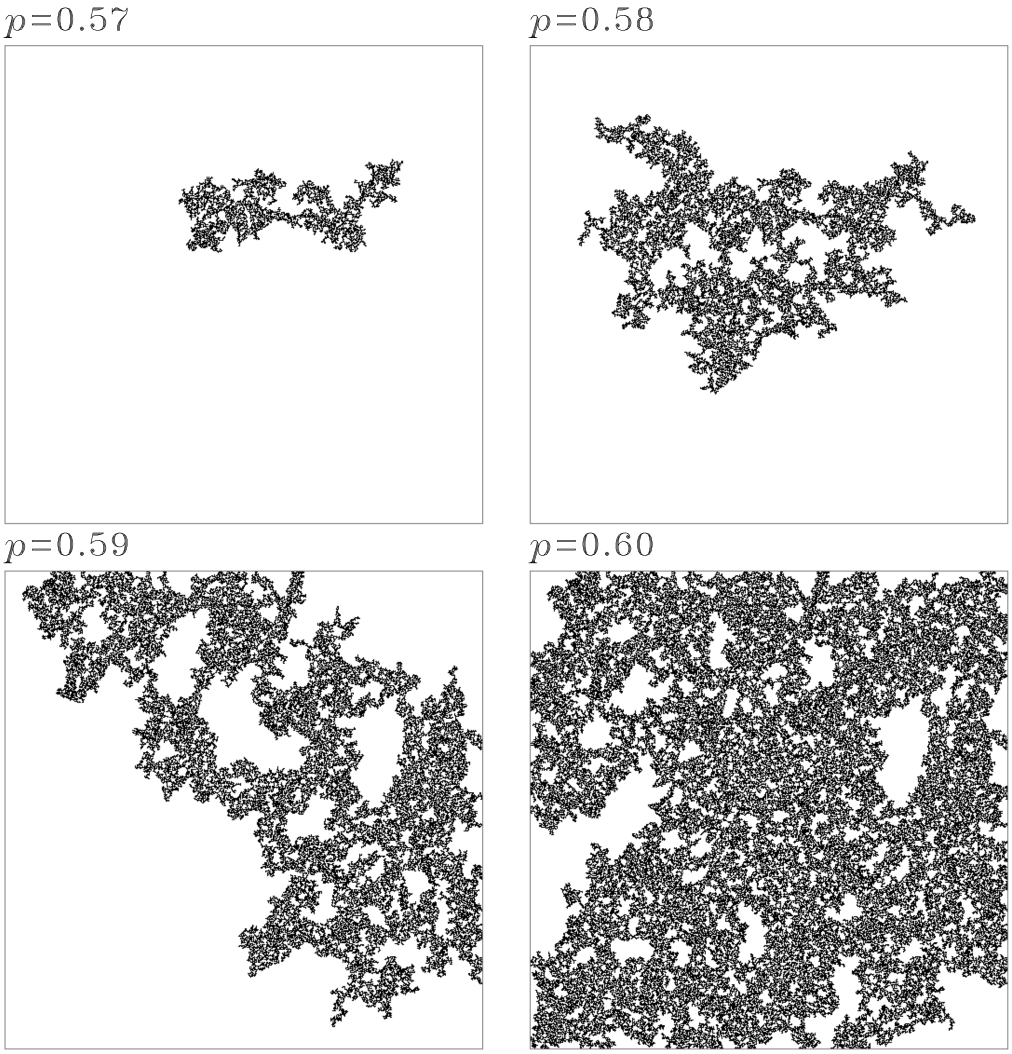


Figure 8.5: Les plus grands amas dans sur un réseau 512×512, pour des probabilités d'occupation $p = 0.57$, 0.58, 0.59 et 0.6. Cette faible augmentation de p conduit à une croissance très rapide du plus grand amas. Typiquement, à $p = 0.57$ le plus grand amas est pas mal plus petit que la taille du réseau, mais atteint une longueur comparable au réseau à $p = 0.59$, et déjà à $p = 0.6$ ressemble à une éponge couvrant les deux dimensions du réseau jusqu'à ses frontières.

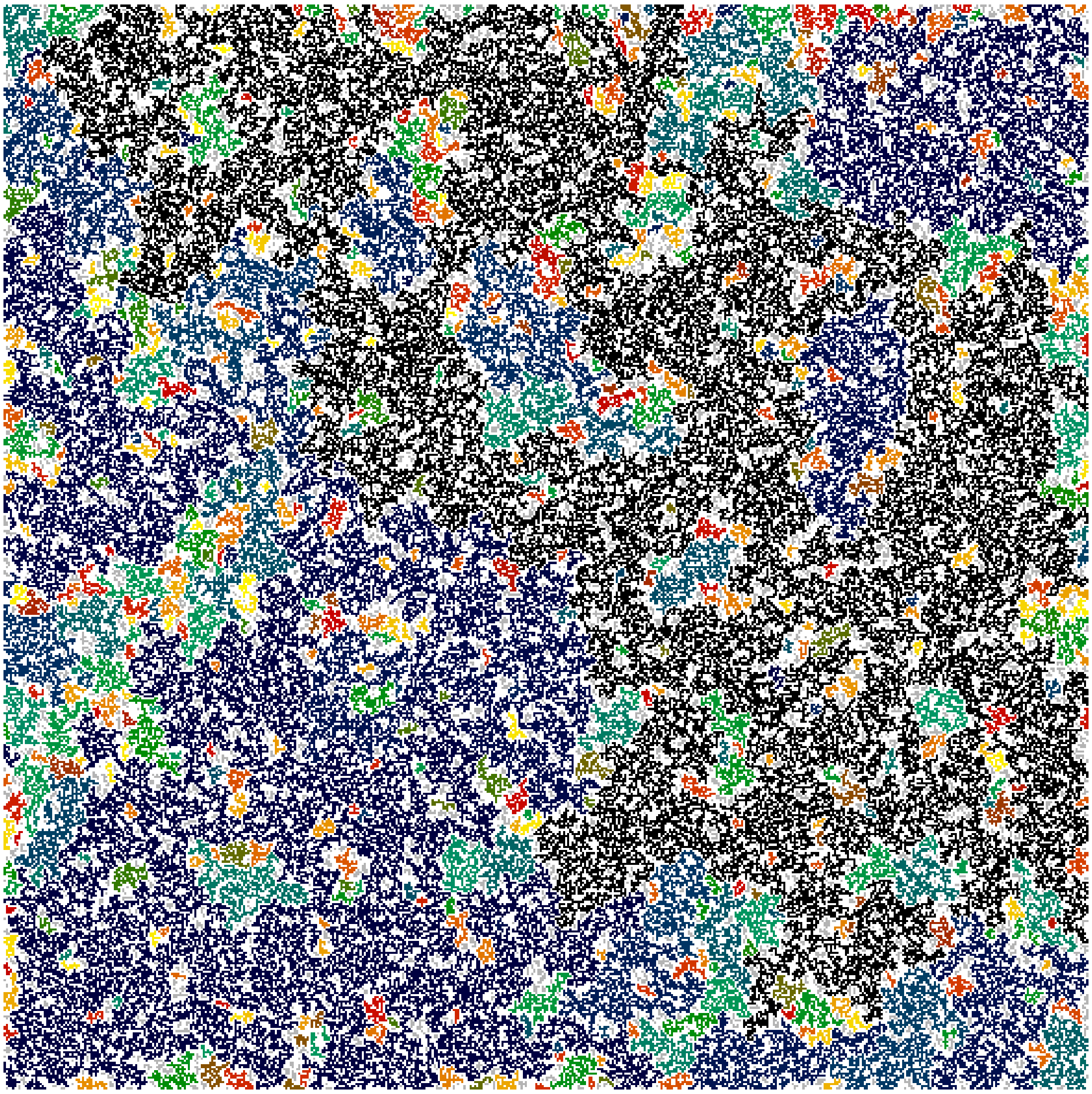


Figure 8.6: Les 661 plus grand amas sur un réseau de dimension $N \times N = 512 \times 512$, à $p = 0.59$. Les sites laissés en blanc sont vides, et les sites en gris sont occupés mais ne font pas partie de ces 661 plus grands amas. Le plus grand amas de tous est tracé en noir; il regroupe $S = 53537$ des 154867 sites occupés ici, et traverse tout le réseau. Les trous dans les grands amas contiennent d'autres amas, eux-même montrant des trous contenant d'autres amas encore plus petits, et ainsi de suite jusqu'aux sites occupés isolés.

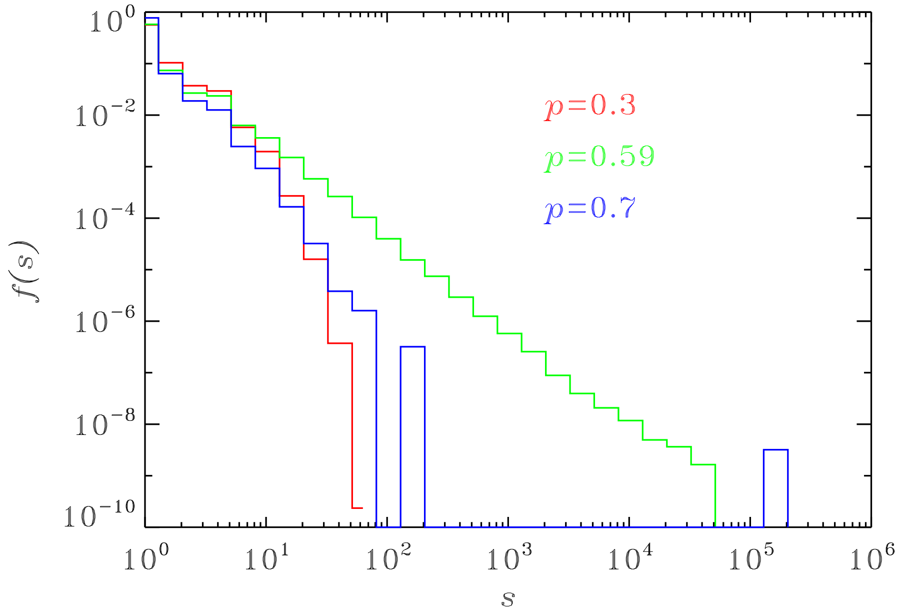


Figure 8.7: Fonctions de densité de probabilité de la taille des amas sur un réseau 512×512, pour $p = 0.3$ (rouge), 0.59 (vert) et 0.7 (bleu). Noter comment les FDP des amas pour $p = 0.3$ et 0.7 se ressemblent grandement, sauf évidemment pour la présence dans le second cas d'un super-amas regroupant plus de 10^5 sites occupés, correspondant à la petite colonne d'histogramme solitaire bleue dans le coin inférieur droit du graphique. Ces FDPs sont calculées à partir de 10 réalisations indépendantes de l'occupation du réseau à chaque valeur de p.

chute rapidement quand s augmente, réflexion du fait que le réseau est dominé par les amas de petites tailles (cf. Fig. 8.3). À $p = 0.7$, il existe un gigantesque amas qui regroupe presque tous les sites occupés; les quelques autres amas contenus dans les trous de ce super-amas-éponge étant de beaucoup plus petite taille, leur FDP ressemblent en fait fortement à cette du réseau à $p = 0.3$. Le cas à $p = 0.59$ est particulièrement intéressant car la FDP prend la forme d'une loi de puissance, i.e.,

$$f(s) = s_0 s^{-\alpha} , \qquad \alpha > 0 . \tag{8.8}$$

avec ici $\alpha \simeq 1.85$. Ceci indique qu'il n'y a pas d'échelle caractéristique à la taille des amas.

Nous nous attarderons plus longuement à ces questions d'invariance d'échelle un peu plus loin. Pour le moment, revenons à la Figure 8.4; il est clair que la saturation de la taille du plus grand amas une fois que p dépasse 0.65 environ est une conséquence directe du fait que l'amas ne peut plus "grandir" plus loin que les frontières du réseau. Que se passerait-il dans une situation d'un réseau de très, très grande taille, ou même dans la limite $N \to \infty$? On s'attendrait à un niveau de saturation tendant lui-même vers l'infini, et donc une variation extrêmement rapide de S avec p au moment de cette transition.

Simuler sur un réseau de taille $N \to \infty$, c'est long... Mais il demeure possible d'étudier certains aspects de ces comportements apparamment divergents à l'aide de simulations sur réseaux de tailles finies. Par exemple, La Figure 8.8 montre les mêmes résultats numériques, que sur la Fig. 8.4, sauf que cette fois ce qui est porté en graphique en fonction de p est la quantité

$$F(p) = \frac{S(p)}{pN^2} , \tag{8.9}$$

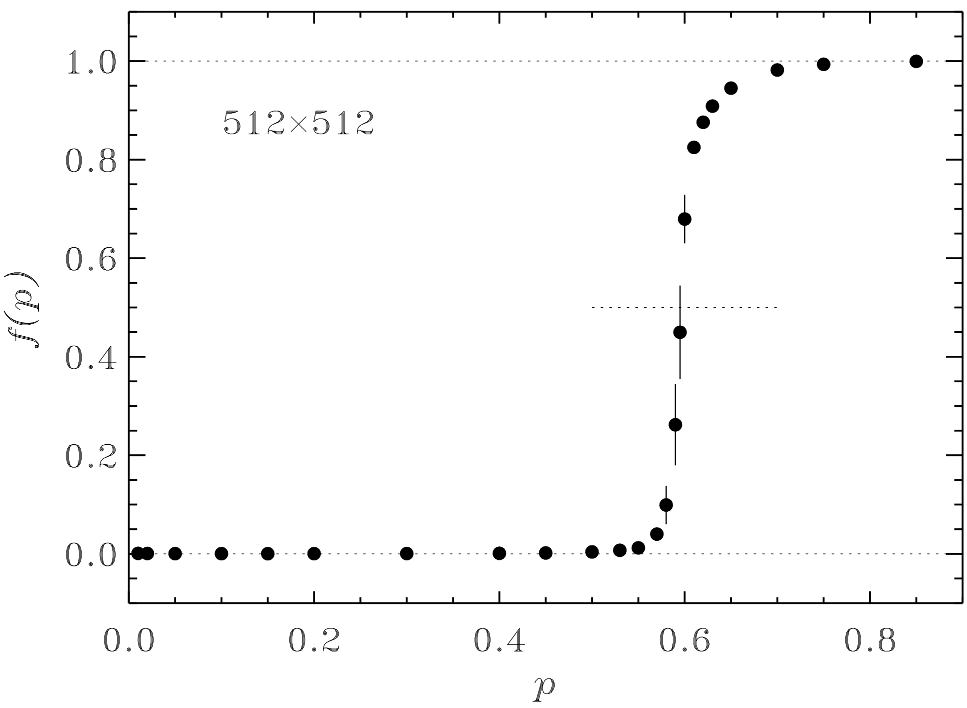


Figure 8.8: Identique à la Figure 8.4, sauf que cette fois ce qui est porté en graphique est le rapport F entre le nombre de sites occupés faisant partie du plus grand amas et le nombre total de sites occupés. Le tiret horizontal pointillé indique la valeur $F(p) = 1/2$.

soit la fraction des sites occupés qui font partie du plus grand amas. On y observe un comportement fort intéressant: tant que $p \lesssim 0.5$, cette quantité est effectivement zéro, *même si la taille du plus grand amas croit substantiellement* (cf. Fig. 8.4); autrement dit, le plus grand amas du réeau ne se distingue pas particulièrement des autres grands amas présents. À l'autre extrême, pour $p \gtrsim 0.65$, le plus grand amas a effectivement cannibalisé tous les autres, et regroupe la quasi-totalité des sites occupés. Ce qui est frappant est que la transition entre ces deux régimes se produit à l'intérieur d'un très faible intervalle en p. Vers $p = 0.55$, $F(p)$ se met à augmenter très rapidement, pour saturer près de un déjà à $p \simeq 0.7$. En fait, un peu sous $p = 0.6$ la croissance de S semble diverger, dans le sens que $\mathrm{d}F/\mathrm{d}p \to \infty$. La valeur exacte de p à laquelle ceci se produit définit ici le seuil de percolation p_c pour ce réseau 2D. On notera sur la Figure 8.8 qu'à ce seuil de percolation, le plus grand amas regroupe la moitié des sites occupés, i.e.,

$$S(p_c) = \frac{1}{2} p_c N^2 \; . \tag{8.10}$$

La divergence de S dans la limite $p \to p_c$ ressemble au comportement observé en 1D dans la limite $p \to 1$, sauf que cette fois le seuil de percolation apparait à une valeur de p nettement plus petite que un. Un minimum de réflexion révèle pourquoi: en 1D, il n'y a aucune façon de "contourner" un site vide, tandis qu'un 2D et plus, cette possibilité existe, comme la forme très complexe des amas de la Figure 8.5 l'illustre clairement. Pour un réseau cartésien régulier à quatre voisin immédiat, le seuil de percolation se trouve à $p_c = 0.592746$. Contrairement au cas 1D, il n'existe ici aucun équivalent de l'éq. (8.3), cette valeur du seuil p_c doit être déterminée numériquement, et s'avère différente pour différents types de réseaux, comme on peut le constater en examinant la compilation présentée au Tableau 8.1.

Bien qu'aucune théorie existante ne permette de calculer analytiquement les valeurs des seuils de percolation autrement qu'en 1D, certaines tendances générales intuitivement raisonnables se dégagent du Tableau 8.1: plus la connectivité et/ou la dimensionalité spatiale sont élevées,

Table 8.1: Seuil de percolation sur différents types de réseaux

Dimension	Géométrie	Connectivité	p_c
1	Linéaire	2	1.0
2	Carré	4	0.592746
2	Triangulaire	6	0.5
2	Hexagonal	3	0.6962
3	Cubique	6	0.3116

i.e., plus il y a de voisins immédiats, plus le seuil de percolation est bas. En effet, à nombre égal de sites occupés, une dimensionalité et/ou connectivité plus élevée offre plus de possibilités de contourner un site vide.

Au seuil de percolation p_c, et seulement au seuil, les énoncés suivants sont tous mutuellement équivalents:

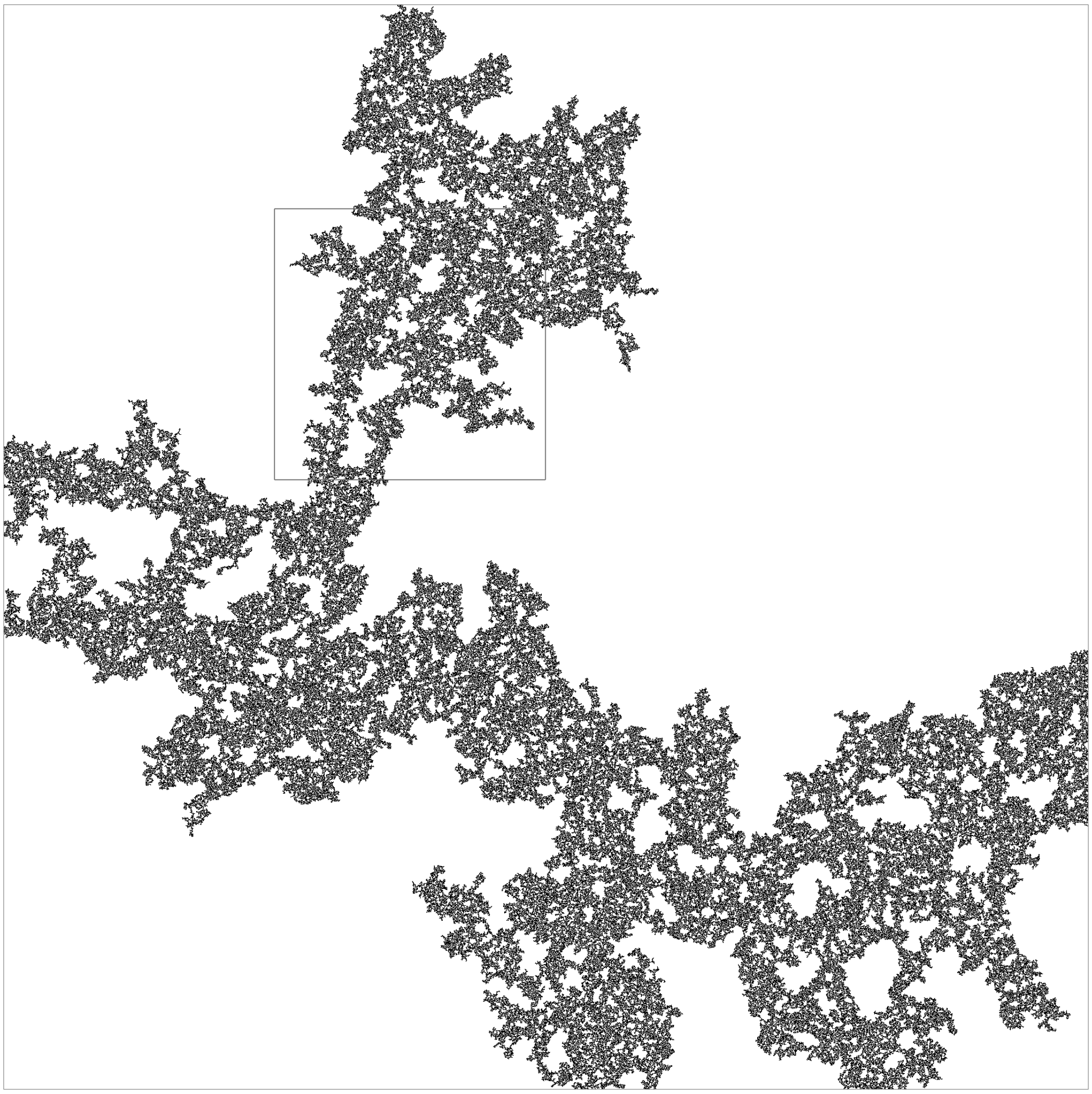
1. La distribution des tailles de tous les amas présents sur le réseau prend la forme d'une loi de puissance;
2. La dimension linéaire du plus grand amas $\simeq N$;
3. Le plus grand amas contient une fraction $F = 0.5$ de tous les sites occupés;
4. $\mathrm{d}S/\mathrm{d}p \to \infty$ dans la limite $p \to p_c$
5. La variance associée à la taille moyenne du plus grand amas dans un ensemble de simulations atteint sa valeur maximale.
6. La dimension fractale du plus grand amas est minimale.

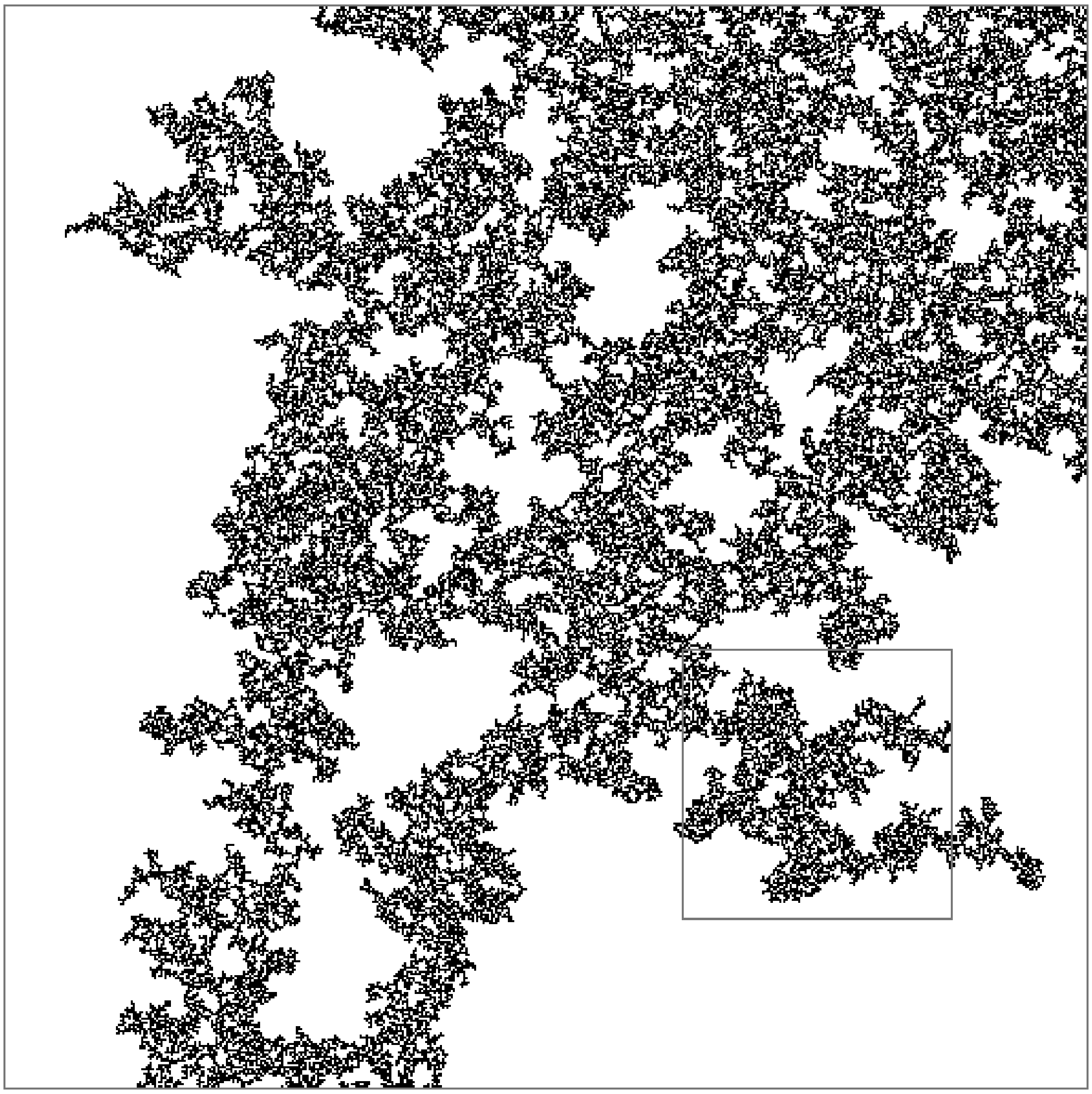
Bon, c'est peut-être bien amusant tout ça, mais qu'est-ce-que la percolation a à voir avec la criticalité? Le lien vient évidemment du comportement du plus grand amas dans la limite $p \to p_c$. Tout comme, dans le cas du chaudron d'eau presque bouillante où l'ajout d'une quantité infinitésimale de chaleur déclenche l'ébullition dans tout le chaudron, ici l'ajout d'un seul site occupé peut former, par fusion de deux amas préexistants, un amas traversant tout le réseau; ou encore, dans le contexte de la conductivité électrique, on passe "instantanément" d'une objet isolant à un objet conducteur. La caractéristique fondamentale d'un système dit critique est donc la suivante: il doit exister un **paramètre de contrôle** (la probabilité d'occupation p dans le contexte de la percolation) qui peut prendre une **valeur critique** où la sensibilité du système par rapport à toute variation de ce paramètre de contrôle devient extrême[2]. De surcroit, la manifestation du comportement critique demande un **ajustement fin** de la valeur du paramètre de contrôle; en percolation sur un très grand réseau 2D à quatre voisin immédiats, choisir $p = 0.59$, $p = 0.592$, $p = 0.5927$ ou $p = 0.59274$ fait toute une différence au niveau du comportement global du système.

8.4 Invariance d'échelle et universalité

Il a déjà été mentionné que les amas présents sur le réseau ont une géométrie fractale, conséquence directe de l'invariance d'échelle caractérisant notre définition d'un amas: un grand amas peut être vu comme la fusion de plus petits amas, eux-mêmes représentant la fusion d'amas encore plus petits, et ainsi de suite jusqu'à l'échelle des sites individuels. Examinez bien la Figure 8.9, montrant deux zooms successifs sur le plus grand amas d'un réseau 2048×2048 à $p = 0.59$, pour vous convaincre que c'en est bien le cas. On pourrait donc s'attendre à ce que certaine pro-

[2] certains diraient "infinie", mais je n'aime pas l'usage, parce que l'infini, c'est vraiment très grand...





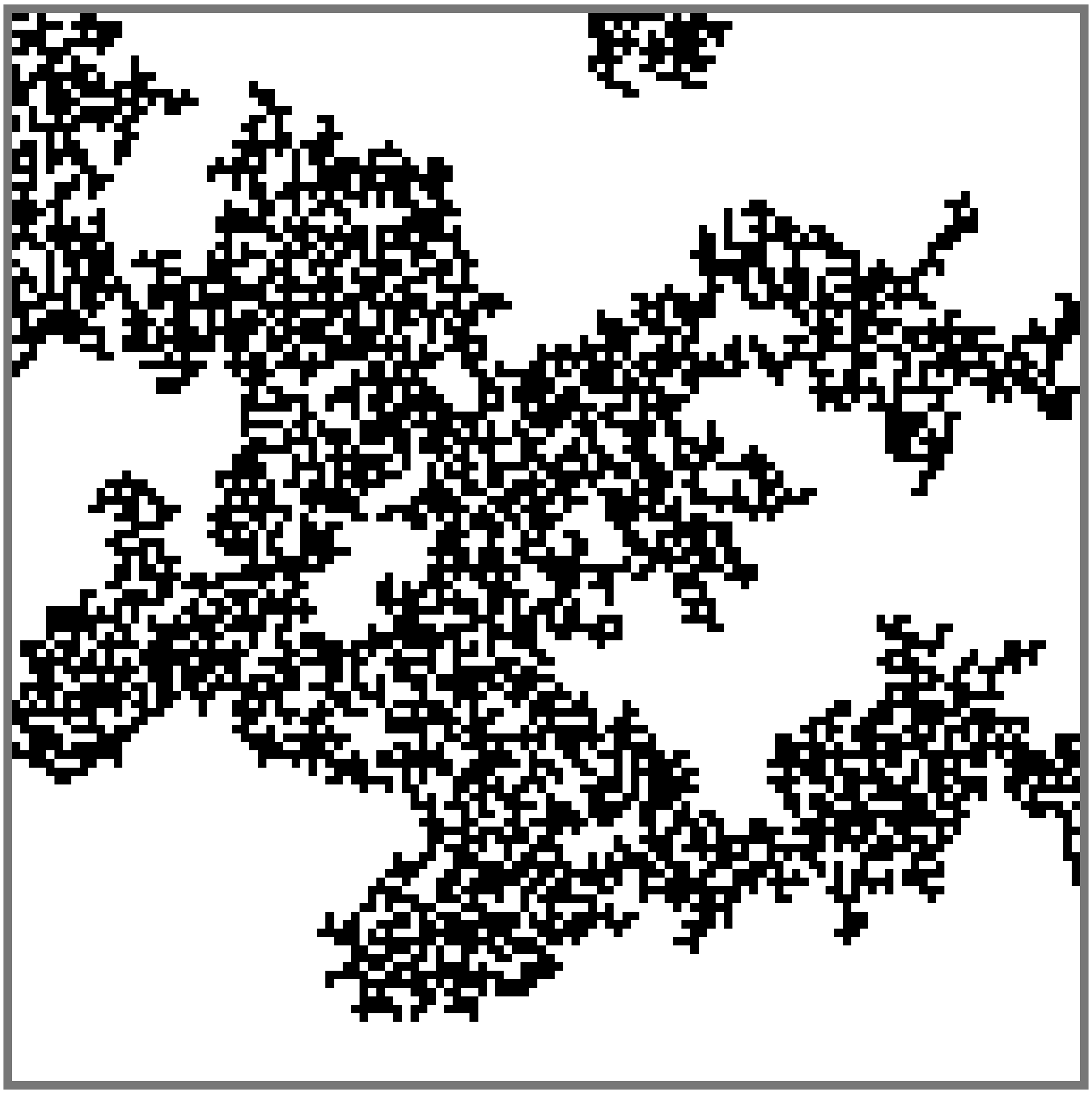


Figure 8.9: [Commence deux pages auparavant] Le plus grand amas dans une simulation 2048×2048, avec deux agrandissements successifs par un facteur 4, chaque encadré sur les deux premières Figures indiquant la région couverte par la suivante. Notez comment l'ilôt fractal ci-dessus n'est joint au corp de l'amas que par un seul site, à son extrémité Nord-Ouest. Notez également, à toutes les échelles, la présence de "trous" dont les formes sont également fractales.

priétés des amas soient indépendantes de la taille du réseau, tant qu'ils ont une taille inférieure aux dimensions de ce dernier.

La Figure 8.10 présente les équivalents des Figures 8.4 et 8.8 , mais cette fois sur trois tailles de réseau, soit $N \times N = 128 \times 128$, 256×256 et 512×512. On voit bien en (A) que la croissance du plus grand amas en fonction de p est indépendante de la taille du réseau, jusqu'à saturation de la courbe, qui se produit à un niveau qui lui augmente évidemment avec la taille du réseau. Cependant, quand la taille du plus grand amas est normalisée par le nombre total de sites occupés, en (B), les trois courbes deviennent impossibles à distinguer, à part pour le fait que les variances des tailles moyennes diminuent pour des réseaux de tailles de plus en plus grandes.

On a vu précédemment qu'au seuil de percolation, les tailles des amas se distribuent selon une loi de puissance:

$$f(s) = s_0 s^{-\alpha} , \qquad \alpha > 0 . \tag{8.11}$$

La fonction de densité de probabilité $f(s)$ mesure ici la probabilité que la taille s tombe entre s et $s + \mathrm{d}s$. On se rappellera que ces fonctions sont normalisées, dans le sens que la probabilité totale doit être unitaire:

$$\int f(s)\,\mathrm{d}s = 1 , \tag{8.12}$$

ce qui détermine effectivement la valeur de la constante s_0 apparaissant dans l'éq. (8.11). Dans notre modèle de percolation 2D, la taille moyenne $\langle s \rangle$ des amas est alors donnée par l'intégrale suivante (cf. §5.2):

$$\langle s \rangle = \int_1^S s\,f(s)\,\mathrm{d}s, \tag{8.13}$$

où la borne d'intégration inférieure correspond au plus petit amas représentable sur le réseau ($s = 1$), et la borne supérieure au plus grand ($s = S$ selon la notation introduite précédemment). Substituant l'éq. (8.11) dans l'éq. (8.13), on obtient:

$$\langle s \rangle = s_0 \int_1^S s \times s^{-\alpha}\mathrm{d}s = s_0 \int_1^S s^{1-\alpha}\mathrm{d}s$$

$$= s_0 \left[\frac{s^{2-\alpha}}{2-\alpha} \right]_1^S = \frac{s_0}{2-\alpha} \left[S^{2-\alpha} - 1 \right] . \tag{8.14}$$

Si $S \gg 1$ (et c'est certainement le cas au seuil de percolation sur un réseau de grande de taille), on se retrouve dans la situation suivante:

1. $\alpha < 2$; l'exposant $2 - \alpha$ est alors positif, et donc l'évaluation de la taille moyenne est déterminée par la taille du plus grand amas présent sur le réseau:

$$\langle s \rangle \simeq \frac{s_0 S^{2-\alpha}}{2-\alpha} , \qquad [\alpha < 2] . \tag{8.15}$$

2. $\alpha > 2$; l'exposant $2 - \alpha$ est alors négatif, et donc la moyenne est dominée par les plus petits amas, via leur contribution à l'évaluation de l'intégrale:

$$\langle s \rangle \simeq \frac{s_0}{(\alpha - 2)} , \qquad [\alpha > 2] . \tag{8.16}$$

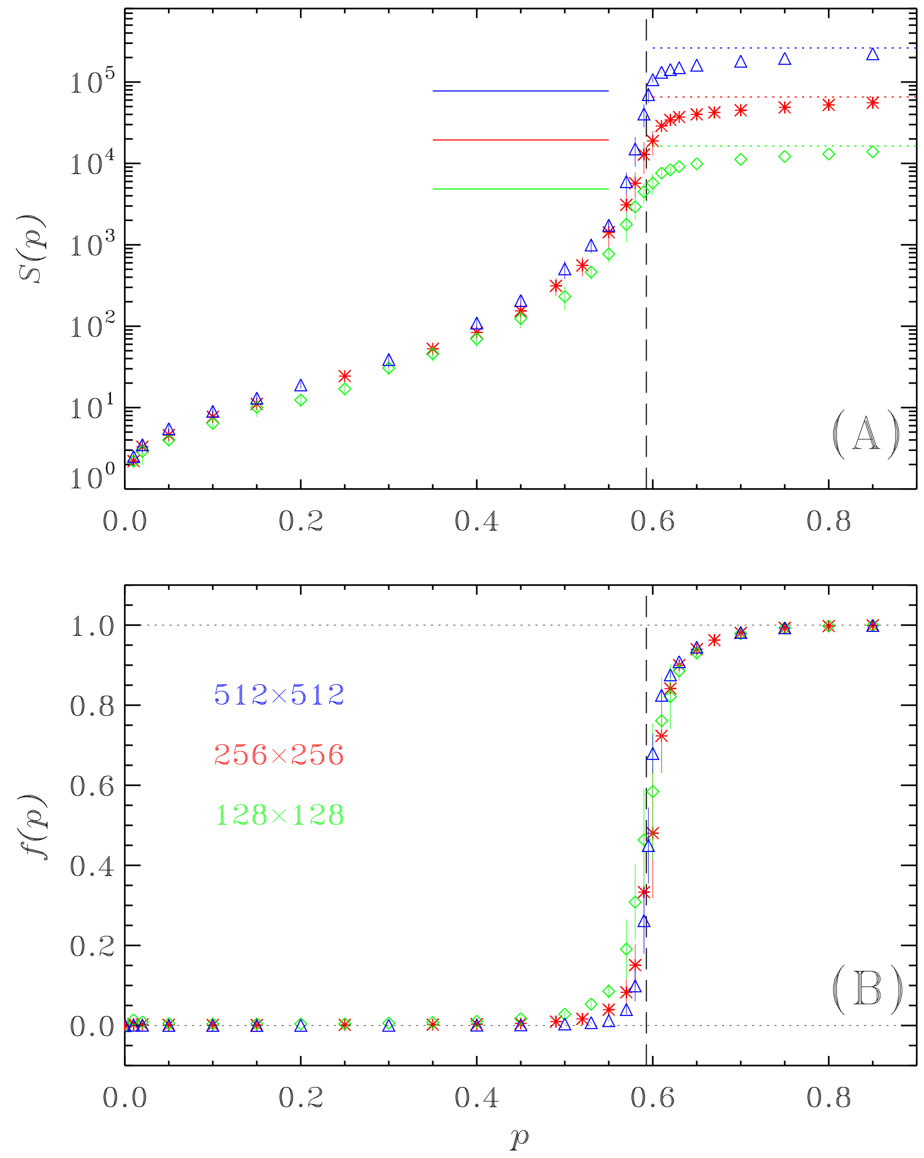


Figure 8.10: Équivalent des Figures (A) 8.4 et (B) 8.8 , sur des réseaux $N \times N = 128 \times 128$, 256×256 et 512×512, tel qu'indiqué par le code couleur. La ligne verticale en tiret indique la valeur $p_c = 0.592746$ du seuil de percolation sur un réseau de dimensions infinies, et les tirets horizontaux en (A) indiquent les tailles attendues du plus gros amas sur chaque réseau.

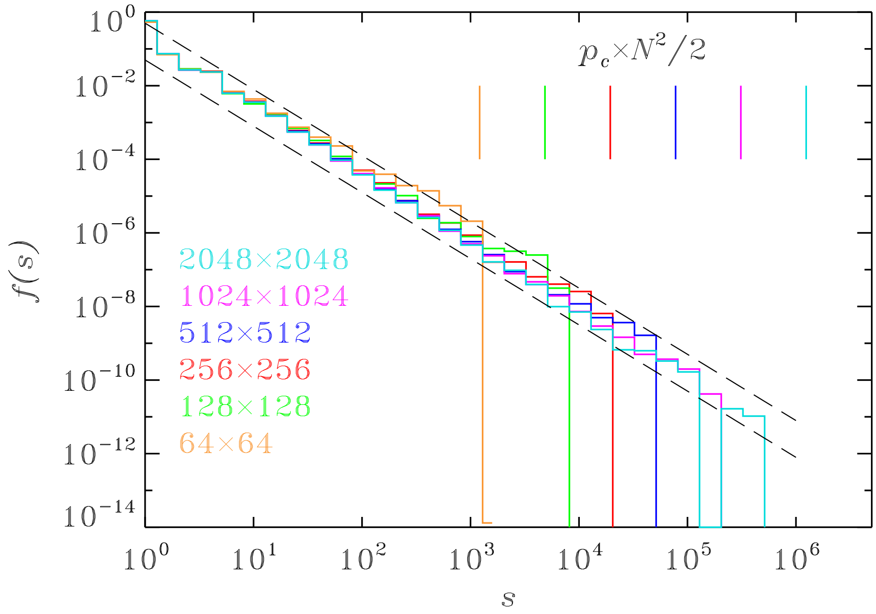


Figure 8.11: Distribution des tailles des amas très près du seuil de percolation ($p = 0.59$), pour un ensemble de 10 simulations par réseau, sur des réseaux allant de $N \times N = 64 \times 64$ à 2048×2048, tel qu'indiqué par le code couleur. Les tirets verticaux colorés en haut à droite indiquent la taille attendue ($S = p_c N^2/2$) du plus grand amas sur chaque taille réseau. Les deux droites en tirets indiquent une pente logarithmique de -1.85; Cette valeur s'avère universelle, dans le sens qu'elle est indépendante de la taille du réseau.

Le cas particulier $\alpha = 2$ est laissé en exercice! Mais il est clair que l'exposant α capture quelquechose de fondamental ici. Bien que ce ne soit pas nécessairement évident *a priori*, la valeur de cet exposant est indépendante de la taille du réseau, comme le montre clairement la Figure 8.11: mis à part les étendues des distributions vers les grandes tailles, les amas au seuil de percolation se distribuent en loi de puissance avec $\alpha = 1.85$, autant sur un maillage 64×64 que sur un réseau 2048×2048, comptant 1024 fois plus de sites. La valeur $\alpha = -1.85$ est dite **universelle** pour la classe des réseaux cartésiens réguliers en deux dimensions spatiales.

On a vu que la croissance de la taille S du plus grand amas avec p semblait également indépendante de la taille du réseau. Cette propriété est capturée par la relation mathématique suivante:

$$\lim_{p \to p_c} S(p) \propto (p_c - p)^{-\beta} , \qquad \beta > 0 , \tag{8.17}$$

où β est un **exposant critique** dont la valeur est encore une fois universelle. Je vous laisse vérifier que pour $\beta > 0$, $\mathrm{d}S/\mathrm{d}p$ diverge bien dans la limite $p \to p_c$.

Déterminer la valeur numérique de l'exposant critique β à partir de simulations sur réseau est facile en principe; on n'a qu'à porter en graphique log-log la taille maximale moyenne des plus gros amas en fonction de $p_c - p$, identifier l'intervalle de $p_c - p$ où la variation apparait linéaire, y ajuster une droite et en mesurer la pente. La Figure 8.12 montre le résultat d'un tel exercice, pour une série de 10 simulations sur un réseau de taille 512×512. Dans l'intervalle $0.01 \lesssim p_c - p \lesssim 0.1$, une droite de pente -1.67 (donc $\beta = 5/3$) offre une bonne représentation des résultats numériques; cependant très près du seuil de percolation ($p_c - p \lesssim 10^{-2}$, disons), les résultats numériques décrochent nettement de la loi de puissance; il s'agit ici d'une conséquence

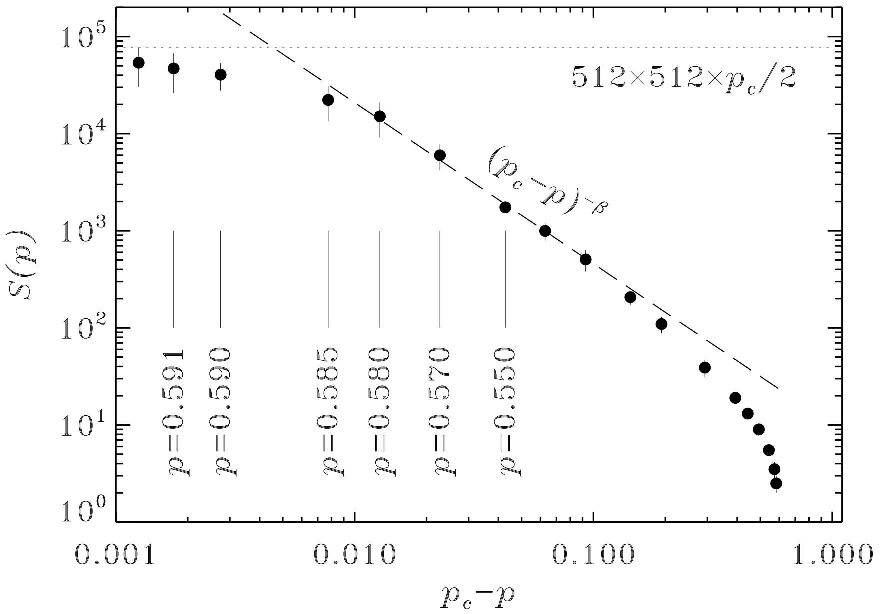


Figure 8.12: Détermination de l'exposant critique β à partir de résultats numériques sur réseau 512×512. Le trait pointillé horizontal indique la taille moyenne attendue du plus grand amas au seuil de percolation, et la droite en tirets correspond à l'éq. (8.17) avec $\beta = 5/3$, ajusté aux données dans l'intervalle $0.01 \leq p_c - p \lesssim 0.1$.

de la taille finie du réseau; au seuil de percolation, on s'attend à un plus grand amas de taille moyenne $S = N^2 \times p_c/2$ (ligne pointillée horizontale sur la Fig. 8.12). Le problème se traduit graphiquement par le fait que la loi de puissance donnée par l'éq. (8.17), demeurant en théorie valide dans la limite $p_c - p \to 0$, intersecte le trait pointillé horizontal à une valeur finie, $p_c - p \simeq 0.047$ ici. Ce point d'intersection marque le bris de l'invariance d'échelle, puisque c'est normalement au seuil de percolation $(p - p_c)$ que le plus grand amas a une taille $S = N^2 \times p_c/2$. Tout les points situés à gauche de ce point d'intersection sur la Fig. (8.12) sont donc *de facto* invalides par rapport à l'éq. (8.17), et un examen attentif de la Figure montre que déjà à $p = 0.585$ le bris de l'invariance d'échelle commence à se faire sentir. Sur un tel diagramme log-log, $p_c - p = 0$ c'est à l'infini dans la direction gauche. La seule possibilité pour éviter l'intersection des droites en tirets et pointilés est donc de laisser cette dernière monter à l'infini vers le haut... ce qui implique $N \to \infty$, soit un réseau de taille infinie. On en conclut que sur un réseau de taille finie, *un bris de l'invariance d'échelle apparaitra inévitablement toujours avant le seuil de percolation.*

8.5 Systèmes critiques en physique

Notre étude de la percolation a été motivée en début de chapitre par quelques phénomènes critiques d'intérêt en physique, soit les transitions de phase en thermodynamique, l'écoulement d'un fluide dans un milieu poreux, et la conductivité électrique d'un alliage ou d'un milieu granulaire. Il en existe plusieurs autres, incluant:

1. Le passage du paramagnétisme au ferromagnétisme au point de Curie;

2. la polymérisation de liquides colloidaux (comme quand on cuit du blanc d'oeuf!);

3. la transition de l'état amorphe à un état cristallin structuré, ou entre diverses configurations cristallines, lorsque les propriétés macroscopique (pression, température) changent;
4. l'apparition de la superfluidité dans l'Helium liquide;
5. Le développement de fissures dans un solide soumis à un stress mécanique

Dans le plus ésotérique —et essentiellement invérifiable expérimentalement,— on pourrait ajouter les bris de symétries des Lois de la Physique que certains spéculent ont eu lieu dans les premiers instants suivant le Big Bang, produisant supposémment les périodes dites inflationnaires de l'expansion de l'univers.

Dans tous les cas l'intéret vient de l'extrême sensibilité du comportement global par rapport à une variation d'un paramètre de contrôle, phénomène qui ne se manifeste que pour un ajustement très fin de ce paramètre à sa valeur critique. La nécessité d'un tel ajustement fin, habituellement par un mécanisme de contrôle extérieur au système, pourrait laisser croire que la criticalité est un phénomène très particulier ne se matérialisant que très rarement dans un système "naturel", où aucun agent extérieur n'est là pour ajuster le paramètre de contrôle. Il existe cependant une vaste gamme de systèmes naturels pouvant atteindre leur état critique et y rester via leur dynamique interne, i.e., sans un tel ajustement extérieur d'un paramètre de contrôle; de tels systèmes sont dits en **autorégulation critique**, et vous n'aurez pas à attendre plus loin que le prochain chapitre pour en rencontrer quelques-uns!

Exercices:

1. Dessinez un réseau hexagonal à connectivité 3, et un réseau triangulaire à connectivité 6. Comment caseriez vous ces réseaux dans un tableau 2D en langage C?
2. Générez des réseaux 2D de tailles $N = 64$, $N = 256$ et $N = 1024$, à $p = 0.5$, selon la procédure décrite à la §8.3. Mesurez ensuite le nombre de sites occupés. Répétez le processus dix fois pour chaque taille de réseau, et calculez les valeurs valeurs moyenne et variance du nombre de sites occupés pour chaque valeur de N, et comparez à la valeur attendue pN^2.
3. Calculez la valeur moyenne d'une variable dont la fonction de densité de probabilité prend la forme d'une loi de puissance d'indice -2.

Bibliographie:

Ce chapitre est de mon cru, bien que beaucoup ait été écrit sur la percolation, et évidemment encore plus sur la criticalité. En particulier, à ce jour le grand classique sur la théorie de la percolation demeure:

Stauffer, D., & Aharony, A., *Introduction to percolation theory*, 2^{e} éd., Taylor & Francis (1994).

Il y a également beaucoup à apprendre de l'ouvrage suivant, quoiqu'il soit à un niveau mathématique et physique passablement plus élevé que celui de ce cours:

Sornette, D., *Critical phenomena in natural sciences*, Springer (2000);

le chapitre 12 y traite spécifiquement de percolation, mais les premiers chapitres contiennent une masse d'informations pertinentes aux systèmes critiques en physique, et à la statistique de variables distribuées en lois de puissance. Attention aux pages de Wikipedia sur la percolation, elles ne me paraissent pas particulièrement impressionnantes... (juillet 2011).

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;

```c
#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.

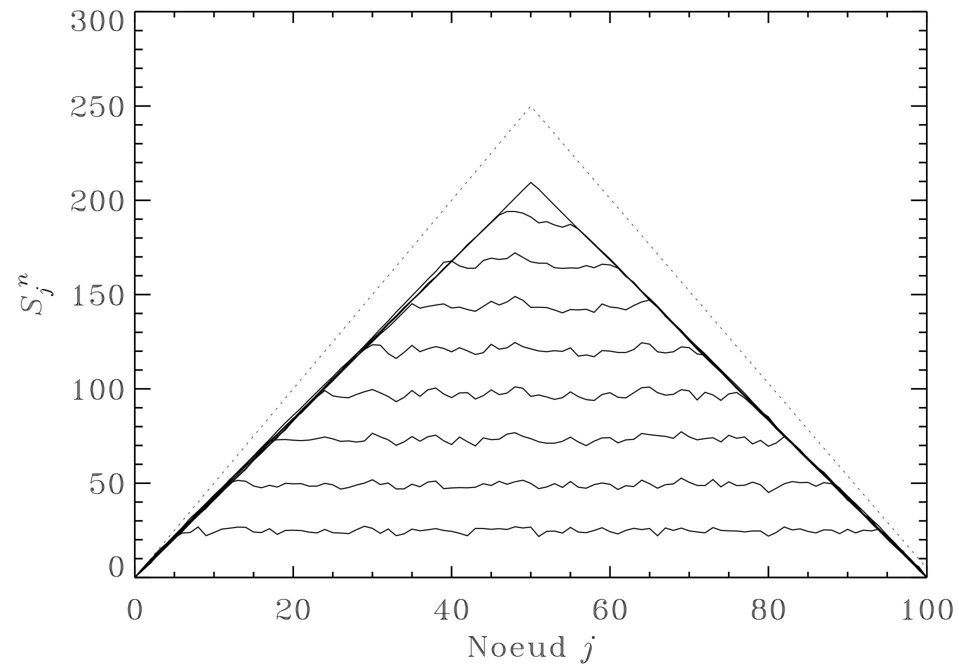


Figure 9.2: Croissance du tas de sable produit par le code C de la Figure 9.1, ici avec $J = 101$, $Z_c = 5$ et $E = 0.1$. Le trait pointillé indique la forme attendue d'un tas parfaitement triangulaire, avec les pentes droites et gauche égales à $\pm Z_c$. Chaque courbe est séparée de la précédente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

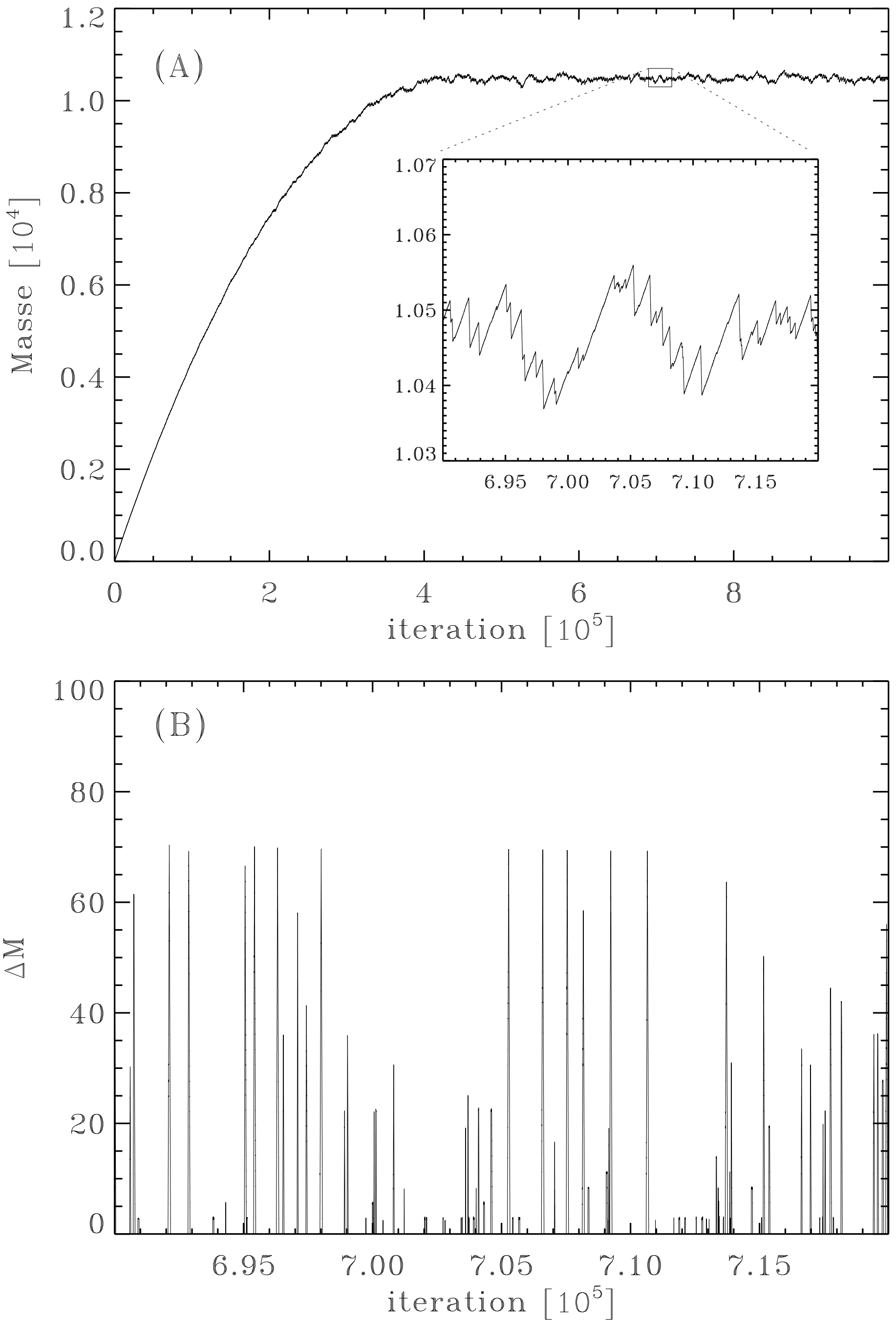


Figure 9.3: Évolution temporelle du modèle TdS en une dimension spatiale, à partir d'une condition initiale $S_j^0 = 0$. Les paramètres de cette simulations sont $J = 101$, $Z_c = 5$, et $E = 0.1$. La partie (A) montre l'évolution de la masse M du tas, l'encadré offrant un zoom sur une petite portion 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.

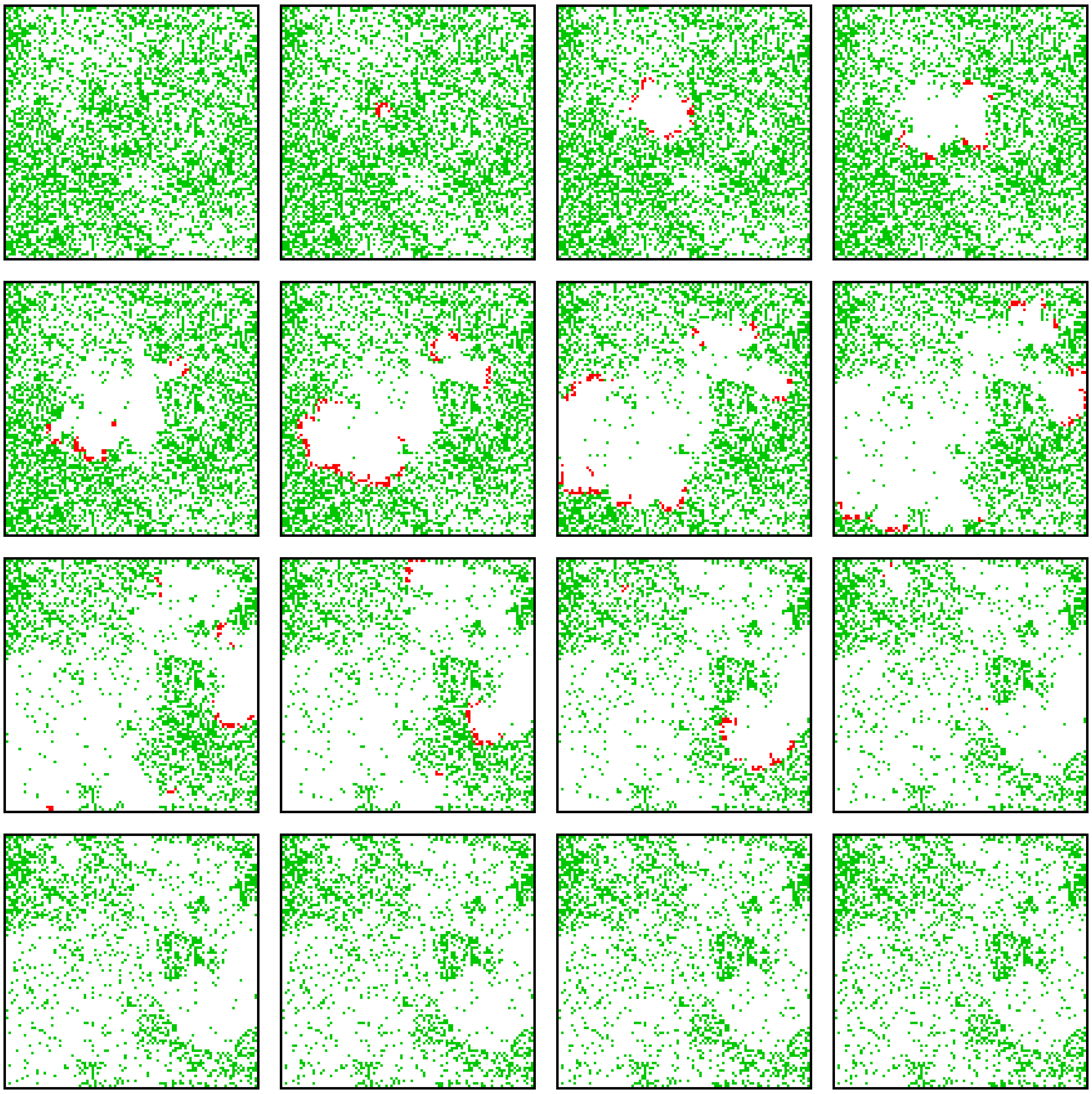


Figure 9.4: Séquence d'instantanés de l'état du réseau pour le modèle FdF avec $p_g = 10^{-3}$ et $p_f = 10^{-5}$, séparés chacun par 10 itérations. Les sites vides sont en blanc, les sites actifs en rouge, et les sites occupés mais inactifs en vert. Au second instantané la foudre vient de frapper un 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.

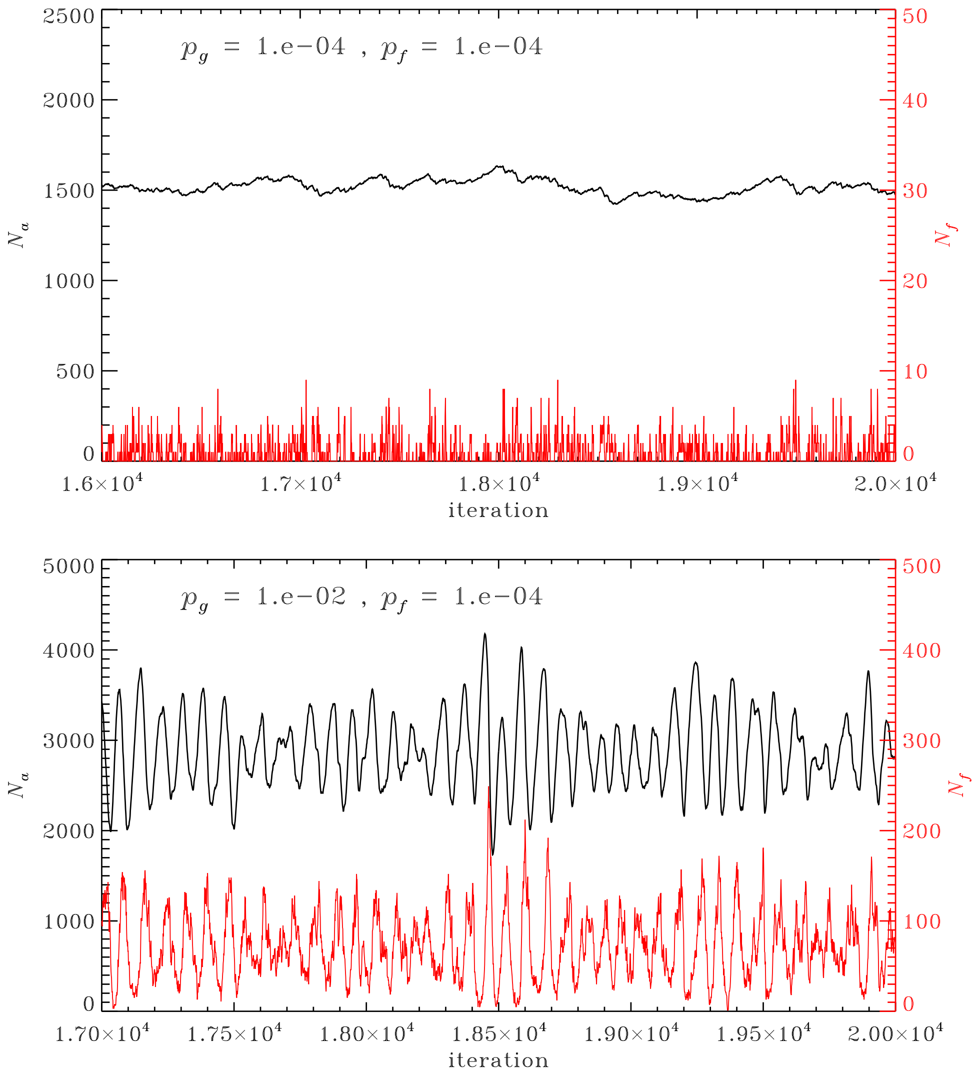


Figure 9.6: Évolution temporelle du nombre de sites occupés inactifs (traits noir) et actifs (traits rouge) dans le modèle FdF avec diverses combinaisons de valeurs de p_g et p_f, dans le régime où la probabilité d'activation est relativement élevée. Toutes les simulations sont effectuées sur un réseau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

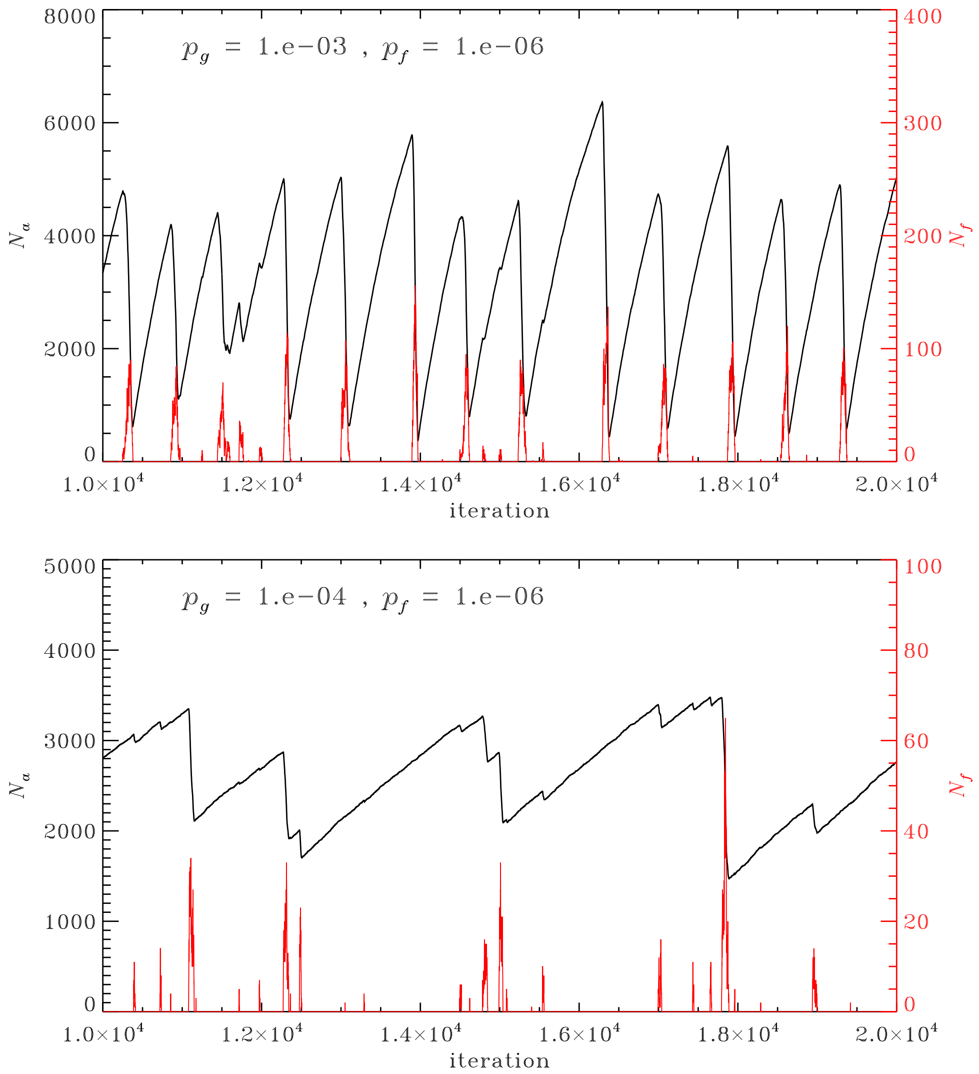


Figure 9.7: Suite de la précédente, maintenant dans le régime où la probabilité d'activation p_f est très faible. Notez bien que les échelles horizontale et verticale des graphiques ne sont pas toujours les mêmes d'un graphique à l'autre.

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

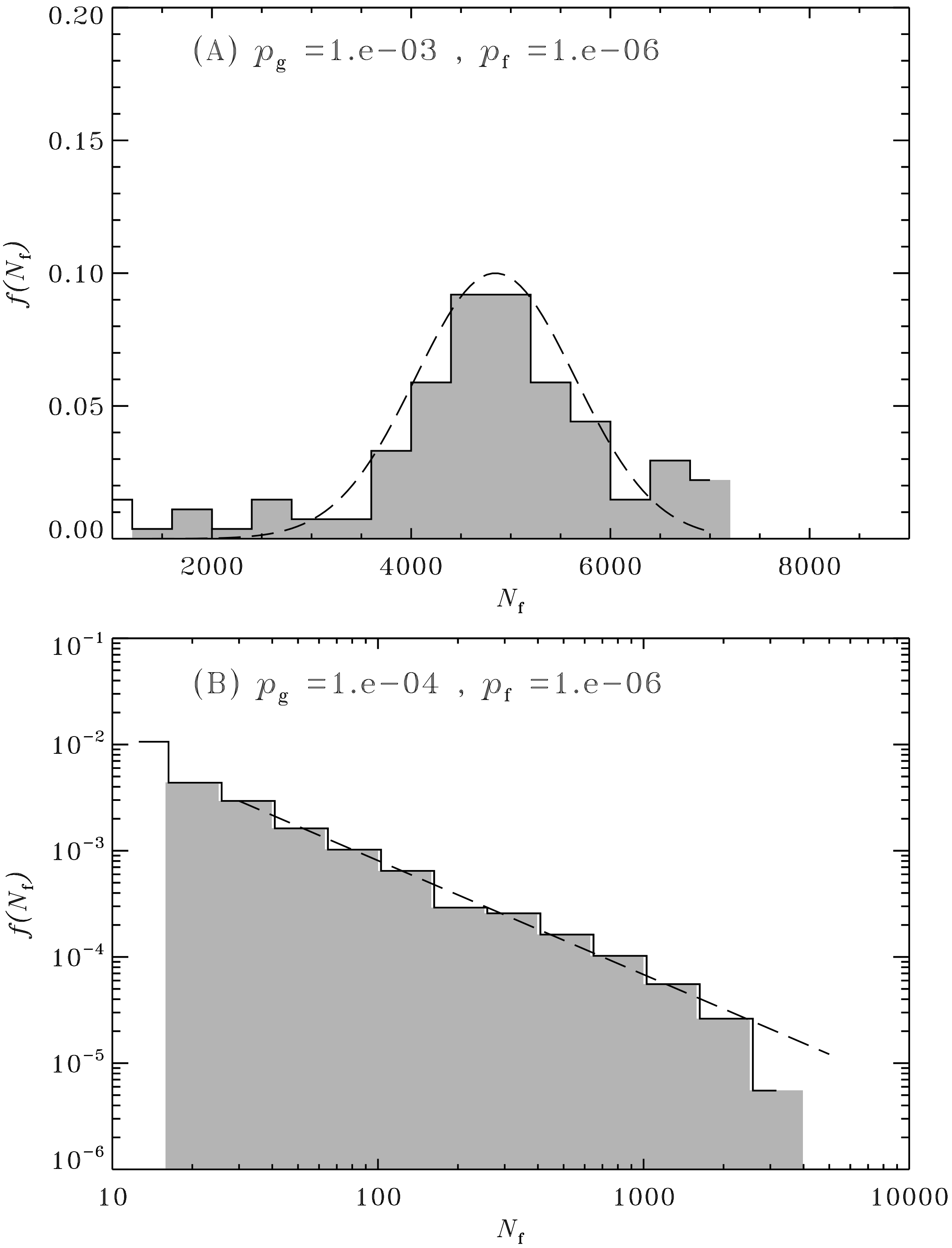


Figure 9.8: Fonctions de densité de probabilité des tailles des feux balayant le réseau, pour les deux simulations de la Figure 9.7. En (A), la distribution est tolérablement bien représentée par une gaussienne, tandis qu'en (B) on observe une loi de puissance, ici avec pente logarithmique −1.07.

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:

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

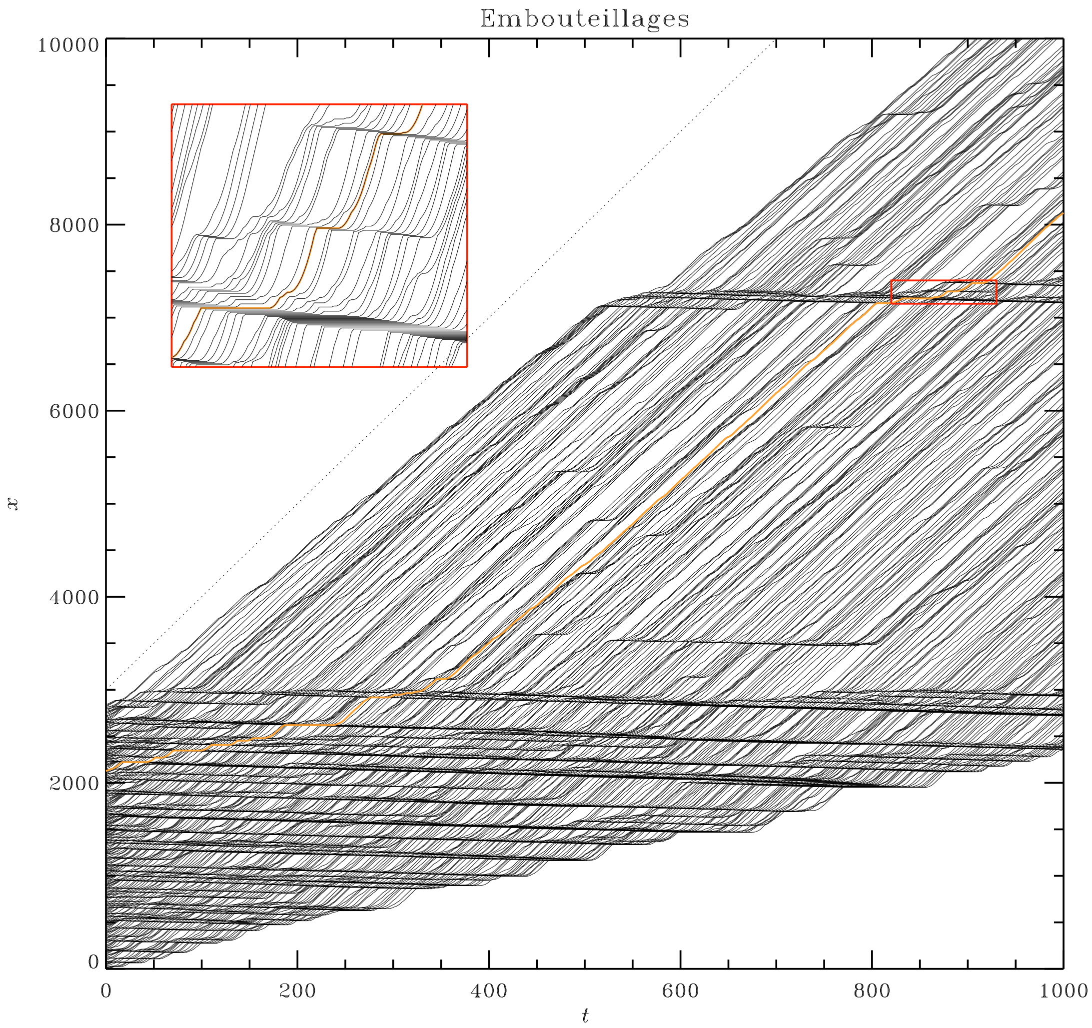


Figure 9.10: Évolution de la position des voitures (axe vertical) en fonction du temps (axe horizontal). On note deux phases distinctes, la première durant laquelle la condition initiale relaxe, par une série d'embouteillages majeurs, vers un état où la vitesse de toutes les voitures est approximativement 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.

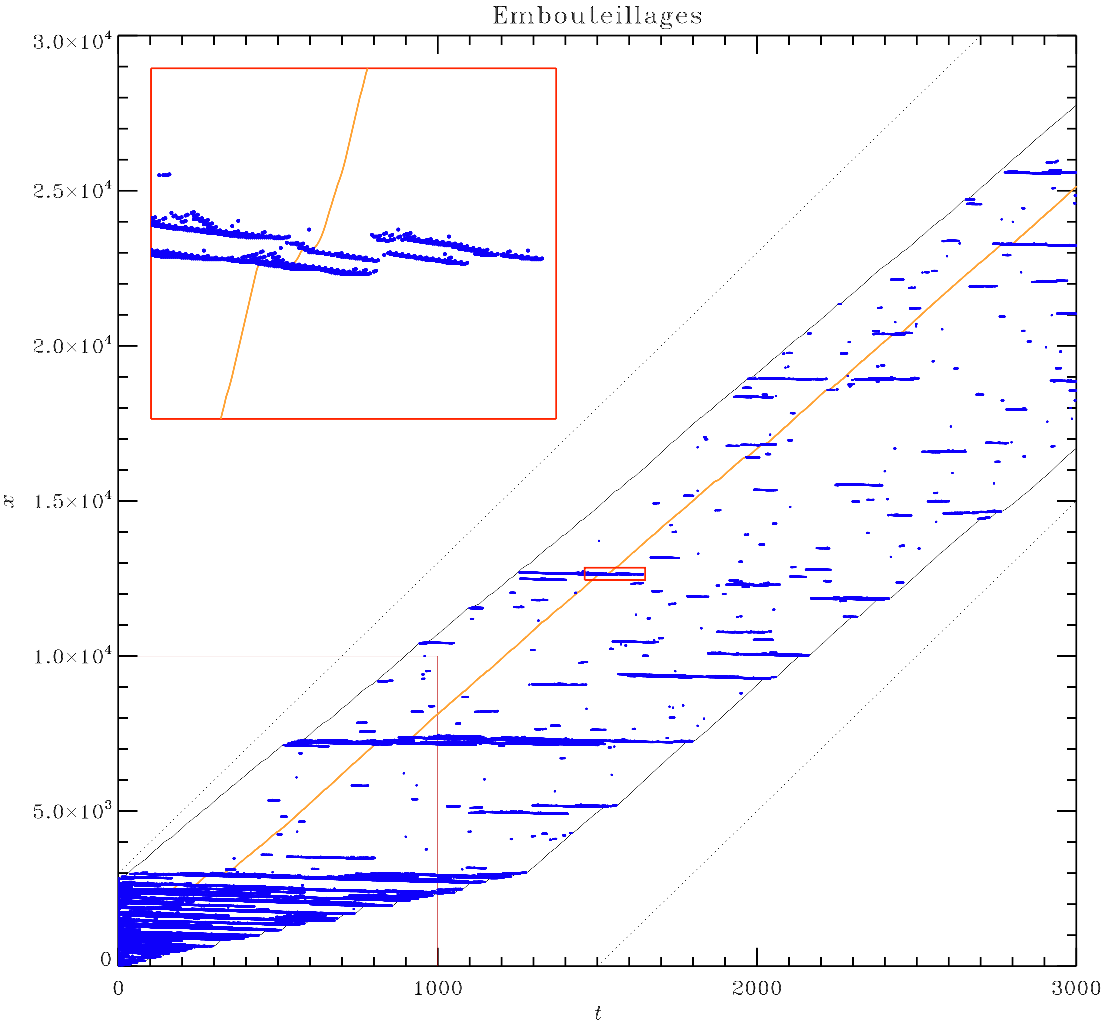


Figure 9.11: La même simulation que sur la Figure 9.10, mais cette fois couvrant un plus grand intervalle temporel et montrant la distribution spatiotemporelle des embouteillages. Chaque point bleu correspond à une voiture au repos ou presque ($v \leq 1$). Le carré inférieur gauche définit l'intervalle 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.

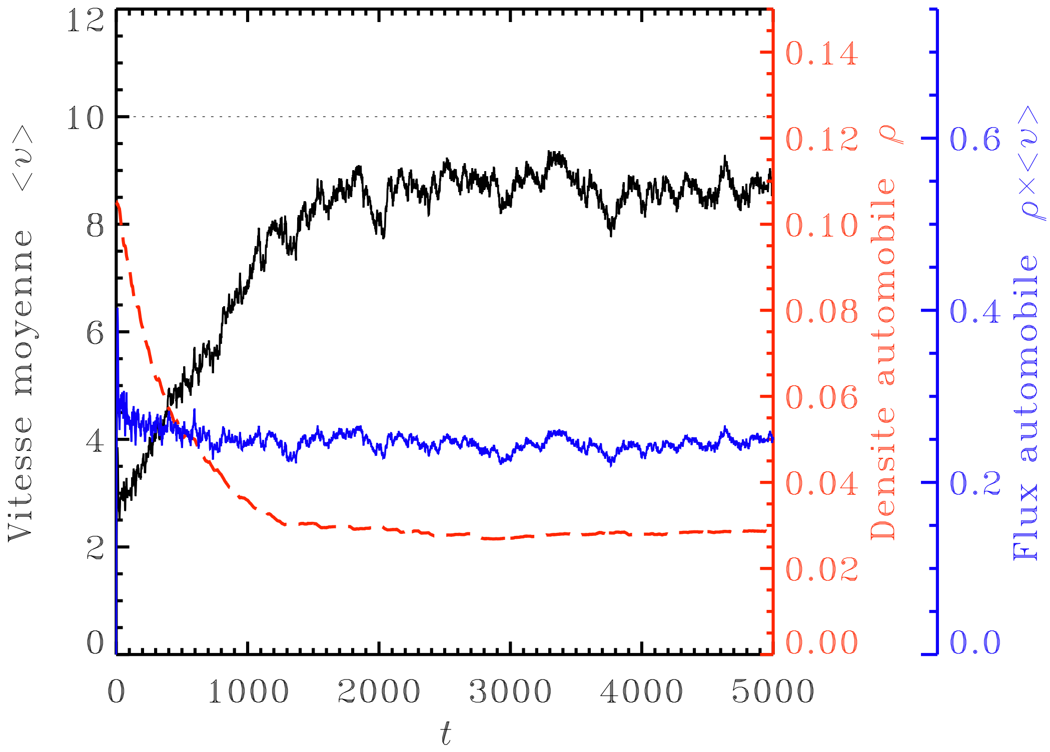


Figure 9.12: Variation temporelle de la vitesse moyenne $\langle v\rangle$, de la densité de voitures ρ, et du flux Φ, pour la simulation des Figures 9.10 et 9.11, maintenant poussée jusqu'à 5000 pas de temps.

(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.

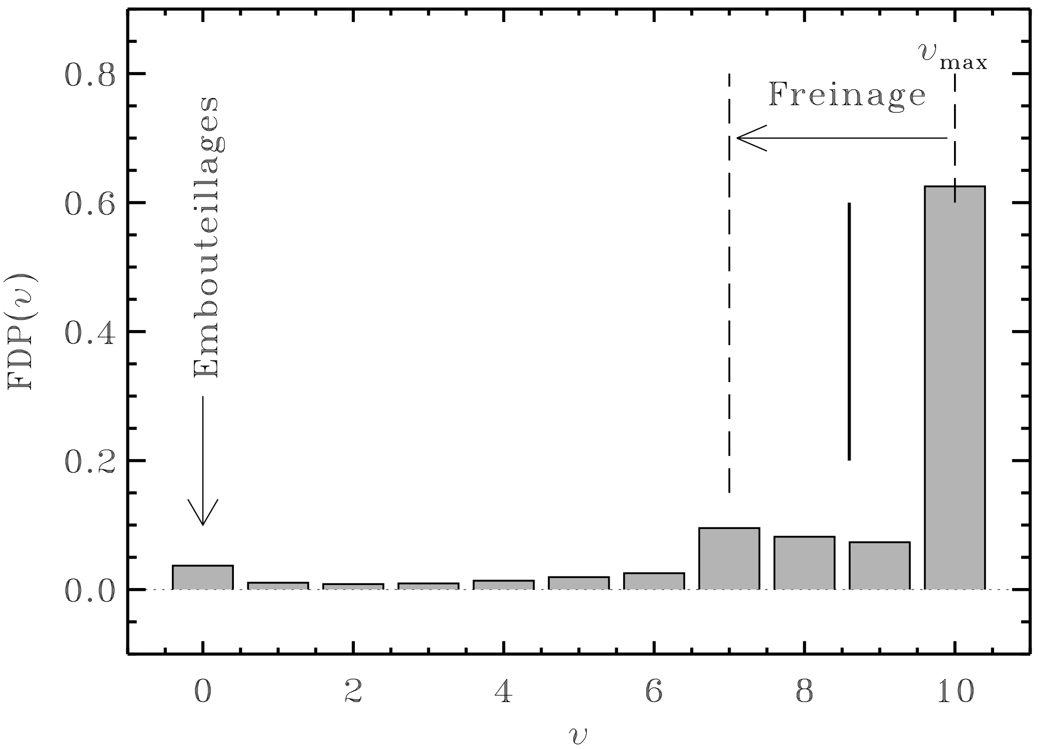


Figure 9.13: Fonction de distribution de probabilité des vitesses, construite à partir des mesures de vitesses de toutes les voitures à chaque pas de temps pour la simulation des Figs. 9.10, 9.11, et 9.12, dans la phase "fluide" $t > 3500$. Le trait vertical noir indique la vitesse moyenne, et le pic 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 ($\to \infty$) 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).

Chapitre 10

Calcul évolutif

10.1 Retour sur la modélisation

Nous allons clore ce cours avec un petit retour sur les problèmes d'optimisation, du genre si souvent produits par l'usage des théories physiques pour l'analyse et l'interprétation des observations ou mesures expérimentales. Les théories sont utlisées pour produire un **modèle** qui se veut une représentation simplifiée et idéalisée de la réalité. On se retouve souvent face au besoin d'ajuster certains paramètres numériques du modèle de manière à minimiser l'écart entre les données observationnelles et leurs équivalents synthétiques calculés à l'aide du modèle. À prime abord ceci définit un problème d'optimisation tout ce qu'il y a de plus classique, mais en pratique plusieurs problèmes peuvent compliquer la chose. En particulier, la relation mathématico-physique entre les paramètres du modèle et les observables rend souvent le problème d'optimisation fortement multimodal, dans le sens que l'espace des paramètres contient plusieurs extrema secondaires. Nous avons déjà rencontré cette situation au chapitre 6, où il avait été mentionné qu'il n'existe aucune approche numérique *garantissant* la détection de l'extremum global.

Dans ce dixième et dernier chapitre, nous allons faire un petit survol de l'approche numérique qui, d'après maintenant 30 ans d'accumulation d'évidence empirique, semble s'approcher le plus d'une méthode d'optimisation qui soit à la fois globale et robuste, dans le sens de bien fonctionner sur une vaste gamme de problèmes: le calcul évolutif. Le concept central au calcul évolutif, inspiré directement de l'évolution en biologie, est la sélection, parmi un ensemble de solutions-test, des solutions les plus performantes comme point de départ à la production de la prochaine "génération" de solutions-test.

10.2 La puissance de la sélection cumulative

L'idée qu'un processus de sélection puisse accélérer une recherche du genre grimpe stochastique semble évidente, mais ce qui l'est pas mal moins et l'impact d'un tel processus de sélection sur une succession de générations de solutions-test. Un exemple très simple illustre ceci à merveille. Prenons la petite phrase suivante, tirée d'un ouvrage bien connu d'un auteur qui vaut la peine d'être lu:

```
D E S S I N E   M O I   U N   M O U T O N
```

Cette phrase compte 21 caractères, tirés d'un alphabet de 27 lettres (on compte les espaces blancs comme un caractère et un membre de l'alphabet). Installons maintenant le proverbial singe face à un clavier d'ordi, avec une bonne réserve de bananes, et la mission de reproduire cette phrase cible. L'idée est de laisser notre singe taper aléatoirement des séquences de 21 caractères, jusqu'à temps qu'il tombe sur la bonne. Voici un exemple de 10 de ces essais, avec le nombre de lettres correctes indiqué dans la colonne de droite:

```
H C W O X O I Q B E D D D P L I E G U V E      0
I U   X Y U M O A D U C O T F   V W A G E      0
L F C C P I   K A Q Q A H Z J R S V R R T      0
C X Q R Z F Q J Z O B C B N K Y L D S R U      2
S C N C     A I G F S F G K W U K T K O U      1
X D D W G O O X X R P E I F K D B W K F Q      0
Z O Q N A A V M O   L H F   P P B R E B A      0
Y   P F V W Y I U X Q J H P C C A R Y N D      0
A O A R   G G E   V B E N G J B H Q Z K G      0
Y J Y E O Q F E L Y A N L R J A Y V V Q F      0
```

Ca ne ressemble absolument pas à la phrase cible, quoiqu'en y regardant bien on constate que le quatrième essai a en fait reproduit deux des lettres au bon endroit dans la séquence, soit le "O" du "MOI" et le "N" du "UN". Considérant la longueur de la séquence et la taille de l'alphabet, produire une phrase avec deux lettres correctes en 10 essais est tout à fait probable (vous aurez à le vérifier dans un des problèmes en fin de chapitre!). La probabilité p de reproduire correctement les 21 lettres est donnée par

$$p = \left(\frac{1}{27}\right)^{21} = 8.737 \times 10^{-31} \ , \tag{10.1}$$

soit une chance sur $\sim 10^{30}$. En physique, évidemment, on n'a pas peur des gros chiffres; 10^{30}, après tout, c'est seulement la moitié de la masse du soleil mesurée en kilogrammes... Mais il faut tout de même apprécier que 10^{30}, c'est vraiment gros. Si je vous donne 10^{30} grains de sable de qualité-plage (diamètre 0.2mm) pour en faire un tas, votre tas recouvrera la Terre entière d'une épaisseur de 8 kilomètres. Et votre tâche est de trouver, dans ce tas, un seul et unique grain très spécifique. Je n'ai pas fait le calcul, mais je soupçonne fort que même avec une armée de 5 milliard de singes entrainés à examiner un grain de sable à la seconde, la quantité de bananes requises pour trouver ce fameux grain St-Exupériesque est beaucoup, beaucoup plus grande que la masse de l'Univers, matière sombre incluse. Bref, bonne chance...

Considérons maintenant la variation suivante; de nos 10 essais aléatoires, on choisi le meilleur (on devrait peut-être dire plutôt le moins pire, à ce stade), soit ici le quatrième; on en fait 10 copies conformes; puis, dans chaque copie, on introduit des "mutations", de la manière suivante: chaque lettre se fait remplacer par une autre lettre choisie aléatoirement dans l'alphabet, avec une probabilité p_m. Si cette probabilité $p_m \ll 1$, alors on peut imaginer que seule une ou deux lettres seront remplacées dans chaque phrase. Une fois ce processus complété pour les 10 phrases recopiés, on rechoisit la meilleure dans cette nouvelle "population", et on recommence le cycle de copie/mutation. Vous percevez j'espère l'analogie avec l'évolution en biologie; le choix du meilleur, c'est la sélection naturelle (en version assez extrême); la copie en 10 nouvelles phrases, c'est la reproduction (en version asexuée); les mutations, c'est la variabilité génétique produite par les erreurs de copies de l'ADN.

On peut facilement imaginer que ce processus accélèrera la reproduction de notre phase cible, mais je doute fort que vous réalisiez jusqu'à quel point cette accélération est substantielle. La Figure 10.1 montre la variation de l'erreur (mesurée ici comme le nombre de lettres incorrectes) en fonction du nombre de "générations" de 10 phrases produites. La phrase cible est atteinte ici en 835 itérations, durant lesquelles 8350 phrases ont été "évaluées". On est *très très* loin du 10^{30} de la recherche aléatoire!

Une minute de réflexion devrait suffire pour comprendre que la mutation est ce qui nous fait progresser rapidement ici; donc il devrait être possible de faire mieux que 835 itérations en augmentant p_m, dont la valeur était de 10^{-2} pour la "solution" de la Figure 10.1. La Figure 10.2 montre trois convergences vers la phrase-cible, pour $p_m = 10^{-3}$, 10^{-2} et 10^{-1}. À $p_m = 10^{-3}$, la convergence est plus lente qu'à 10^{-2}, comme on s'y attendait; mais à $p_m = 0.1$, la convergence, bien que très rapide au début, sature rapidement à un niveau d'erreur substantiel. Le problème est ici que la mutation ne fait pas seulement remplacer les mauvaises lettres par des bonnes, elle fait aussi l'inverse, et la convergence sature lorsque la probablilité totale de détruire une

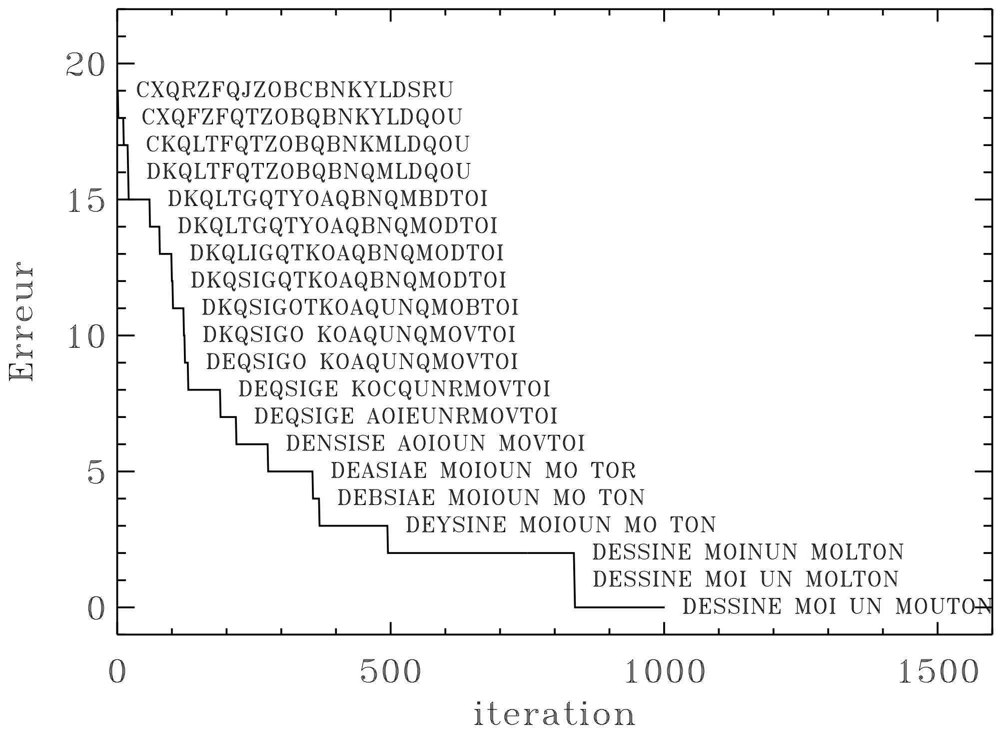


Figure 10.1: L'évolution du Petit Prince par sélection artificielle.

"bonne" lettre devient égale à celle de "produire" une bonne lettre. Un des exercices en fin de chapitre vous guide à travers le calcul de cette probabilité. Donc, la mutation c'est bien, mais seulement à petite dose. Si vous avez encore besoin d'en être convaincu, allez chercher sur le Web des photos des enfants de Tchernobyl...

Évidemment, comme représentation du processus d'évolution biologique cet exemple n'est pas très réaliste: (1) on ne choisit ici que la meilleure de toutes les solutions-test, ce qui représente une forme de sélection plutôt extrême; (2) la "reproduction" produit ici des copies quasi-conformes du parent; (3) la qualité des solutions est mesurée par rapport à un absolu connu *a priori*, ce qui n'est pas du tout le cas en biologie (n'en déplaise à Teilhard de Chardin). Il n'en demeure pas moins qu'il est possible de bâtir des méthodes d'optimisation globale qui incorporent ces idées dans un contexte purement numérique. Ses éléments-clef sont: (1) un processus de sélection des solutions, basé sur une mesure de leur "qualité"; (2) la reproduction des solutions qui sont jugées les meilleures, d'une manière qui préserve au moins en partie leurs caractéristiques avantageuses (**héritabilité**); et (3) l'injection de **variabilité** dans la population, habituellement via le processus de reproduction, afin de pouvoir explorer les coins de l'espace des paramètres qui ne sont pas "échantillonnés" par la population initiale. Par exemple, dans l'exemple littéraire ci-dessus, aucune des phrases-tests n'avait de "D" en première position; aucun mécanisme de reproduction basé uniquement sur un échange de lettres entre solutions n'aurait pu, en soi, conduire à la phrase-cible.

10.3 Un algorithme évolutif de base

Voyons maintenant comment appliquer cette idée à un problème d'optimisation du genre de ceux considérés précédemment au chapitre 6, et plus précisément notre désormais familier problème de recherche du maximum global du patron de diffraction d'une ouverture carrée.

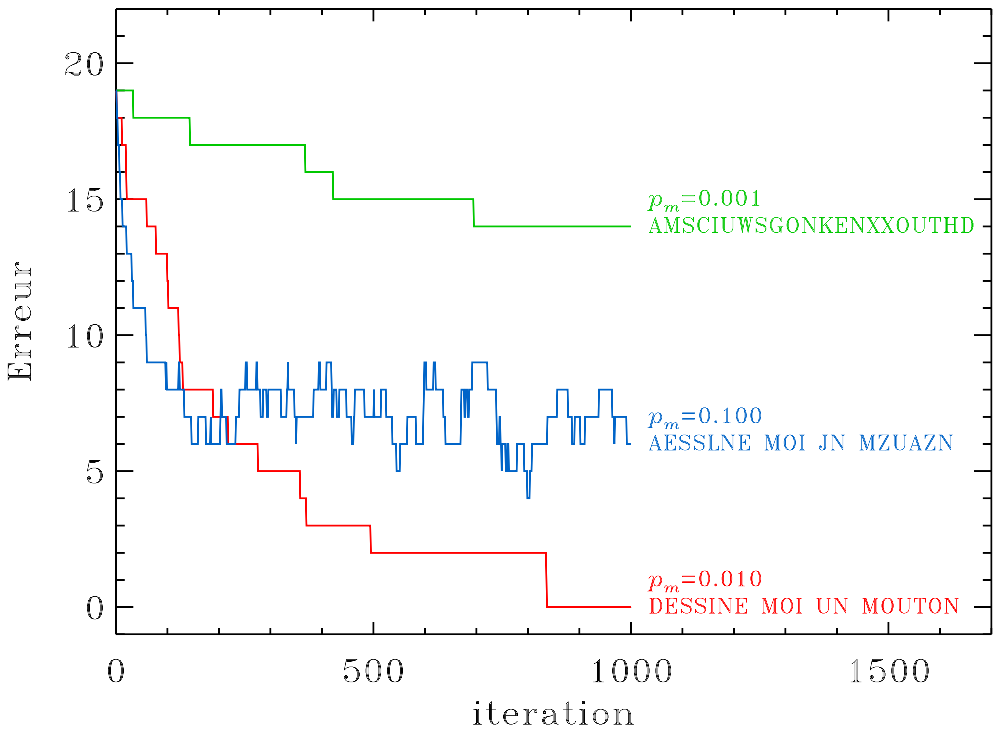


Figure 10.2: Impact de la probabilité de mutation sur l'évolution linguistique. Si la mutation est trop élevée, la convergence sature (voir texte).

Nous considérons dans ce qui suit un algorithme évolutif très simple. Les choix spécifiques faits ci-dessous au niveau de la triade des processus de sélection/héritabilité/variabilité ne sont pas uniques, et certainement pas optimaux, mais ils conduisent à un algorithme relativement facile à coder et dont la performance est tout à fait acceptable.

10.3.1 L'algorithme

Le calcul évolutif procède de manière itérative et agit sur une population de N solutions-test au problème. Dans sa forme de base discutée ici, il ne requiert que la capacité d'assigner à chaque solution-test une mesure de qualité permettant de distinguer les meilleures solutions-test des pires. On peut voir cette mesure de qualité comme une fonction des paramètres du problème, et vu sous cet angle le calcul évolutif n'est qu'une autre méthode d'optimisation de cette mesure de qualité, quelle qu"elle soit. Un algorithme de base a l'air de ceci:

1. **Initialisation:** on produit une population de N solutions (ici, des points (x, y)) complètement aléatoires;
2. **Évaluation:** On évalue la qualité des membes de la population;
3. **Classement:** On classe les solutions en ordre croissant de qualité;
4. **Test:** Si la meilleure solution satisfait au critère de convergence, on arrête ici; sinon on continue;
5. **Sélection**: On choisit deux bonnes solutions, en fonction de leur rang;
6. **Reproduction:** Les deux solutions choisies se reproduisent en deux nouvelles solutions;

7. **Itération:** Les étapes 5 et 6 sont répétées jusqu'à ce que l'on ait produit un nouvel ensemble de N solutions,

8. **Remplacement:** On remplace l'ancienne population par la nouvelle.

9. **Prochaine génération:** Retour à l'étape 2.

Examinons maintenant le détail des différences étapes de cet algorithme.

10.3.2 Initialisation

L'évolution débute habituellement avec une population aléatoire; par exemple, pour le problème de recherche du maximum de diffraction en 2D, chaque membre de la population est un point (x, y), dont les valeurs devraient être extraites d'une distribution aléatoire uniforme dans l'intervalle considéré, par exemple $[-3\pi, 3\pi]$ comme sur la Figure 6.14. Cet échantillonnage initial est important, car il sera la source du "pool génétique" sur laquelle travaillera le processus évolutif. On veut que ce pool soit le plus varié possible.

Dans le cas de notre problème de diffraction 2D, l'initialisation de la population se ferait en C comme suit:

```
range=3.*PI
for (i=0 ; i<N ; i++) {
   x[i]=range*( -1.+2.*rand()/RAND_MAX ) ;
   y[i]=range*( -1.+2.*rand()/RAND_MAX ) ;
}
```

où la fonction `rand()` est le générateur générique du langage C, et la variable `range` contrôle ici l'étendue de la région de l'espace dees paramètres à explorer. Ce bout de code présuppose que les tableaux `x[N]` et `y[N]` et la variable `PI` ont définis de manière appropriée.

10.3.3 Évaluation

Il faut maintenant assigner une mesure de qualité ("fitness") à chaque membre de la population. Dans le cas de la recherche de maximum du patron de diffraction 2D, ce pourrait être simplement la valeur absolue (ou le carré) de l'évaluation de l'éq. (6.15); Une fonction C effectuant cette évaluation est déjà incluse dans le code C illustrant la méthode de grimpe (§6.5), et ne sera donc pas reproduite ici. En général, c'est à vous de définir une mesure quantitative de la qualité qui soit appropriée au problème.

10.3.4 Classement

Dans le processus de sélection qui sera décrit plus bas, il s'avèrera pratique d'avoir accès aux N solutions formant la population classées en fonction de leur mesure de qualité. Le classement est un sujet classique couvert dans tous les cours de programmation. Vous pourrez trouver dans le *Numerical Recipes* quelques codes C qui, à partir d'un tableau `X[N]` de longueur N contenant les valeurs numériques de la mesure de qualité, produisent en sortie un vecteur d'entiers `irank` dont chaque élément contient le rang de l'élément correspondant dans `X`, le rang le plus élevé correspondant à l'élément ayant la valeur la plus élevée; autrement dit, l'élément `irank[N]` contient le meilleur de la population, `irank[N-1]` le second, et ainsi de suite jusqu'à `irank[1]` pour le pire.

10.3.5 Sélection

Il existe plusieurs mécanismes de sélection possibles. Ce n'est en général *pas* une bonne idée de choisir seulement le meilleur membre de la population pour produire la génération suivante, car celà peut souvent conduire à une convergence prématurée sur un extremum secondaire. Il est essentiel de préserver un bon niveau de variabilité dans la population, donc les solutions qui ne sont pas les meilleures ne doivent pas être éliminées trop rapidement du “pool génétique”.

Voici une possibilité simple à coder et qui fonctionne bien: les “parents” sont choisis par paires (la reproduction étant définitivement plus agréable à deux que seul...), le premier aléatoirement parmi les $N/5$ meilleurs de la population, le second aléatoirement parmi la population en entier. Le facteur $1/5$ est ici un choix empirique, et détermine la **pression sélective**; une valeur numérique plus grande (e.g., $1/2$) donnerait moins d'influence aux meilleurs de la population, tandis que $1/N$ ferait que seul le meilleur serait choisi à tous les coups comme premier parent.

```
i1=irank(N-1-floor(1.*(N/5)*rand()/RAND_MAX)) ;  /* parent parmi les premiers 20% */
i2=i1 ;
while { i2==i1 } do { i2= floor((1.*(N-1)*rand()/RAND_MAX) ; } /* n'importe qui  */
```

Ici les entiers `i1` et `i2` identifient, dans les tableaux `x[N]` et `y[N]`, les position des paramètres x et y des deux parents choisis pour la reproduction. Notez bien que la fonction générique `floor` du C tronque le nombre réel donné en argument, donc pour que `i2` se retrouve entre 0 et $N-1$ on doit bien multiplier `rand()` par N, et non $N-1$. La boucle `while` est requise pour empêcher que les deux parents correspondent au même individu; pour une population de petite taille, de tels croisements “consanguins” pourraient rapidement conduire à une convergence prématurée sur un extremum suboptimal.

10.3.6 Reproduction

La reproduction est un processus en deux étapes: le **croisement**, qui consiste à partager le “matériel génétique” des deux parents entre les deux “enfants”; et la **mutation**, qui consiste à introduire une perturbation au matériel génétique des enfants. Ces opérations ont lieu avec des probabilités p_c et p_m, respectivement. L'expérience montre que $0.5 \leq p_c \leq 1.0$ fonctionne bien (on choisira $p_c = 0.8$ dans ce qui suit), mais il est définitivement préférable d'avoir $p_m \ll 1$. En C, on peut formuler ce genre de test probabiliste avec l'instruction:

```
if ( 1.*rand()/RAND_MAX < pcross ) { ... }
```

où, comme auparavant, `1.*rand()/RAND_MAX` produit un nombre aléatoire extrait d'une distribution uniforme dans l'intervalle $[0, 1]$.

Toujours dans le cas de notre problème de diffraction 2D, le matériel génétique est simplement les valeurs de x et y définissant chaque membre de la population. Donc, soit les points (x_1^n, y_1^n), (x_2^n, y_2^n) définissant les deux parents de la génération n choisis à l'étape “sélection”, et deux nombres aléatoires r_1, r_2 extraits d'une distribution uniforme dans l'intervalle $[0, 1]$. Le processus de croisement produit deux rejetons à partir des deux parents, selon les expressions suivantes

$$x_1^{n+1} = r_1 x_1^n + (1 - r_1)\, x_2^n \ , \qquad x_2^{n+1} = (1 - r_1)\, x_1^n + r_1 x_2^n \ , \tag{10.2}$$

$$y_1^{n+1} = r_2 y_1^n + (1 - r_2)\, y_2^n \ , \qquad y_2^{n+1} = (1 - r_2)\, y_1^n + r_2 y_2^n \ . \tag{10.3}$$

Notez que cette forme de moyenne pondérée aléatoirement entre les x et y des deux parents a comme effet que les rejetons se retrouveront automatiquement dans les intervalles:

$$\min(x_1^n, x_n^2) \leq x_1^{n+1}, x_2^{n+1} \leq \max(x_1^n, x_2^n) \ , \tag{10.4}$$

$$\min(y_1^n, y_n^2) \leq y_1^{n+1}, y_2^{n+1} \leq \max(y_1^n, y_2^n) \ . \tag{10.5}$$

Pour chacun des paramètres, le croisement n'opère qu'avec une probabilité p_c; si la procédure de test refuse le croisement, alors les deux deux rejetons héritent chacun de l'une des valeurs de paramètre des parents; par exemple, si le croisement est refusé pour la variable x, on aurait alors:

$$x_1^{n+1} = x_1^n \ , \qquad x_2^{n+1} = x_2^n \ , \tag{10.6}$$

La mutation consiste à ajouter une "perturbation" aux paramètres x ou y de chacun des rejetons, mais si seulement un test accepte la mutation (un test par paramètre par rejeton). On aurait par exemple:

$$x_1^{n+1} = x_1^{n+1} + g(\sigma), \tag{10.7}$$

où $g(\sigma)$ est un nombre aléatoire extrait d'une distribution gaussienne de variance σ (cf. §6.6).

10.3.7 Remplacement

Une fois les rejetons produits par le processus décrit ci-dessus, on doit les insérer dans la population. La stratégie dite de **remplacement générationnel complet** consiste à accumuler les rejetons dans un tableau temporaire, et une fois qu'on en a produit N ils remplacent la population-parent à la fin de l'itération générationnelle. Une stratégie additionnelle très utile, l'**élitisme**, consiste à recopier intact le meilleur individu de la population-parent dans la population-rejeton, afin de s'assurer que les aléas du processus reproductif n'en viennent pas à faire reculer le processus évolutif.

10.3.8 Test de convergence

Comme dans toute méthode d'optimisation globale, il n'y a pas de choix unique et universel pour le critère d'arrêt. Typiquement, on itère soit pour un nombre de générations fixé *a priori* (boucle inconditionnelle), ou tant que la meilleure solution ne change plus de manière significative pendant un nombre prédéterminé d'itération (genre 100. Ou mille; ou plus...?).

10.4 La diffraction 2D, troisième tour...

Appliquons maintenant le calcul évolutif au problème de la recherche du maximum central du patron de diffraction 2D. La Figure 10.3 illustre la distribution des membres de la population en fonction du temps, résultant de l'application de l'algorithme évolutif décrit ci-dessus. Remarquez que la population initiale n'inclut aucun membre ayant "atterri" sur les flancs du pic central, ce qui implique que même avec ces 20 parachutistes, la grimpe classique aurait échoué ici. Cependant, dès la seconde génération un croisement a produit un individu qui s'est retrouvé au flanc du pic central, et l'effet combiné de la sélection, des croisements et des mutations durant les 3–4 générations subséquentes fait converger l'ensemble de la population sur le pic central, quelque peu hors-centre (génération 6). Comme les x et y des différents individus sont maintenant presqu'identiques, le croisement n'accomplit plus grand chose, et c'est l'effet cumulatif des mutations, couplé à la sélection avec élitisme, qui déplace lentement la population vers l'extremum global à $(0,0)$.

La Figure 10.4 montre la convergence de l'algorithme évolutif, pour différentes probabilités de mutation p_m. L'erreur ε à une génération est ici mesurée en fonction du meilleur individu (x^*, y^*) de la population, selon

$$\varepsilon = |1 - I(x^*, y^*)| \tag{10.8}$$

On retrouve l'effet déjà noté dans le dessin du mouton: la mutation est requise, mais doit demeurer suffisamment basse sinon la convergence est freinée. À $p_m = 0.1$ on converge à une précision de 10^{-6} en une quarantaine de générations, pour un total de près de 800 évaluations

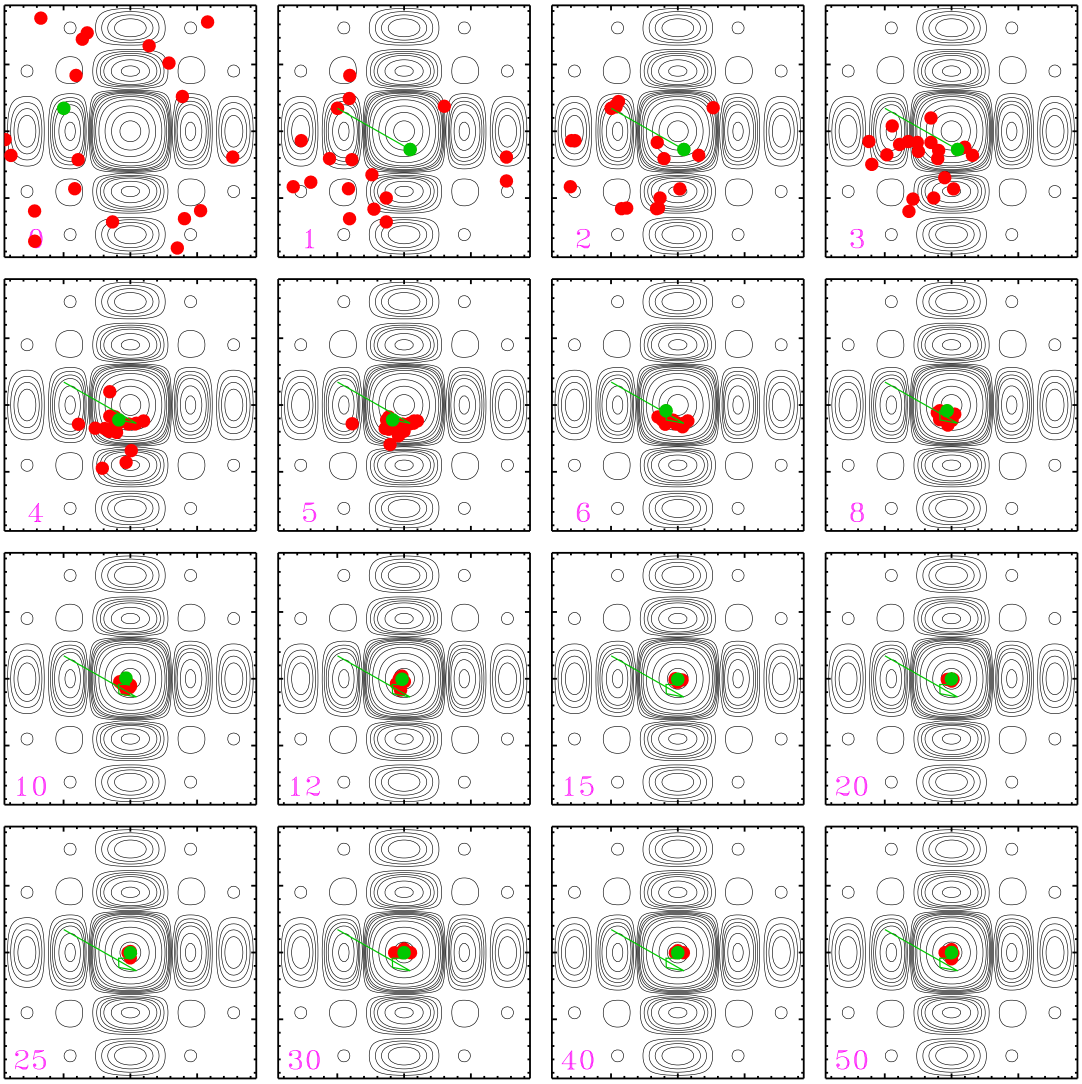


Figure 10.3: Recherche du maximum central du patron de diffraction 2D par calcul évolutif. L'espace de recherche couvre ici l'intervalle $[-3\pi, 3\pi]$ en x et y. Le point coloré en vert correspond au meilleur individu de sa génération (numérotée dans le coin inférieur gauche de chaque image). Notez comment aucun des membres de la population initiale aléatoire n'est tombé suffisamment près du pic central pour qu'une grimpe classique ne conduise à l'extremum global à $(x, y) = (0, 0)$.

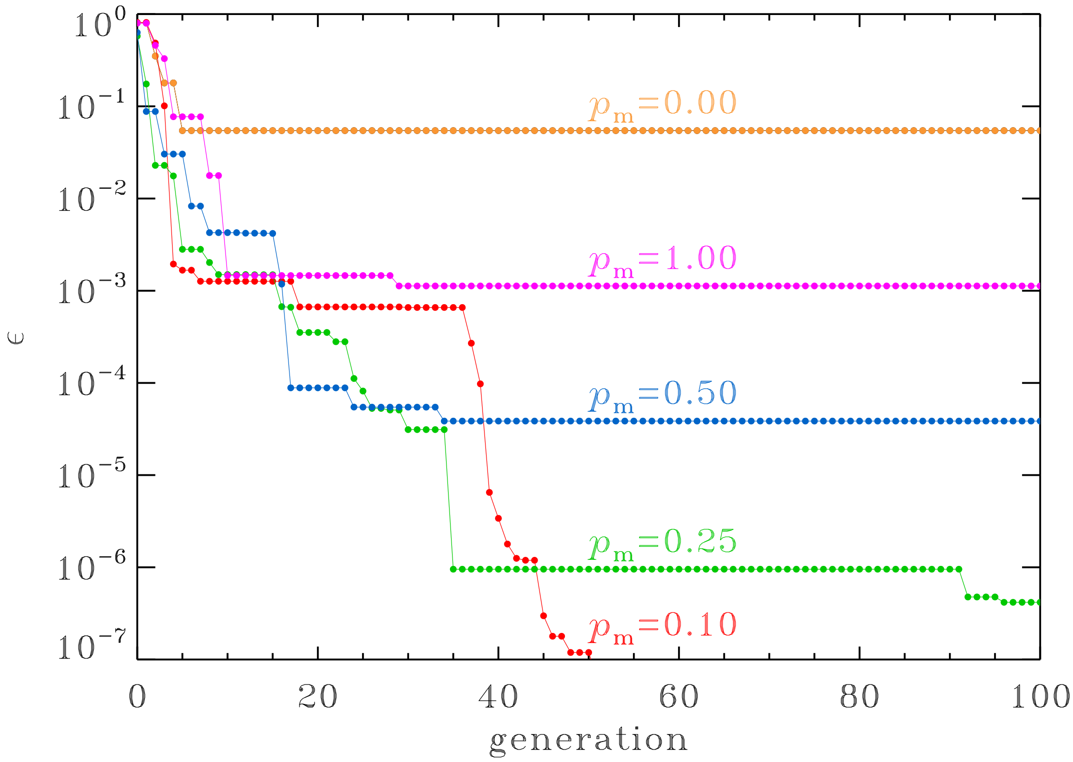


Figure 10.4: Convergence de l'algorithme évolutif de base pour différentes probabilités de mutation p_m, tel qu'indiqué, toujours pour le problème de diffraction 2D.

de la fonction $I(x, y)$, ce qui est comparable à la grimpe stochastique de la Figure 6.14. Mais ici, le pic central est localisé à tous les coups! Et comme vous pourrez le vérifier dans un des exercices en fin de chapitre, la performance globale de l'approche évolutive se dégrade beaucoup plus lentement que celle de toutes les autres techniques d'optimisation décrites précédemment à mesure que la dimensionalité de l'espace de recherche augmente.

Vous imaginerez sans difficultés que le comportement évolutif de l'algorithme peut être fortement influencé non seulement par les valeurs choisies pour p_c et p_m, mais aussi par d'autres facteurs comme la taille de la population, la pression de sélection, le choix des distributions statistiques contrôlant la taille des mutations, etc. Les choix ci-dessus sont bons, mais certainement pas optimaux, dans le sens que d'autres valeurs des divers paramètres évolutifs pourraient bien accélérer la convergence *pour ce problème de diffraction 2D*. Cependant, les meilleures valeurs de paramètres évolutifs pour un autre problème d'optimisation seraient fort probablement différentes. En général, il est préférable de s'en tenir à des valeurs qui, sans être vraiment optimales pour aucun problème spécifique, donnent des résultats acceptables pour une vaste gamme de problèmes très différents. Une méthode d'optimisation ayant cette propriété est dite **robuste**.

Notons finalement qu'en posant $p_c = 0$ et $p_m = 1$, on se retrouve avec un algorithme qui commence fort à ressembler à une grimpe stochastique opérant en mode parallèle.

10.5 Les algorithmes génétiques

Les **algorithmes génétiques** sont une classe d'algorithmes évolutifs qui poussent encore plus loin l'analogie biologique, cette fois au niveau de l'encodage des paramètres du modèle et des opérateurs de reproduction. Plutôt que d'effectuer la reproduction (§10.3.6) par moyenne stochastique (eq. (10.2)) et perturbation ((eq. 10.7)), les paramètres définissant chaque "individu" sont encodés en une chaine de bits (le "chromosome"), et le croisement entre deux individus est effectué en coupant les deux chaines à un bit choisi aléatoirement, et en interchangeant les fragments de chaines situés plus loin que le point de coupe. Les deux chaines résultantes, qui incorporent maintenant des morceaux *intacts* des chromosomes parentaux. Le processus de mutation consiste maintenant à inverser ($0 \to 1$ ou $1 \to 0$) un ou plusieurs bits choisis aléatoirement sur chaque chaine. Ces chaines sont finalement décodées pour produire les valeurs numériques des paramètres définissant l'instance du modèle associée à chacun des deux rejetons.

L'avantage de procéder ainsi vient du fait que des blocs de bits contigus qui confèrent à leur porteurs une mesure de qualité supérieure à la moyenne voient leur fréquence augmenter geométriquement dans le pool génétique. Ceci est quantifié et formalisé par le **théorème fondamental** des algorithmes génétiques, dû à John Holland, un des doyens-fondateurs de ces méthodes. Voir la bibliographie en fin de chapitre si vous êtes intéressé(e)s à en apprendre plus là-dessus.

Bibliographie:

L'exemple de l'évolution de la phrase littéraire de la §10.2 est empruntée de l'excellent ouvrage "grand public":

Dawkins, R., *The Blind Watchmaker*, W.W. Norton (1986),

mais j'ai changé la phrase-cible, celle de Dawkins étant tirée de Shakespeare, comme on s'y attendrait d'un Brit d'Oxford.

Les algorithmes évolutifs sont un sujet sur lequel j'ai déjà pas mal travaillé dans le passé, et sont en fait une des rares techniques d'optimisation d'intérêt qui ne soit pas discutée dans *Numerical Recipes* auquel j'ai si souvent fait référence dans ces notes de cours. Si vous voulez en savoir plus là-dessus, j'ai écrit il y a quelques années une petite introduction au sujet dont je ne suis pas mécontent:

Charbonneau, P., *An introduction to genetic algorithms for numerical optimization*, NCAR Technical Note 450+IA, National Center for Atmospheric Research (2002)

Ca peut se commander du NCAR via le Web... ou vous pouvez passer à mon bureau ramasser une des copies que j'ai déménagé à Montréal... Le grand classique théorique dans le domaine, et solidement technique, demeure

Holland, J.H., *Adaptation in natural and artificial systems*, 2^e éd., MIT Press (1992).