Université de Montréal

MÉTHODES THERMODYNAMIQUES APPLIQUÉES À L'IMAGERIE MÉDICALE

par

Gaël C. Sitzia-Verleure

Département de physique
Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures
en vue de l'obtention du grade de
Maître ès sciences (M.Sc.)
en Physique

novembre 2003

Université de Montréal
Faculté des études supérieures

Ce mémoire intitulé

MÉTHODES THERMODYNAMIQUES APPLIQUÉES À L'IMAGERIE MÉDICALE

présenté par

Gaël C. Sitzia-Verleure

a été évalué par un jury composé des personnes suivantes :

Claude Leroy
(président-rapporteur)

Bernard Goulard
(directeur de recherche)

Jean-Marc Lina
(co-directeur)

Raynald Laprade
(membre du jury)

Mémoire accepté le:

_____

En mémoire de Jacqueline Pardessus-Verleure.

(...) Ce n'était pas un point de coté, c'était un cancer de biais. Y avait à mon insu, sous-jacent à mon flanc, scattérisant mes bronches, comme un crabe affamé qui me brouttait le poumon. Le soir même, chez l'écailler du coin, j'ai bouffé un tourteau. Ça nous fait Un Partout.

Pierre Desproges (1939-1988).

SOMMAIRE

Le sujet de cette recherche est l'application de principes de thermodynamique à la théorie de l'information pour l'analyse de signaux.
Le cas de l'imagerie médicale sera plus particulièrement mis en emphase par le biais de la mammographie, en utilisant la représentation multi-échelles (en ondelettes), l'utilisation des principes de thermodynamique aux maxima des modes en ondelettes permet d'estimer la rugosité locale.
Nous mettrons l'accent sur la relation de ce paramètre avec la présence de foyers cancéreux et de micro-calcifications, et l'évolution de ce paramètre au cours du temps.

MOTS CLEF : Thermodynamique, ondelettes, mammographie, imagerie médicale, cancer du sein.

SUMMARY

The subject of this research is the application of thermodynamic principles to the information theory for signal analysis.
The case of medical imagery will be looked more closely by the study of mammography, by using a multi-scale representation (in wavelets). The use of thermodynamical theories to the maxima of the modulus of the wavelets permits us to estimate the local roughness.
We will put a focus on the relation of this parameter with the presence of cancerous tumors and micro-calcifications, and the time evolution of this parameter.

KEY WORDS : Thermodynamic, wavelet, mammography, medical imaging, breast cancer.

REMERCIEMENTS

Je voudrais remercier mes co-directeurs Bernard Goulard et Jean-Marc Lina pour m'avoir permis de travailler sur la caractérisation de textures en mammographie au sein du groupe Physnum du Centre de Recherches Mathématiques de l'Université de Montréal.

Je voudrais ensuite remercier Louise Lafortune, Louis Lemay, Laurent Lewis et Luc Turbide du département de Physique de l'université de Montréal, Muriel Pascaletti, Vincent Masciotra, Louis Pelletier, Daniel Ouimet, Béatrice Kowaliczko et Christian Leger du Centre de Recherches Mathématiques de l'université de Montréal, Marcelle Bertrand du service de polycopie et surtout mes parents et ma famille pour leur aide, compréhension et soutien tout au long de cette maîtrise.

Cette recherche doit aussi son accomplissement au Prof. Alain Arnéodo du laboratoire de physique de l'Ecole Nationale Supérieure de Lyon (France), au Dr. Caroline Fourcade de la faculté de pharmacie de l'université de Montpellier I (France) ainsi qu'au grand Dr. Joakim Valéro du service de gynécologie obstétrique maternité niveau 3 (REA NEONAT) du Prof. Oury de l'hôpital Robert Debré de Paris XIX (France) dont les expertises autant mathématique, physique que médicale se sont montrées vitales pour l'achèvement de la programmation et une plus fine compréhension de la problématique.

A Claire Dupuy-Dourreau, pour les longues heures qu'elle a passé à corriger ce mémoire avec une patience et une ténacité exemplaire, pour cette lutte acharnée contre les fautes d'accords, de temps et de syntaxes, je la remercie du fond du coeur.

Je remercie aussi Diego Clonda, Jean Daunizeau, Philippe St-Jean, Anne Bellio, Christophe Grova, Karine Christophe, Ervig Lapalme, Jacky Paquereau et Marc Bergevin, les membres et amis du groupe Physnum du centre de Recherches Mathématiques de l'université de Montréal, tant pour leur aide que leurs conseils.

De manière plus personnelle, je voudrais remercier chaleureusement mes amis, Jérôme de san Fulgencio, Suzanne Alves, Francis Bourgeois, Xavier Herbet, Nicolas Mahé, Jérôme Meyran, Juliette Sirinelli, Violette Finet, Josée Bérubé, Irina

Kezele, Caroline Coutu, Catherine Cyr-Gagnon, Fernand Rocholl, Thomas Phillips, Vincent Guerrieri, Andreas Pfizenmaier, Laurie McNeill, Verena Fischer, Richard D. Anderson, la famille Blonde, Kamila Belhocine, Yassine Kaboukie, Remi Montroty, Lise Genest, Denis Jacquerye, Frédéric Soustra, Clarisse De Los Santos, Francesca Ruffini, Zayneb Beynis, Mae K. Phitiére, David Gomez-Ullatte, Michèle Titcombe, Dominique Martin, Florence Vallana, Yann Berton, Aaron Barnes, Romain Caruso, Antoine Busi et toi aussi qui lit ces lignes, que j'ai oublié dans cette longue liste de gens merveilleux, et à qui je pense aussi en écrivant ces mots, pour m'avoir motivé jusqu'à la fin du mémoire allant même jusqu'à défier le froid arctique montréalais et mes sautes d'humeurs pour me soutenir tout au cours de ces deux dernières années.

Je voudrais enfin finir ces remerciements par un grand merci au personnel de la Clinique Poulpique pour son incontestable soutien et son support moral admirable.

Table des matières

Table des figures

Liste des tableaux

INTRODUCTION

De tout temps, l'homme a tenté de simplifier, de comprendre et de maîtriser le monde qui l'entourait. Ainsi les anciens prêtres égyptiens impressionnaient le peuple par l'utilisation de *magie thermodynamique* pour ouvrir les portes des temples. Le 19^{me} siècle verra la naissance de la science de la thermodynamique à proprement parler ainsi que d'outils permettant l'étude et la décomposition en temps-fréquence de signaux divers tels que la série et la transformée de Fourier.

Il faudra attendre 1984 afin de pouvoir effectuer une étude en espace et échelle de signaux grâce au développement des ondelettes par la communauté physicienne et mathématicienne par l'intermédiaire de Grossman et Morley. Les ondelettes serviront alors de microscope mathématique pour extraire des informations sur les propriétés en échelles d'objets fractals.

Dès le début des années 90, Arnéodo, Bacry et Muzy utilisent alors les ondelettes pour réviser le formalisme multi-fractal. Cette nouvelle approche est basée sur la thermodynamique et permettra entre autres la caractérisation de la rugosité des signaux étudiés.

On peut étudier la rugosité de signaux deux dimensions grâce au coefficient de Hölder. Cette rugosité est basée sur l'étude des singularités locales du signal à travers les échelles. En appliquant ces principes plus particulièrement aux mammographies, nous pourrons alors localiser les variations de rugosité caractérisant les tissus cancéreux et les micro-calcifications à l'origine des zones cancéreuses.

Le cancer est une des maladies les plus répandues dans les pays industrialisés. Le cancer du sein est une des principales causes de mortalité chez la femme. Deux femmes sur dix-neuf au Canada sont atteintes par cette maladie. Chez les femmes diagnostiquées, 23% ont moins de 50 ans, 46% ont entre 50 et 69 ans et 32% ont plus de 70 ans (FQC, 2001).
Grâce à un dépistage plus précoce et à l'efficacité accrue des traitements médicaux (auto-examen des seins, examen clinique des seins, mammographie), on constate une baisse constante du taux de mortalité due au cancer du sein depuis 1990 (FQC, 2001 ; Société Canadienne du Cancer (SCC), 2001).

Le but de cette recherche est l'application de certains principes de thermodynamique à la théorie de l'information pour l'analyse de signaux. L'emphase sera plus

particulièrement mise sur le cas de l'imagerie médicale par le biais de la mammographie, par l'utilisation de la représentation multi-échelles (en ondelettes), ainsi que des principes de thermodynamique aux maxima des modes en ondelettes afin d'estimer la rugosité locale. Nous mettrons l'accent sur la relation de ce paramètre avec la présence de foyers cancéreux et de micro-calcifications, et l'évolution de ce paramètre au cours du temps.

Dans le chapitre 1, nous commencerons donc par quelques rappels théoriques sur la thermodynamique et son évolution jusqu'à la mécanique statistique. Par la suite nous survolerons les fractales et multi-fractales, ainsi que les ondelettes en décrivant plus particulièrement la Méthode des Maxima dans une Transformation en Ondelettes (MMTO) et la thermodynamique, nous permettant d'introduire les coefficients de Hölder et de Hausdorff. La Méthode des Maxima des Transformées d'Ondelettes (MMTO) redéfinit le signal sous la forme d'un squelette espace-échelle d'où l'on peut extraire son spectre de singularité grâce aux comportements des fonctions de partition définies sur ce même squelette. Nous expliquerons par la suite les hypothèses de travail pour l'utilisation de principes thermodynamiques pour la pré-détection des cancers et la mammographie.

Dans le chapitre 2, nous orienterons notre exposé vers la mammographie en précisant les caractéristiques physiologiques mammaires ainsi que les différentes méthodes de formation des images mammographiques numériques. Nous profiterons de ces explications pour émettre des hypothèses sur l'origine de la fractalité de la texture mammaire et les résultats que nous avons retrouvés, en particulier les travaux de Cadwell sur la texture mono-fractale du parenchyme mammaire par le calcul de l'exposant de Hurst (h). Il nous a été ainsi possible de retrouver la distinction entre les tissus mammaires adipeux ($h = 0.3$) des tissus mammaires denses ($h = 0.65$) trouvée dans le travail de Kestener et al [**13**].

Le chapitre 3 expliquera les détails et innovations du travail fourni pour la programmation du code permettant de calculer la rugosité locale d'un signal 1D ou d'une image 2D, ainsi que les suppositions et approximations raisonnables qui ont permis l'accomplissement de cette recherche. Nous expliquerons plus précisément l'algorithme de fonctionnement du programme Waveplz et de la gestion du traitement d'images de tailles importantes comme une mammographie. Nous analyserons les techniques employées pour l'implantation numérique de la Méthode des Maxima des Transformées en Ondelettes (MMTO), le chaînage des lignes de maxima, le calcul des fonctions de partition, du poids de Boltzmann, de l'exposant de Hölder et de la dimension de Hausdorff. Nous terminerons cette partie en expliquant les différents programmes développés pour la visualisation des résultats (voir figure 1).

Nous finirons dans le chapitre 4 par l'exposé des résultats expérimentaux obtenus

lors de la calibration du programme de détection de texture sur des signaux de synthèse mono-fractale 1D et 2D (Brownien fractionnaire). Nous expliquerons les modifications apportées au code et leurs origines. Nous terminerons par l'exposé des derniers travaux sur l'isolation de foyer cancéreux et de micro-calcifications, l'évolution temporelle de la texture mammaire au cours du temps par l'étude de 4 cas cliniques pour une meilleure classification et détection des singularités sur les mammographies. En conclusion, nous débattrons de la stabilité et de la confiance que l'on peut porter au programme, les améliorations possibles et les développements futurs à effectuer.

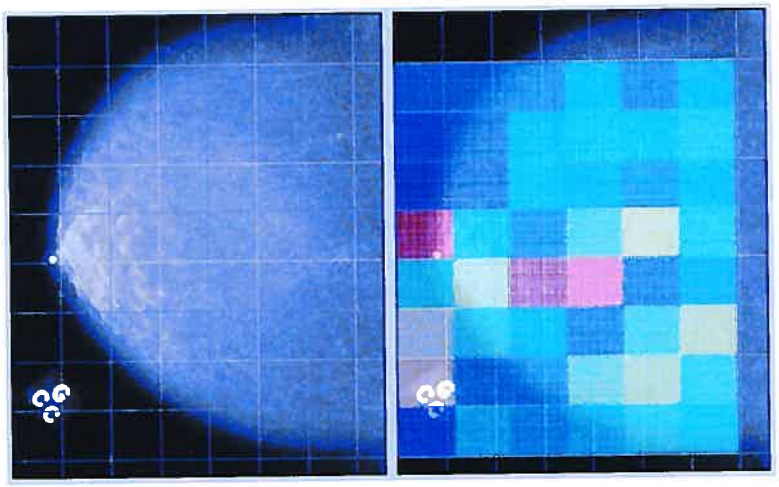


FIG. 1. Visualisation de la détection de micro-calcifications et de textures cancéreuses dans une mammographie

Chapitre 1

MODÉLISATIONS

Le but de ce chapitre est de poser les bases théoriques physiques et mathématiques qui nous ont permis d'élaborer un programme de caractérisation de texture dans un signal avec une application toute particulière à la pré-détection de micro calcifications et de textures cancéreuses dans les mammographies. Nous établirons les principes fondamentaux qui nous ont permis d'utiliser les principes thermodynamiques tels que la notion d'entropie dans le domaine du traitement de l'image et de l'information qu'elle porte.

La première section nous plongera dans le monde de la thermodynamique en montrant l'évolution de la notion d'entropie à travers les travaux des chercheurs qui ont permis son application à divers domaines d'étude. Nous parlerons ainsi de la loi de l'écoulement de la chaleur par Joseph Fourier en 1811 et de l'invention du terme d'entropie par Clausius en 1865, en passant par l'établissement du deuxième principe de thermodynamique par Sadi Carnot en 1824. Nous expliquerons ensuite le passage de la thermodynamique à la physique statistique (ou mécanique statistique) ainsi que la vision révolutionnaire de l'entropie par Ludwig Boltzmann. Le démon de James Clark Maxwell nous permettra de poursuivre dans le monde de la théorie de l'information. Nous y verrons comment Claude Shannon et Warren Weaver relièrent l'entropie, et la notion de désordre qu'elle implique, aux manques d'information dans les transferts de données. Cette nouvelle interprétation de l'entropie appliquée aux signaux et à l'information par Shannon justifiera ainsi l'application à la mammographie que nous en avons faite au cours de cette recherche.

Dans la deuxième section, nous procéderons à un bref rappel sur les fractales nous permettant d'établir les bases du formalisme multifractal. Nous pourrons ainsi comprendre le rapport entre mesures multifractales et spectre de singularité.

Dans la troisième section, nous étudierons les ondelettes et la formation de leurs transformées, essentielles pour la compréhension de la méthode thermodynamique employée dans ce traitement particulier de l'image mammographique. Nous expliquerons alors le rapport entre transformation en ondelettes et analyse de singularités dans les signaux.

Dans la quatrième section, nous établirons le rapport entre les trois sections précédentes pour expliquer la méthode de la thermodynamique des fractales appliquées aux ondelettes. Nous commencerons par explorer la Méthode des Maxima des Transformées en Ondelettes (MMTO), puis son application dans la description thermodynamique des mesures multifractales pour la mesure de textures en imagerie.

1.1. De Boltzmann à Shannon : évolution du concept d'entropie

Dans cette section nous allons discuter de l'évolution du concept d'entropie en commençant par l'étude des principes fondateurs de la thermodynamique et de la création de ce terme par Clausius. Puis nous détaillerons le passage du macroscopique au microscopique par le développement de la physique statistique et l'explication de l'entropie par Boltzmann. Enfin, nous finirons par le passage de la physique statistique à la théorie de l'information et de l'interprétation de l'entropie par Shannon.

1.1.1. Thermodynamique

Depuis la découverte du feu, l'être humain a toujours été fasciné par l'énergie que dégageait cette flamme. Lui permettant de changer de statut dans la nature, cette maîtrise de l'énergie lui permit de faire progresser l'humanité par le contrôle des éléments naturels tels que le froid de la nuit ou la cuisson des aliments. De nombreuses inventions peuvent être attribuées à la maîtrise de la chaleur et ses applications comme l'aléolipile de Héron d'Alexandrie ou le système d'ouverture de porte de temple. Ce dernier utilisait la force de la vapeur pour faire circuler de l'eau dans un seau qui provoquait l'ouverture de la porte [**37**] (voir figure 1). Il faudra cependant attendre le 19ème siècle avant que l'homme ne commence à comprendre la science qui se cache derrière toutes ces machines à vapeur.

1.1.1.1. *Bases de thermodynamique*

Joseph Fourier fut l'un des pionniers dans la science de la chaleur (ou thermodynamique) avec son étude sur l'écoulement de la chaleur en 1811. La thermodynamique n'utilise alors aucune hypothèse quant à la structure atomique de la matière. Elle explique le comportement de la matière ou des systèmes en fonction de la variation de température T, de la pression P, du volume V, de la masse m, l'énergie comme la chaleur Q ou le travail W et de l'entropie S.

Le but de la thermodynamique est ainsi l'étude du fonctionnement et le bilan énergétique des machines thermiques mais aussi des échanges ou transferts de chaleur d'un système à l'autre.

On considère alors le système thermodynamique comme une certaine portion

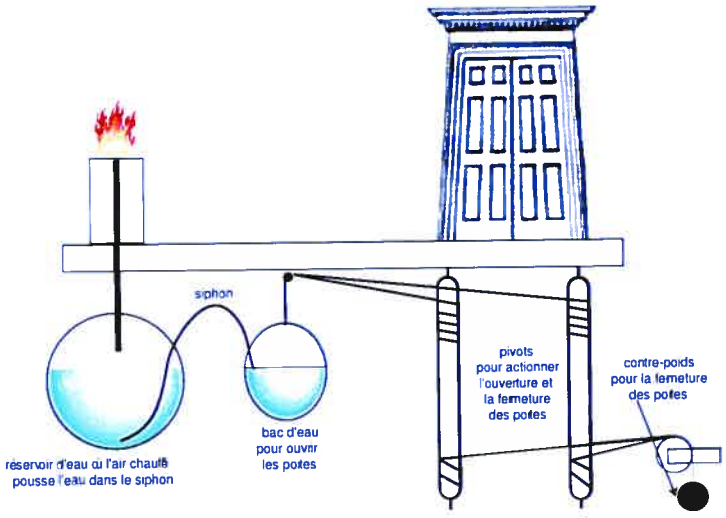


FIG. 1. Ouverture et fermeture de portes de temple par utilisation de chaleur

de l'espace délimitée par des frontières renfermant un gaz, un solide ou un liquide qui peuvent être isolés de tout transfert de chaleur ou de matière avec l'extérieur (adiabatique), ouvert ou fermé.

Le gaz étudié est considéré, dans le cadre de la thermodynamique, comme parfait. On entend par là que les atomes ou les molécules composant ce gaz sont assimilés à des masses ponctuelles n'ayant aucune interaction entre elles (énergie potentielle nulle). On assume aussi que seul le choc de ces molécules sur les parois de l'enceinte est responsable de la pression P que l'on mesure. Les gaz parfaits sont ainsi régis par une équation suivante :

$$PV = nRT \tag{1}$$

où P la pression du gaz dans le système, V le volume qu'il occupe, n le nombre de moles de gaz ($n = \frac{N}{N_A}$, N le nombre d'atomes ou de molécules du gaz dans l'enceinte et $N_A = 6.0222.10^{26}$ molécules par kilomole le nombre d'Avogadro), $R = 8.3143.10^3$ Joule par kilomole par Kelvin la constante universelle des gaz et T la température du gaz [**28**].

La thermodynamique dite classique décrit uniquement ces systèmes en terme d'état initial et final de leur évolution et dresse le bilan énergétique. Elle ne

cherche pas à élucider les mécanismes des transformations subites lors des changements d'état de la matière [31].

C'est dans ce cadre idéalisé que la première loi de la thermodynamique est énoncée :

> La somme algébrique du travail W et de la chaleur Q échangés par le système avec le milieu extérieur est égale à la variation ΔU de son énergie interne.

Elle est suivie en 1824 par l'énoncé de la seconde loi de la thermodynamique, aussi appelé principe d'évolution, par Sadi Carnot sur l'irréversibilité des transformations. Le bilan énergétique des systèmes ne permet pas de prévoir le sens d'évolution. Il fallut donc postuler l'irréversibilité des processus observés expérimentalement, telle que la fonte d'un cube de glace dans un four. Son énoncé mathématique s'exprime sous la forme suivante :

$$\sum \frac{\delta Q}{T} \geq 0. \tag{2}$$

Si l'équation 2 montre un signe égal $(=)$, le cycle est réversible. Si elle implique un signe supérieur $(>)$, le cycle est irréversible [31].

1.1.1.2. *L'entropie de Clausius*

C'est la thermodynamique qui a vu naître le terme d'entropie en 1865 dans un article de Rudolf Clausius fondateur des principes thermodynamiques [26]. Le mot entropie vient du grec *trope* qui signifie changer de direction. L'entropie S, dont la notion fût introduite par le second principe de la thermodynamique, décrit alors le comportement des systèmes en apportant la notion d'irréversibilité de certains échanges d'énergie. On l'exprime mathématiquement par la formule :

$$dS > \frac{dQ}{T}. \tag{3}$$

L'entropie S croît si le système tend vers son équilibre. S est maximum si le système est à l'équilibre.

L'entropie de Clausius est alors utilisé comme représentation mathématique de l'irréversibilité de certains phénomènes. Ainsi l'entropie d'une tasse de café au lait aura une entropie supérieure à la somme des entropies du café et du lait séparés ; la différence étant l'entropie du mélange [30].

1.1.2. Mécanique statistique

Cette sous-section traite du passage de la thermodynamique classique à la thermodynamique statistique que Ludwig Boltzmann tentera toute sa vie de clarifier la notion d'entropie, la faisant évoluer de l'expression de l'irréversibilité

d'un système à la quantification de son désordre et l'interprétation de ses diverses variations. Nous introduirons aussi les notions de d'ensemble de Gibbs et de transition de phases, notions essentielles pour la compréhension de la détection de textures par méthode multifractale.

1.1.2.1. *Bases de mécanique statistique*

Vers la fin du dix-neuvième siècle, Boltzmann développa la thermodynamique statistique (ou mécanique statistique) afin de pouvoir prédire les diverses propriétés macroscopiques des gaz à partir de leur description microscopique [**27**]. Il fut alors possible de poser l'équation d'état de l'énergie interne U d'un gaz parfait (équation 4) en utilisant la représentation de microscopique de la température T.

$$U = \frac{3}{2} N k_B T \tag{4}$$

avec $k_B = 1.3806.10^{-23}$ Joule par Kelvin la constante de Boltzmann et N le nombre de particules dans le gaz. On utilisera dès à présent les termes d'énergie d'agitation thermique $k_B T$ et de facteur de Boltzmann $\beta = \frac{1}{k_B T}$.

Dans cette représentation, l'énergie interne étant en majorité due à l'agitation $< v^2 >$ des particules de gaz, on peut aussi l'écrire tel que :

$$U = N(\frac{1}{2} m < v^2 >) \tag{5}$$

Grâce à l'étude des collisions entre particules, on peut passer aisément de l'équation d'état des gaz parfaits (voir 1) à l'équation des gaz de Van Der Waals (gaz imparfaits) permettant une description plus réaliste des gaz.

$$(p + b)(V - a) = N k_B T \tag{6}$$

avec p la pression du gaz et V le volume de gaz étudié.

1.1.2.2. *L'entropie de Boltzmann*

En thermodynamique nous distinguons les variables extensives[1] des variables intensives[2] auxquelles elles sont conjuguées. Le premier principe de thermodynamique stipule que seules ces variables suffisent à décrire l'équilibre du système.

La mécanique statistique apporte alors un complément d'information sur la structure du gaz en ajoutant la notion d'états microscopiques du système. On ne voit

[1] Les variables extensives sont les variables dépendantes de la masse telles que l'énergie interne U, le volume du système V, le nombre de particules N, qui correspondent aux équilibres thermique, mécanique et osmotique du système.

[2] Les variables intensives sont indépendantes de la masse tels que la température T, la pression p et le potentiel chimique μ.

plus alors le système dans son ensemble mais aussi à travers le positionnement des atomes et molécules qui le compose dans une portion de l'espace de l'univers limité par des frontières. Cette approche statistique permet donc d'affecter à chacune des Ω configurations possibles une équiprobabilité de $\frac{1}{\Omega}$ [**30**].

Suite aux travaux de Maxwell, Gibbs et de lui-même, Boltzmann redéfinira en 1877 le terme d'entropie introduit par le second principe de thermodynamique (équation 2). L'entropie de Boltzmann $S(U, V, N)$, ou entropie statistique, représentera alors une mesure du désordre donnant une information sur l'état d'homogénéité du système.

$$S = k_B ln(\Omega), \tag{7}$$

avec Ω le nombre d'états microscopiques du système et k_B la constante de Boltzmann [**32**].

Le bilan d'énergie sur les trois équilibres de volume, de pression et de potentiel chimique conduit à :

$$TdS = dU + pdV - \mu dN \tag{8}$$

Cela nous permet de déduire les trois équations d'états du système :

$$\begin{aligned} T &= (\frac{\partial U}{\partial S})_{V,N}, \\ p &= -(\frac{\partial U}{\partial V})_{S,N}, \\ \mu &= (\frac{\partial U}{\partial N})_{S,V}. \end{aligned} \tag{9}$$

Comme nous allons le voir dans la prochaine section, il est parfois très utile de travailler avec des variables intensives. On utilise alors des transformées de Legendre qui permettent de remplacer une variable extensive par sa variable conjuguée. L'énergie libre F, ou fonction de Helmholtz[3], est très utilisée en physique :

$$F = U - TS \tag{10}$$

qui est une fonction de la température T, le volume de gaz V et le nombre de particules N. L'énergie libre F conduit alors naturellement à d'autres équations d'état :

[3] le choix de la notion F vient du rôle joué par cette fonction dans les transformations monothermes pour lesquelles le système est en relation avec une seule source [**29**].

$$
\begin{aligned}
S &= -(\frac{\partial F}{\partial T})_{V,N}, \\
p &= -(\frac{\partial F}{\partial V})_{T,N}, \\
\mu &= (\frac{\partial F}{\partial N})_{T,V}.
\end{aligned}
\tag{11}
$$

On utilise aussi l'enthalpie libre G, ou fonction de Gibbs :

$$G = F + pV \tag{12}$$

qui est une fonction de l'énergie libre F, la pression p et le volume de gaz V. L'enthalpie libre G est aussi reliée au potentiel chimique μ et au nombre de particules N par la relation de Gibbs-Duhem :

$$G = N\mu. \tag{13}$$

Pour ce dernier potentiel thermodynamique, on a les trois équations d'états suivantes :

$$
\begin{aligned}
S &= -(\frac{\partial G}{\partial T})_{p,N}, \\
V &= -(\frac{\partial G}{\partial p})_{T,N}, \\
\mu &= (\frac{\partial G}{\partial N})_{T,p}.
\end{aligned}
\tag{14}
$$

1.1.2.3. *Ensembles de Gibbs*

Le premier postulat de physique statistique stipule que tous les micro-états d'un système isolé sont accessibles et équiprobables. La probabilité de se trouver dans un micro-état donné est donc $1/\Omega$. Si ce n'est pas le cas, le système est hors équilibre et va évoluer pour satisfaire cette condition.

Le second postulat indique qu'à un instant donné un système en équilibre se trouve dans un micro-état et un seul. Au cours du temps, le système change de micro-état. Ceci est notamment dû aux légères fluctuations associées à l'incertitude (Heisenberg) sur l'énergie U du système.

Si on regarde le système sur un temps très long, on doit trouver que le temps passé dans chacun des micro-états est le même. Autrement dit, le système aura visité tous les micro-états au bout d'un temps infini, causé par les fluctuations d'énergie [**16**].

La physique statistique fait ainsi état de différentes descriptions du système thermodynamique à travers différents ensembles dits de Gibbs.

L'ensemble micro-canonique est l'ensemble des micro-états d'un système isolé[4]. U, N et V sont alors constants et la probabilité P_i de trouver le système dans un micro-état i est (avec $\sum_i P_i = 1$) :

$$P_i = \frac{1}{\Omega} = e^{-\frac{S}{k_B}}, \tag{15}$$

où S l'entropie de Boltzmann (voir équation 7). Pour un système isolé, le simple dénombrement des micro-états donne l'entropie S. S est donc toujours positif ou nul.

Dans l'ensemble canonique, le système peut échanger de l'énergie avec le réservoir mais pas de matière. Il faut donc considérer un plus grand nombre de micro-états. On définit ici l'énergie $U_j(i)$ comme le système ayant une énergie U_j dans une configuration moléculaire correspondant au micro-état i. Donc à chaque énergie $U_j(i)$, il va falloir dénombrer $1 \leq i \leq \Omega(U_j(i))$ micro-états. L'ensemble de tous les micro-états ij est l'ensemble canonique. La probabilité P_{ij} de trouver le système dans un micro-état i à l'énergie U_j et à une température T sera donc :

$$P_{ij} = \frac{1}{Z} e^{-\frac{U_j(i)}{k_B T}}, \tag{16}$$

où la fonction Z représente le facteur de normalisation de toutes les probabilités. Z est aussi appelée fonction de partition et est définie par :

$$Z = \sum_j \sum_i e^{-\frac{U_j(i)}{k_B T}}. \tag{17}$$

Dans l'ensemble canonique, il y a un poids statistique qui est associé à chaque micro-état. Ce poids statistique est une simple exponentielle de l'énergie interne normalisée par l'agitation thermique $k_B T$. C'est le facteur de Boltzmann ou poids de Boltzmann. Il est parfois pratique de remplacer $1/k_B T$ par β dans le facteur de Boltzmann [**27**].

Rappelons que la fonction de partition Z permet de calculer toutes les grandeurs thermodynamiques mesurables. Ainsi l'énergie moyenne $< U >$ du système peut être obtenue dans l'ensemble canonique grâce aux variables T, V, N et l'énergie libre $F(T, V, N)$:

[4] système adiabatique avec aucun échange de matière ou d'énergie avec le réservoir dans lequel le système se trouve.

$$Z = e^{-\frac{F}{k_B T}}. \tag{18}$$

car

$$< U >= F + TS = F - T(\frac{\partial F}{\partial T})_{V,N} \tag{19}$$

est vérifiée lorsque $F = -k_B lnZ$. On obtient ainsi :

$$S = -(\frac{\partial F}{\partial T})_{V,N} = k_B lnZ - \frac{k_B T}{Z}(\frac{\partial Z}{\partial T})_{V,N} = k_B(lnZ + \beta < U >) \tag{20}$$

Dans l'ensemble grand canonique, on considère tous les micro- états i pour lesquels le système peut à la fois échanger de la chaleur et des particules avec le réservoir [**16**].

Dans ces conditions, la probabilité P_{ijN} d'avoir le système dans un micro-état i avec N particules et à l'énergie $U_j(i)$ sera :

$$P_{ijN} = \frac{1}{Q} e^{-\frac{U_j(i) - \mu N}{k_B T}}, \tag{21}$$

avec Q la fonction de partition grand canonique telle que :

$$Q = \sum_N \sum_j \sum_i e^{\frac{\mu N}{k_B T} - \frac{U_j(i)}{k_B T}} = \sum_N e^{\frac{N\mu}{k_B T}} Z(T, V, N) \tag{22}$$

On peut retrouver les variables thermodynamiques à partir de Q :

$$pV = k_B T lnQ. \tag{23}$$

1.1.2.4. *Transition de phases*

Au cours des années, diverses méthodes ont décrit de manière plus réaliste les collisions et les transports de particules comme l'équation de Boltzmann ou le formalisme de Langevin. La mécanique quantique a aussi fait son apparition dans ce domaine en permettant d'étudier de manière classique des gaz quantiques tels que les gaz de Fermi-Dirac et de Bose-Einstein. Il n'existe cependant pas que des collisions au niveau microscopique dans ces représentations. Différentes interactions agissent entre les particules comme les interactions coulombiennes, de Van Der Waals, élastiques et dipolaires magnétiques. Ces interactions ont tendance à ordonner, orienter et organiser les entités microscopiques en structures bien définies lorsque l'agitation thermique est petite devant l'énergie potentielle. En abaissant la température du système, on assiste donc à une transition de la matière vers une phase ordonnée [**16**].

Ces transitions de phases correspondent à des changements des propriétés physiques de la matière. De nombreux phénomènes physiques sont liés aux transitions de phases telles que le para-ferromagnétisme, la ferroélectricité, la superfluidité, la supraconductivité, les transitions structurales, les cristaux liquides, la percolation ou encore la vorticité.

Afin de modéliser ces transitions de phases, la physique statistique a alors inventé des modèles de spins comme par exemple le modèle d'Ising. L'énergie interne y est définie via toutes les interactions entre spins et sa minimisation conduit à un ordre de basse température.

Dans les années 70, un nouveau modèle de spins fait alors son apparition. La théorie du groupe de renormalisation montre alors que les exposants critiques caractérisant les transitions de phases ne dépendent que de la symétrie de l'interaction entre entités microscopiques et de la dimensionnalité du système. Le plus surprenant est que les transitions de phases soient indépendantes de la nature des interactions. Une transition naturelle pourrait donc être équivalente à une transition dans un modèle de spins [**16**]. Cette théorie du groupe de renormalisation repose sur l'invariance d'échelle au point critique. Dans le système apparaissent des structures fractales (voir section 1.2).

Comme nous le savons déjà, les transitions de phases correspondent à des discontinuités de diverses fonctions macroscopiques. Ehrenfels proposa la classification en deux ordres de ces transitions [**32**].

Les transitions dit du *premier ordre*, bien rendues en thermodynamique par la relation de Clapeyron, consistent en une discontinuité de $\frac{\partial G}{\partial T}$, causant la présence de chaleur latente, une variation de l'entropie S ainsi que de la densité ρ comme par exemple dans la transition entre liquide et gaz.

Il existe aussi les transitions du *second ordre* qui elles correspondent à une discontinuité de $\frac{\partial^2 G}{\partial^2 T}$ qui est plus complexe à étudier puisqu'elle n'induit pas forcément une variation de la densité mais plutôt une brisure de la symétrie (transition ferro-paramagnétique).

Dans l'ensemble canonique, la fonction de partition Z permet la dérivation de diverses propriétés thermodynamiques du système échangeant de l'énergie avec son environnement (équations 19 et 20). Les transitions correspondent alors à des singularités dans l'énergie libre F et l'entropie S. Pour trouver les points critiques,

il suffit alors de chercher les températures T_c qui annulent la fonction de partition $Z(T, V, N)$ [**16**]. Ceux sont ces transitions de phases que nous caractériserons par la suite dans les mammographies en observant les changements de texture de l'image (voir section 1.4).

1.1.3. Théorie de l'information

Dans cette sous-section, nous allons présenter le passage de l'entropie statistique de Boltzmann à l'entropie de Shannon dans la théorie de l'information. Nous justifierons ainsi l'utilisation de la thermodynamique dans le cadre de la caractérisation de texture.

Nous commencerons par exposer l'expérience suggérée par Maxwell permettant le passage à la théorie de l'information. Nous continuerons par poser les bases de probabilité sur la théorie de l'information nécessaires afin de comprendre l'établissement de l'entropie par Shannon.

1.1.3.1. *Le démon de Maxwell*

En 1867, Maxwell imagina une expérience dont la première conséquence fût la violation du second principe de thermodynamique (voir équation 2).

Imaginons deux enceintes A et B thermiquement isolées contenant un gaz composé de N molécules réparti sur les deux enceintes. La densité et la pression dans les deux enceintes sont identiques. Une petite ouverture est faite entre les deux enceintes, fermée par un opercule ne permettant que le passage d'une molécule de gaz à la fois. Cet opercule est géré par un démon, le démon de Maxwell. Le démon s'arrange alors pour ne laisser passer que les molécules dans un sens, ayant pour conséquence d'avoir à la fin de l'expérience les N molécules dans une seule des deux enceintes. Cette réduction de volume ayant été faite sans compression, l'entropie du gaz a donc diminué violant ainsi le second principe de la thermodynamique [**30**].

1.1.3.2. *Information et probabilité*

En 1929, Léo Szillard remarque que le démon de Maxwell ne peut décroître l'entropie du gaz que par l'obtention d'*information* sur le gaz afin de pouvoir effectuer ce tri.

Cette quantité d'information fût quantifiée en 1948 lorsque Claude Shannon et Warren Weaver travaillèrent à l'optimisation de la transmission de message en téléphonie et en particulier chiffrer la quantité d'information véhiculée par les messages à transmettre. En prenant un message m et la probabilité p_m (avec pour tout m, $0 \leq p_m \leq 1$ et $\sum_m p_m = 1$) qu'il soit transmis, nous pouvons alors

associer une quantité d'information I_m associé à ce message. Cette quantité se calcule grâce à :

$$I_m = log_2 \frac{1}{p_m} \tag{24}$$

Cela signifie donc que plus la probabilité que ce message arrive est grande, moins l'information qu'il contient est importante[5]. Notons que l'usage du log_2 est dû au fait que les messages en communication se transmettent en bits [**30**].

Considérons maintenant la transmission d'un nombre Ω de messages m. On appellera I mesurant la valeur moyenne de la quantité d'information que le destinataire va acquérir. Si le nombre Ω de messages à transmettre est assez grand on peut raisonnablement penser que la probabilité p_m pour que le destinataire reçoive un message m est égale à $\frac{1}{\Omega}$. La mesure I représente alors l'incertitude ou le manque d'information et se calcule à partir de 24 et en considérant que $p_m = \frac{1}{\Omega}$:

$$I = \sum_m p_m I_m = log_2 \Omega \tag{25}$$

1.1.3.3. *L'entropie de Shannon*

Shannon, Szilard, Gabor, Rothstein et Brillouin utilisent le concept d'entropie en théorie de l'information pour démontrer, entre autres, l'équivalence entre néguentropie (l'opposé de l'entropie) et information (1940-1950).

Brillouin montra qu'à la fin de l'expérience de Maxwell, le démon ne possédait pas plus d'informations que l'observateur extérieur, a savoir que tout le gaz se trouvait complètement dans une des enceintes. Il établit donc que l'entropie S du système était proportionnel à l'incertitude I sur la quantité d'information selon la loi :

$$\begin{aligned} S &= k_b log \Omega, \\ &= k_b log 2 \frac{log \Omega}{log 2}, \\ &= k log_2 \Omega. \end{aligned} \tag{26}$$

avec $k = 0,96.10^{-23} JK - 1bit^{-1}$.

Nous obtenons ainsi l'entropie suggérée par Shannon, en théorie de la communication, et pouvant être utilisée pour maximiser le transfert de bit sous une

[5] Une analogie possible est celle d'un coureur. Plus la probabilité p_m qu'il gagne la course est forte le moins l'information I_m de sa victoire à cette course est surprenante [**30**].

contrainte de qualité [15]. L'entropie que l'on obtient alors est une expression probabiliste dans l'ensemble canonique de la forme :

$$S = -k_B \sum_j \sum_i P_{ij} log P_{ij} \tag{27}$$

Dans l'ensemble micro-canonique, on retrouve l'entropie de Shannon sous la forme :

$$S = -k_B \sum_i P_i log P_i \tag{28}$$

Après être successivement passé de la caractérisation de l'irréversibilité de certains processus sous Clausius, puis comme représentation du désordre sous Boltzmann, grâce à Shannon l'entropie devient alors la mesure du manque d'information en théorie de la communication.

Dans les années 80-90, la thermodynamique et la mécanique statistique reprirent un regain d'attention lorsqu'elles furent utilisées pour le calcul du spectre de singularité[6] dans un signal à l'aide du formalisme multifractal et des ondelettes par Arnéodo, Bacry, Muzy et al [7].

1.2. Les fractales et multi fractales

Why is geometry often described as 'cold' and 'dry'? One reason lies in its inability to describe the shape of a cloud, a mountain, a coastline, or a tree. Clouds are not spheres, coastlines are not circles, and bark is not a smooth surface, nor does lightning travel in a straight line (...) The existence of these patterns challenges us to study those forms that Euclid leaves aside as being 'formless', to investigate the morphology of the 'amorphous'.
Benoît B. Mandelbrot.

Dans cette section, nous allons faire un rappel des notions importantes sur les fractales et multifractales et leurs implications dans le traitement de signaux et la caractérisation de sa texture.

Le traitement de tout signal dépend de la nature de ce signal ainsi que de l'information que l'on cherche à extraire. Un signal sera de nature stationnaire si ses propriétés sont statistiquement invariantes au cours de son évolution. Des signaux où apparaissent des évènements transitoires que l'on ne peut prévoir même sous forme statistique serait alors considéré comme non stationnaire.

[6] Appelé aussi spectre de Hölder, le spectre de singularité calculé à partir des exposants de Hölder ponctuels (voir 30) permet une classification plus efficace de la dimension de Hausdorff (voir équation 1.3.2).

Les signaux les plus stables sont généralement décomposés de façon canonique en combinaisons linéaires d'ondes à l'aide de transformées de Fourier (méthode plus rapide qu'une transformé en ondelettes). L'étude des signaux non stationnaires nécessite cependant des techniques différentes de l'analyse de Fourier dû essentiellement à leur nature irrégulière. Ces techniques appropriées à la non stationnarité incluent les ondelettes du type 'temps-fréquence' dont le domaine d'application est, plus particulièrement, l'ensemble des signaux quasi-stationnaires (voir section 1.3). Les ondelettes du type 'temps-échelle' sont ainsi adaptées aux signaux présentant une structure fractale.

Avant de nous lancer dans l'explication du formalisme multifractal et de ses applications, il nous faut tout d'abord expliquer le concept de fractale et de loi d'échelle comme l'a introduit Mandelbrot dans les années 70, et ainsi arriver aux principes fondateurs du formalisme multifractal qui nous permet de décrire les comportements irréguliers locaux et de calculer des coefficients de texture. D'après Lévy-Vèhel [17], la notion théorique centrale à laquelle renvoient les lois d'échelle est celle d'auto similitude - ou d'autosimilarité -, c'est-à-dire que *le tout ressemble (statistiquement) à la partie,* ou *la partie est (statistiquement) équivalente au tout.* Autrement dit, l'information acquise de l'observation de données est indépendante de la résolution. Mais allons un peu plus dans le détail de cette explication de la loi d'échelle par une incursion dans le monde des fractales.

1.2.1. Les fractales

D'après la définition de Mandelbrot en 1974, une fractale est un objet ayant la propriété d'autosimilarité, c'est à dire de se reproduire à une échelle différente[7]. On considère la fonction fractale comme une fonction auto-affine ce qui signifie que la fonction f subit une dilatation isotropique. La fonction diminue ainsi le long de l'axe des abscisses d'un facteur λ et remise à l'échelle de l'incrément de la fonction par un facteur différent λ^{-H} :

$$f(x_o + \lambda x) - f(x_o) \backsim \lambda^H (f(x_o + x) - f(x_o)) \tag{29}$$

avec $x_o \in \mathbb{R}$, $\Lambda > 0$ et $H \in \mathbb{R}$ le coefficient de Hurst, aussi connue sous le nom de coefficient de rugosité. Ce coefficient existe aussi sous forme locale $h(x_o)$. Pour le définir, il nous faut changer la formule 29 de la régularité de Hurst afin qu'elle devienne une quantité locale [8]. :

$$|f(x_o + l) - f(x_o)| \backsim l^{h(x_o)} \tag{30}$$

L'exposant de rugosité local $h(x_o)$ correspond alors au coefficient de Hölder de la fonction f en un point x_o lorsque $h(x_o) < 1$. $h(x_o)$ représente alors la force de la

[7] L'exemple typique pour les fractales est la mesure d'une côte maritime avec une règle à mesurer. Ainsi plus votre règle est petite plus sa longueur augmente.

singularité de la fonction f et est définie comme le plus grand exposant tel qu'il existe un polynôme $P_n(x - x_o)$ d'ordre n tel que :

$$|f(x) - P_n(x - x_o)| \leq C|x - x_o|^{h(x_o)} \tag{31}$$

avec x dans le voisinage de x_o et C une constante, il est rapide de prouver que si $h(x_o) \in]n, n+1[$ alors f est n fois différentiable et pas $n+1$ différentiable au point x_o. le polynôme $P_n(x - x_o)$ correspond alors au développement en série de Taylor de la fonction f autour de $x = x_o$ jusqu'à l'ordre n.

Une autre notion importante dans le domaine fractal est celle de *la dimension fractale*. Si $N(\epsilon)$ est le nombre minimal de boules de rayon ϵ qu'il faut pour recouvrir l'ensemble[8] S, la dimension fractale d_F d'un ensemble S dans $\mathbb{R}$ est définie par le comportement de la loi en puissance de :

$$N(\epsilon) \backsim \epsilon^{-d_F(S)} \tag{32}$$

La dimension fractale d'un objet auto-similaire peut être alors définie telle que :

$$d_F(S) = \frac{log(N(\epsilon))}{log(\epsilon)}, \tag{33}$$

où l'on peut interpréter ϵ comme le *facteur de magnification* et $N(\epsilon)$ le nombre d'objets auto similaires contenus dans une surface S.

Exemple 1 : dans le cas d'un carré de surface ϵ^2 (voir figure 2.a), nous pouvons calculer la dimension fractale de cet objet :

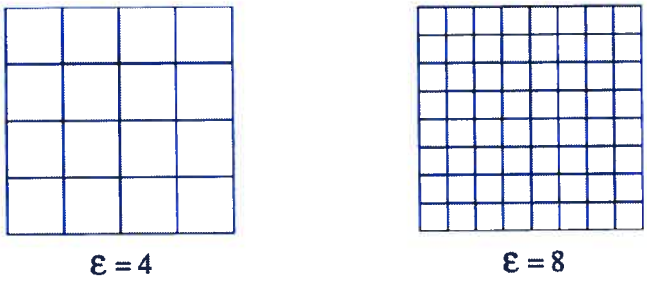
$$d_F(carr) = \frac{log(N(\epsilon))}{log(\epsilon)} = \frac{log(\epsilon^2)}{log(\epsilon)} = 2 \tag{34}$$

Exemple 2 : dans le cas un peu moins trivial du triangle de Sierpinski (voir figure 2.b), la dimension fractale pour $\epsilon = 2$ est :

$$d_F(carr) = \frac{log(N(\epsilon))}{log(\epsilon)} = \frac{log(3)}{log(2)} = 1.5850 \tag{35}$$

Passons maintenant au modèle plus subtil du formalisme multifractal et de l'analyse qui lui est associé, nous permettant de poser les premières pierres d'un algorithme pour le calcul du spectre de singularité d'un signal.

[8] L'ensemble S peut avoir toute sorte de représentation tel une courbe, une surface ou un volume.



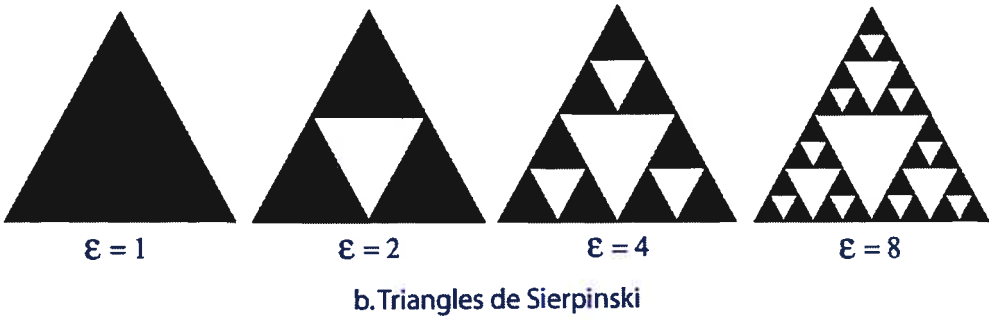


FIG. 2. Exemples de différents types de objets fractals simples : a. le carré, b. le triangle de Sierpinski

1.2.2. L'analyse multifractale

Au-delà des modèles mathématiques permettant d'appréhender l'invariance d'échelle, s'est développé une famille de techniques d'analyses rassemblées sous le vocable d'*analyse multifractale.*

L'analyse multifractale s'applique à des objets quelconques pour en décrire et analyser les variations, les fluctuations de régularité locale. Ce changement récent de paradigme, qui consiste à s'intéresser à des *méthodes fractales* plutôt qu'à des *objets fractals* a joué un rôle crucial dans le développement l'analyse multifractale et est en partie à l'origine de la réussite de l'application du domaine [14]. Elle consiste principalement à décrire, d'un point de vue statistique ou géométrique, la répartition de l'exposant de Hölder du signal (décrivant la rugosité de ce signal). Il existe trois types de spectres multifractals :
- le spectre de Hausdorff,
- le spectre de grandes déviations,
- le spectre de Legendre [17].

Nous nous intéresserons plus particulièrement au spectre de Hausdorff qui fournit la caractérisation géométrique de la répartition des singularités[9] du signal.

Le formalisme multifractal introduit originalement pour l'étude statistique des propriétés d'échelle de mesures singulières a été élaboré par Arnéodo et al. dans le cadre des transformées en ondelette (voir section 1.3) [6].

1.2.3. Mesures multifractales et spectre des singularités

Depuis les travaux de Mandelbrot sur la distribution spatiale de régions dissipatives dans un flux turbulent, le concept de mesure multifractale s'est révélé très fécond pour la modélisation d'objets singuliers présente dans une grande variété de situation physique. Mais commençons par expliquer les termes de mesure et de multifractalité ainsi que celui de singularité.

Le terme de multifractal et le concept de singularité ont été mis en avant par Frisch et Parisi en 1985 dans le contexte de turbulence pleinement développé, et fût formalisé par Halsey et al. en 1986 à travers le cadre des systèmes dynamiques [9].

Dans la description micro-canonique du formalisme multifractal, le caractère singulier d'une mesure[10] μ, en un point x_o de son support est exprimé par un exposant de singularité $\alpha(x_o)$ défini lui-même par le comportement en loi de puissance de la mesure d'une boule $B_{x_o}(\epsilon)$ de taille ϵ et centrée en x_o :

$$\mu(B_{x_o}(\epsilon)) = \int_{B_{x_o}(\epsilon)} d\mu(x_o) \sim \epsilon^{\alpha(x_o)}, \tag{36}$$

pour $\epsilon \rightarrow 0^+$.

L'exposant de singularité $\alpha(x_o)$ en un point x_o rend donc compte du degré local de régularité de la mesure considérée. Notons que dans $\mathbb{R}$, $B_{x_o}(\epsilon)$ est un intervalle de longueur ϵ.

De la même manière que la dimension de Hausdorff est utilisée pour décrire les ensembles dans lesquels la mesure de Lebesgue est nulle (dans $\mathbb{R}$) et qui sont composés de points non-isolés, le spectre de singularité $f(\alpha)$ donne la caractérisation de mesures singulières qui n'ont pas de composante de densité $\rho(x) =$

[9] Une singularité d'une fonction est définie comme le maxima local d'une fonction [10].

[10] Dans le traitement d'images, ainsi que dans divers autres domaines, nous travaillons avec des mesures μ plutôt qu'avec des fonctions. Une mesure implique une notion de résolution ainsi que celle du support $S_{upp}(\mu)$ [17]. Dans le cas du traitement d'image nous comprendrons donc par mesure μ l'équivalent d'un pixels ou d'une partie de l'image, et par support $S_{upp}(\mu)$ l'image au complet ou un ensemble de mesures μ.

$lim_{\epsilon \to 0^+} \mu([x, x+\epsilon])/\epsilon$ [9].

Le spectre $f(\alpha)$ des singularités associé à la mesure μ est la fonction qui, à tout exposant de singularité α, associe la dimension fractale de l'ensemble des points x_o tels que $\alpha(x_o) = \alpha$:

$$f(\alpha) = d_F(\{x_o \in S_{upp}(\mu); \alpha(x_o) = \alpha\}). \tag{37}$$

Une mesure sera dites **homogène** (ou mono fractale) si son spectre des singularités est concentré en un seul point où $\alpha(x) = \alpha$. Si le support de $f(\alpha)$ est étendu, la mesure n'est pas homogène, la valeur de l'exposant $\alpha(x)$ fluctue d'un point à l'autre du support de μ ; on parle dans ce cas d'une mesure **multifractale**.

Étudions maintenant un outil mathématique qui permet d'estimer le spectre de singularité de signaux multifractals.

1.3. ONDELETTES

Dans cette section nous allons étudier les ondelettes ainsi que leurs propriétés. Nous nous attarderons sur les transformés en ondelettes et l'analyse des singularités qu'elles permettent de faire, justifiant ainsi de leur usage pour la caractérisation de la rugosité de signaux.

Les analyses de Fourier et les transformations de Fourier à fenêtre glissante étant inadaptées à l'analyse locale des fluctuations de signaux combinant plusieurs échelles caractéristiques, Morlet et Grossman donnèrent en 1984 un cadre rigoureux aux concepts d'une décomposition espace-échelle. La méthode d'analyse de données par l'utilisation de transformées en ondelettes fût d'abord utilisée pour l'analyse de données sismiques et de signaux acoustiques, puis appliquée à des domaines aussi vastes que variés tels que l'étude des turbulences ou l'analyse d'images météorologiques satellites.

1.3.1. Constructions des ondelettes et de leurs transformées

La décomposition espace-échelle permet la représentation du signal en terme d'une fonction de position (temps, espace) et d'une fonction d'échelle. Le signal n'est plus décomposé en composantes fréquentielles (décomposition harmonique) mais en combinaisons linéaires de fonctions élémentaires localisées en différents points de l'espace caractérisées par des tailles différentes.

Ces fonctions élémentaires sont toutes construites à l'aide d'une unique fonction mère $\Psi(x)$ avec $x \in \mathbb{R}$ qui répond à certaines propriétés de régularité, localisation et oscillation que nous décrirons plus loin (voir équations 40).

Ces fonctions élémentaires $\Psi_{b,a}$ s'obtiennent donc par dilatation et translation

d'une fonction mère $\Psi(x)$. On obtient ainsi un microscope mathématique permettant l'analyse plus précise du signal étudié.

$$\Psi_{b,a}(x) = \frac{1}{\sqrt{a}}\Psi\left(\frac{x-b}{a}\right) \tag{38}$$

a est un paramètre d'échelle (ou fréquentiel) qui est strictement positif tel que $a \in \mathbb{R}^{*+}$, si sa valeur est supérieure à 1, il s'agit d'une dilatation ; si elle est inférieure à 1, c'est une contraction. Le paramètre b est le terme de positionnement (ou terme temporel) de l'ondelette et est défini et varie dans l'ensemble des réels $\mathbb{R}$.

Dans le cas d'une analyse d'image numérique, a et b ne sont pas continus mais discrétisés sur des grilles de taille 2^n par 2^n avec $n \in \mathbb{R}$ (voir équations 39). C'est cette discrétisation particulière qui définit l'analyse en multi-résolution du signal.

La décomposition espace-échelle d'un signal s est une fonction de 2 variables a (échelle) et b (temps) qui définit la convolution du signal s avec l'ondelette $\Psi_{b,a}(t)$ $(t \in \mathbb{R})$, avec [6] :

$$\begin{aligned} &a > 1; \\ &b \in \mathbb{R}^{+}. \end{aligned} \tag{39}$$

L'ondelette analysatrice Ψ est choisie de manière à ce qu'elle respecte certaines contraintes telles que :

$$\int_{-\infty}^{+\infty} \Psi(x)dx = 0; \tag{40}$$

$$\|\Psi_{b,a}\| = 1. \tag{41}$$

Il existe plusieurs autres types d'ondelettes[11], tels que les ondelettes de Grossmann-Morlet qui sont décrites plus haut (voir équation 38).

L'ondelette analysatrice Ψ que nous utiliserons ici pour l'obtention des résultats, est la dérivée seconde d'une gaussienne (le *chapeau mexicain*, voir figure 3) qui

[11] Les ondelettes discrètes d'Ingrid Daubechies, correspondant, comme celles de Grossmann-Morlet, à un algorithme temps-échelle, sont de la forme :

$$2^{j/2}\Psi(2^j x - k), j \in \mathbb{Z}, k \in \mathbb{Z}. \tag{42}$$

Mais les ondelettes de Gabor-Malvar, qui correspondent à un algorithme temps-fréquence, s'expriment comme suit :

$$w(x-l)cos[\pi(k+1/2)(x-l)], l \in \mathbb{Z}, k \in \mathbb{N}. \tag{43}$$

répond aux contraintes décrites plus haut comme, d'ailleurs, toutes les dérivées de la fonction Gaussienne :

$$\Psi^{(N)}(x) = \frac{d^N}{dx^N} e^{-x^2/2}, \tag{44}$$

avec $n_\Psi = N$ où n_Ψ est le nombre de moments nuls de la fonction Ψ [**6**].

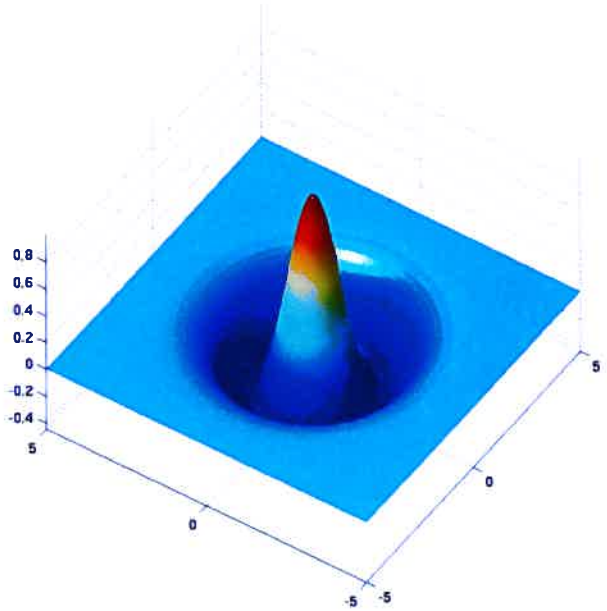


FIG. 3. Ondelette analysatrice : le chapeau mexicain

La transformée en ondelettes $T_\Psi[s](b,a)$ d'un signal $s(x)$, avec $x \in \mathbb{R}$, à une échelle a et pour une position b peut alors être définie tel que :

$$T_\Psi[s](b,a) = \frac{1}{a} \int_{-\infty}^{+\infty} \overline{\Psi}\left(\frac{x-b}{a}\right) s(x)dx, \tag{45}$$

avec $\overline{\Psi}$ le complexe conjugué de Ψ, $a \in \mathbb{R}^{*+}$ et $b \in \mathbb{R}$ [**6**].

1.3.2. Transformation en ondelettes et analyse des singularités

Au début des années 90, Stéphane Mallat et Wen Liang Hwang furent parmi les pionniers dans l'utilisation des transformées en ondelettes pour l'analyse de singularité dans divers signaux. Nous allons, dans cette section, poser les bases et les connaissances nécessaires pour la compréhension d'une analyse de singularité, telles que la quantification de la force de la singularité d'un signal s et l'analyse de sa rugosité [**11**].

Nous avons vu précédemment que le comportement singulier d'un signal $s(x)$ en un point x_o est caractérisé par un exposant h appelé exposant de Hölder quantifiant la *force* de la singularité d'une fonction $s(x)$, ou plus simplement l'irrégularité du signal s (voir équation 30). Ainsi plus la valeur de h tend vers l'unité et plus le signal est régulier.

On peut aussi expliquer le coefficient de Hölder dans le cadre des transformés en ondelettes de la manière suivante. On définit le nombre N tel que les N premiers termes du développement en série de Taylor de $s(x)$ sont annulés par la transformation en ondelettes de $s(x)$ d'ordre N. Le comportement de $s(x)$ autour de $x = x_o$ est caractérisé par :

$$s(x) = c_0 + c_1(x - x_o) + ... + c_N(x - x_o)^N + C|x - x_o|^{h(x_o)} \tag{46}$$

Dans le cas où $h(x_o) > N$:

$$|T_\Psi[s](x_o, a)| \backsim a^N \ (a \longrightarrow 0^+) \tag{47}$$

Dans de telles conditions, l'exposant N est mesuré à la place de l'exposant $h(\mathbf{x}_o)$.

Les coefficients des transformées en ondelettes $|T_\Psi[f](b, a)|$ permettent de mesurer la force du signal s et de ses dérivées de manière globale mais aussi locale. Par localement, nous comprenons un intervalle ouvert ou ponctuel.

Le calcul montre que la transformée en ondelettes d'un signal s au point $b = \mathbf{x}_o$ se comporte en loi de puissance en fonction de l'échelle a avec l'exposant de rugosité locale $h(\mathbf{x}_o)$:

$$|T_\Psi[s](\mathbf{x}_o, a_o)| \backsim a^{h(\mathbf{x}_o)} \ (a_o \longrightarrow 0^+) \tag{48}$$

Pour $n_\Psi > h(x_o)$, la fonction Psi doit être ainsi orthogonale à des polynômes d'ordre faible avec n_Ψ le nombre de moment nul de la fonction Ψ (pour tout m, $0 \leq m < n_\Psi$), [6] :

$$\int_{-\infty}^{+\infty} x^m \Psi(x) dx = 0. \tag{49}$$

1.3.3. Maxima locaux des transformées en ondelettes

Les maxima locaux détectent donc les singularités du signal à l'échelle a. On observe aussi que ces maxima sont organisés selon des lignes et que chacune de ces lignes converge vers une singularité en $\mathbf{x}_o$ du signal lorsque a tend vers 0. On peut ainsi estimer le coefficient de Hölder $h(x_o)$ à partir du comportement du module de la transformée en ondelettes le long des lignes de maxima ; soit la

pente de la courbe $\log_2(|T_\Psi[s](\mathbf{x}_o, a)|)$ en fonction de $\log_2(a)$.

Dans le cas d'un signal s de deux dimensions où l'on ferait plusieurs analyses d'un même point par les ondelettes Ψ_1 et Ψ_2, il est important de noter que cette loi de puissance se retrouve dans le module $M_\Psi[s](\mathbf{x}_o, a_o)$ des transformées en ondelettes sous la forme :

$$M_\Psi[s](\mathbf{x}_o, a_o) = (T_{\Psi_1}[s](\mathbf{x}_o, a_o))^2 + (T_{\Psi_2}[s](\mathbf{x}_o, a_o))^2)^{1/2} \backsim a^{h(\mathbf{x}_o)} \ (a_o \longrightarrow 0^+). \quad (50)$$

Avec, par exemple, pour $x \in \mathbb{R}$ et $y \in \mathbb{R}$:

$$\Psi_1(x, y) = \frac{\partial \phi(x, y)}{\partial x} \quad (51)$$

$$\Psi_2(x, y) = \frac{\partial \phi(x, y)}{\partial y} \quad (52)$$

et la fonction ψ étant une fonction lissante, dont les deux principales sont la Gaussienne :

$$\Psi(x, y) = e^{-(x^2+y^2)/2} = e^{-\frac{\mathbf{x}^2}{2}}, \quad (53)$$

et le chapeau mexicain (voir figure 3) :

$$\Psi(x, y) = (2 - \mathbf{x}^2)e^{-\frac{\mathbf{x}^2}{2}}. \quad (54)$$

On appelle donc Module Maximum d'une Transformée en Ondelettes $M_\Psi[s]$, tout point (x_o, a_o) du demi-plan espace-échelle qui correspond à un maximum local du module de $T_\Psi[s](x, a_o)$ considéré comme une fonction de x :

$|T_\Psi[s](x_o, a_o)| > |T_\Psi[s](x, a_o)|$ pour tout x dans le voisinage droit de x_o,
$|T_\Psi[s](x_o, a_o)| \leq |T_\Psi[s](x, a_o)|$ pour tout x dans le voisinage gauche de x_o.
Ainsi, $\partial |M_\Psi[s]| / \partial x(x_o, a_o) = 0$.

C'est à partir du calcul du module des transformées d'ondelettes (équation 50) que l'on commence à poser les bases de la méthode des Maxima des Transformées en Ondelettes (MMTO) pour la détection des coefficients de texture d'un signal.

1.4. La thermodynamique des fractales appliquée aux ondelettes

Dans cette section, nous allons traiter de l'application des principes de multifractalité et d'ondelettes énumérés précédemment dans le cadre plus particulier de l'analyse de texture de signaux par l'emploi de principes thermodynamiques.

Nous commencerons cette section par une explication de la méthode des maxima

des transformées en ondelettes (MMTO) et de ses principes fondateurs. Puis nous convergerons vers des notions plus précises quand à l'application au formalisme multifractal et la MMTO pour la caractérisation de texture dans les signaux unidimensionnels et bi-dimensionnels.

1.4.1. Méthode des Maxima des Transformées en Ondelettes (MMTO)

Nous avons vu dans la section précédente que nous pouvions calculer le coefficient de Hölder d'un signal s à partir du comportement des transformées d'ondelettes de ce même signal à l'intérieur d'un cône $|x - x_o| < Ca$ dans le demi-plan échelle-espace [**22**]).

La Méthode des Maxima des Transformés en Ondelettes (MMTO), introduite par S. Mallat [**11**] et perfectionnée par A. Arnéodo, N. Decoster et al. [**14, 22**], consiste à mettre en évidence les singularités d'un signal par l'étude du comportement des maxima du module des transformées en ondelettes du signal, dans notre cas il s'agit d'une mammographie.

Dans l'approximation des ondelettes continues, Les maxima locaux caractérisant des singularités du signal nous pouvons nous attendre à ce que les maxima à chaque échelle s'alignent sur des *lignes de maxima*. Ces lignes se trouvent être des courbes se reliant dans le demi-plan espace-échelle (voir figures 4 et 5). Ces lignes de maxima sont obtenues par le chaînage des modules des maxima des transformées en ondelettes de la petite vers la grande échelle. On constate que certaines de ces lignes pointent sur les singularités du signal. Le comportement en loi de puissance des modules des maxima le long de la ligne de maxima l est exprimé dans l'équation 50.

L'ensemble de ces lignes de maxima constitue un squelette. Ce squelette contient toute l'information nécessaire pour étudier les variations de rugosité locale du signal. L'approche thermodynamique, élaborée dans le contexte de l'analyse multifractale, peut être utilisée pour extraire cette information à partir du squelette, comme nous le verrons par la suite.

Un autre problème se pose quand à l'utilisation de lignes de maxima dans l'analyse multifractale par ondelettes que nous allons utiliser [**22**]. Puisque l'on parle de lignes de maxima pour l'analyse de signaux unidimensionnels, il serait pertinent d'user de surfaces de maxima dans l'étude d'un signal bidimensionnel. Cependant puisque l'analyse ne se fait pas sur une fonction continue mais sur une image digitale donc un signal discrétisé, les coefficients d'ondelettes fils issues d'un même point x_o de l'image ne seront donc pas identiques, divergeant de plus en plus alors que les échelles tendent vers une image de plus en plus fine. L'approximation et déformation numérique utilisées ici permettent donc l'utilisation des lignes de maxima pour un signal bidimensionnel.

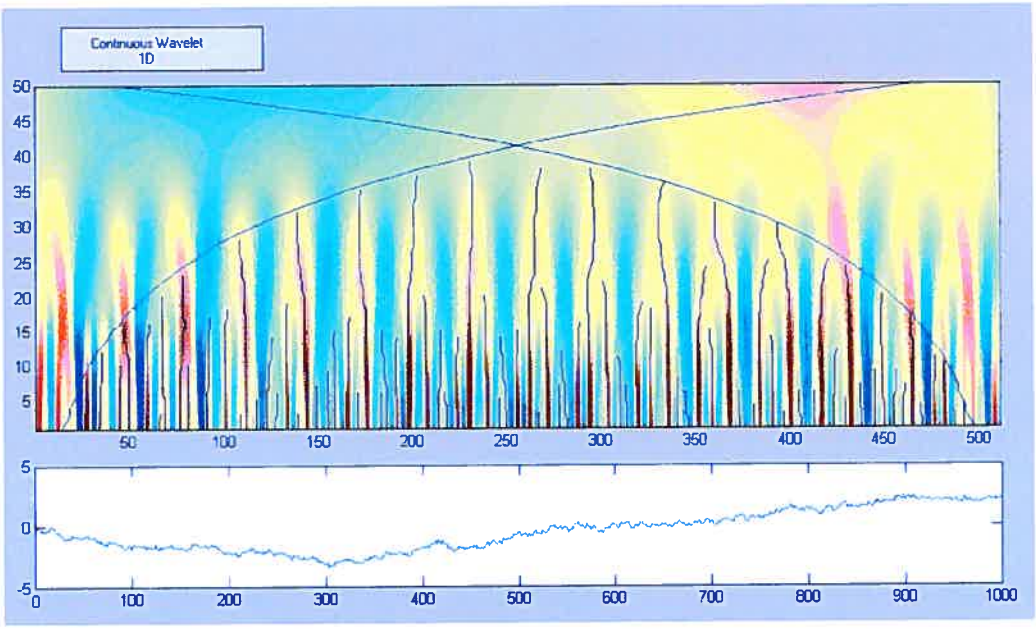


FIG. 4. Lignes de maxima de la transformée en ondelettes continues d'un brownien fractionnaire 1d avec $h = 0.3$ de 512 pixels

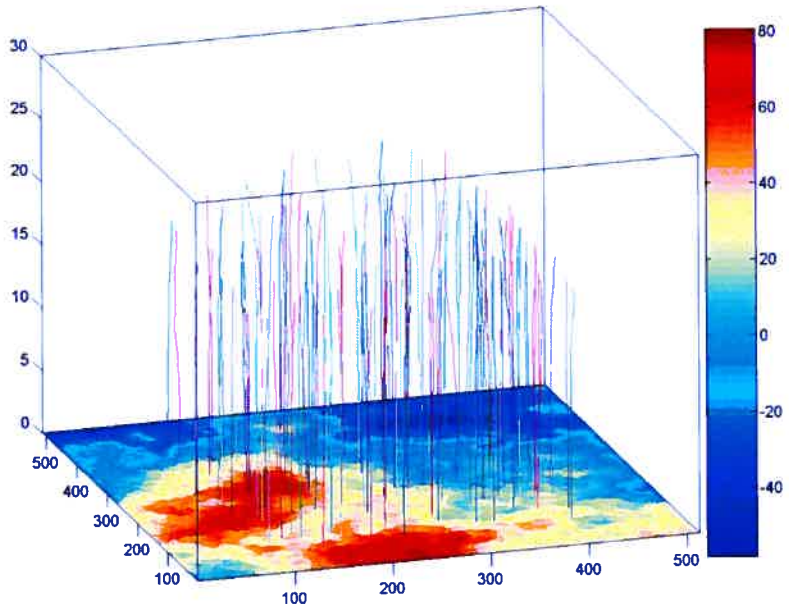


FIG. 5. Lignes de maxima de la transformée en ondelettes continues d'un brownien fractionnaire 2d avec $h = 0.7$ de 512 par 512 pixels

Notons que d'après S. Jaffard, la première limitation pour l'analyse en ondelettes

semble être que des fonctions ayant à chaque échelle les mêmes histogrammes de coefficients d'ondelettes peuvent avoir des spectres de singularité complètement différents ; en analyse multifractale, ce n'est pas seulement l'histogramme des coefficients qui est important, mais aussi leurs positions. C'est pourquoi aucune formule déduisant le spectre des singularités à partir de la connaissance des histogrammes ne peut être valide en toute généralité. On peut cependant espérer que certaines formules soient *plus valides que d'autres* [**18**].

Avant de pouvoir vérifier cette affirmation, finissons d'exposer la méthode d'application à l'imagerie médicale de la MMTO par un rappel sur les bases de thermodynamique et leurs interventions dans la caractérisation de textures en mammographie.

1.4.2. Description thermodynamique des mesures multifractales

Il existe une analogie entre mesures multifractales et les systèmes vus par la mécanique statistique (voir section 1.1). Cette analogie nous permet de simplifier la fonction de partition par l'introduction d'un paramètre q. On peut ainsi identifier le paramètre q à une température ($\beta = 1/k_B T$) (avec k_B le coefficient de Boltzmann et T la température) ce qui permet d'examiner les différentes phases d'auto similarité de la mesure considérée. En faisant varier q, on caractérise successivement les régions du support de la mesure où celui-ci est le plus concentré jusqu'aux régions où il est le plus raréfié.

Si μ est une mesure définie sur une variété (le plan par exemple) dont le support S_{upp} est recouvert de $N(\epsilon)$ boules $B_i(\epsilon)$ de rayon ϵ, la fonction de partition $Z(q,\epsilon)$ (de normalisation de toutes les mesures) prend alors la forme :

$$Z(q,\epsilon) = \sum_{i=1}^{N(\epsilon)} \mu(B_i(\epsilon))^q(\epsilon) \tag{55}$$

En allant chercher la limite thermodynamique d'un volume fini ($V = ln\frac{1}{\epsilon} \rightarrow_{\epsilon \rightarrow 0^+} +\infty$) et en identifiant la force de singularité $\alpha_i = \frac{ln\mu_i}{ln1/\epsilon}$ à une énergie U_j par unité de volume d'un micro-état i, on peut écrire la fonction de partition 55 sous la forme plus habituelle en posant $\beta \sim q$ [**6**] :

$$Z(q,\epsilon) = \sum_i e^{-\beta U_j} \tag{56}$$

On définit le spectre $\tau(q)$ à partir du comportement en loi de puissance de la fonction de partition $Z(q,\epsilon)$ quand le rayon ϵ tend vers 0^+ :

$$Z(q,\epsilon) \backsim \epsilon^{\tau(q)} \tag{57}$$

Ainsi le spectre $\tau(q)$ n'est rien d'autre que l'expression de l'énergie libre $F(\beta)$ par unité de volume.

On peut relier le spectre $\tau(q)$ au spectre des singularités $f(\alpha)$ (voir équation 37). Si l'on considère α comme une variable aléatoire continue de densité (à l'échelle de ϵ) $\rho(\alpha)\epsilon^{-f(\alpha)}$, et puisque α est la variable conjuguée de q alors $Z(q,\epsilon)$ devient :

$$Z(q,\epsilon) = \sum_{i=1}^{N(\epsilon)} \mu_i^q(B_i(\epsilon)) \cong \int \rho(\alpha)\epsilon^{-f(\alpha_i}\epsilon^{-q\alpha_i} d\alpha_i, \tag{58}$$

avec $N(\epsilon)$ le nombre de boules $B(\epsilon)$ de rayon ϵ nécessaires pour recouvrir une surface S.

On peut ainsi définir $\tau(q)$ à la limite ϵ tend vers 0^+ :

$$\tau(q) = min(q\alpha - f(\alpha)) \tag{59}$$

α correspond à l'inverse de la température, alors que $f(\alpha)$ s'identifie à l'énergie libre.

Enfin, cette analogie trouve ses racines dans la théorie des systèmes dynamiques qui permet de décrire les mesures invariantes observées dans certaines situations chaotiques comme des états de Gibbs. Outre son caractère esthétique, le formalisme thermodynamique est intéressant à plusieurs niveaux. Sur le plan mathématique, cette analogie permet de comprendre comment thermodynamique et formalisme multifractal peuvent faire l'objet de la même formalisation rigoureuse. Il permet d'autre part de comprendre les éventuelles non analycités dans le spectre des singularités comme des transitions de phases et donc de faire un lien entre les propriétés *microscopiques* de symétrie de l'objet considéré (invariance d'échelle) et les propriétés *macroscopiques* de régularité des spectres observés.

1.4.3. Application de la M.M.T.O.

L'application à la mammographie de principes mathématiques tels que les fractales prend alors une dimension physique. On ne considère plus la mammographie comme une image mais comme un système physique dont on calcule la fonction de partition, l'entropie, les changements de phases.

Afin d'appliquer ces principes de thermodynamique à la mammographie, considérons à chaque échelle a et sur une ligne de maxima l, les modules des maxima des coefficients d'ondelettes $M(a,l)$ (voir section 1.3.3) définis tels que :

$$M(a,l) = \sup_{\substack{(a',b) \in l \\ a' \leqslant a}} |M(a',b)| \tag{60}$$

Ce processus est en fait une régularisation des lignes de maxima afin de respecter la symétrie entre les échelles telle que le décrit Jaffard afin de toujours obtenir une croissance de la valeur du maxima local (ou tout du moins sa stabilisation). Cette régularisation permet ainsi de ne pas avoir de décroissance de la valeur de l'entropie du système et ainsi enfreindre la seconde loi de la thermodynamique [18].

Cette approximation permet de redéfinir le poids de Boltzmann :

$$W(a,l,q) = \frac{|M(a,l)|^q}{Z(a,q)} \tag{61}$$

Où $Z(a,q)$ est l'énergie libre :

$$Z(a,q) = \sum_{l \in l(a)} |M(a,l)|^q \tag{62}$$

Et ainsi établir le spectre des singularités $D(h)$ ainsi que l'exposant de rugosité local h pour un a fixé :

$$h(a,q) = \sum_{l \in l(a)} W(a,l,q) ln|M(a,l)| \tag{63}$$

(h correspond à une énergie moyenne)

$$D(a,q) = \sum_{l \in l(a)} W(a,l,q) ln(W(a,l,q)) \tag{64}$$

($D(h)$ correspond à une entropie donc toujours ≥ 0)

On pourra ainsi obtenir graphiquement la dimension de Hausdorff en fonction de l'exposant de Hölder et ainsi obtenir le coefficient de texture de l'image observée :

$$\frac{h(a,q)}{ln(a)} \xrightarrow[a \to 0^+]{} h(q) \tag{65}$$

$$\frac{D(a,q)}{ln(a)} \xrightarrow[a \to 0^+]{} D(q) \tag{66}$$

Avant de passer à l'implémentation numérique de ces principes théoriques qui nous permettront la détection de singularités, étudions la physique cachée derrière les mammographies ainsi que la physiologie du sein et les problèmes qui y sont liés. Cela nous permettra d'établir une première carte des différentes textures que nous nous attendrons à discerner lors des analyses des mammographies.

Chapitre 2

APPLICATION À LA MAMMOGRAPHIE

Le but de ce chapitre est de poser les bases anatomiques et physiques qui justifient l'application de l'approche multifractale sur un nombre d'échelles limitées aux mammographies. Nous commencerons cet exposé par une étude succincte de la structure interne d'un sein sain ainsi que son évolution au cours de la vie. Nous pourrons établir quels types de texture et quelles variations structurelles sont dites normales et ainsi voir la robustesse des travaux de Caldwell, Kestener et al sur le parenchyme mammaire. Par la suite, nous nous attarderons sur les diverses pathologies atteignant les glandes mammaires et leurs caractéristiques. Nous poursuivrons par une présentation des diverses techniques de radiographies numériques et leurs influences sur les données stockées sur ordinateur. Nous terminerons cette section par une discussion sur la texture des mammographies et leurs origines multifractales probables.

2.1. Anatomie et physiologie du sein

Dans cette section, nous allons étudier la structure interne de la glande mammaire bénigne et maligne. Cette partie nous permettra, en plus d'établir l'origine anatomique de la texture de la mammographie, de construire une liste non-exostive des pathologies cancéreuses du sein.

2.1.1. Cas d'un sein sain

Commençons par analyser la structure anatomique de la glande mammaire. Nous poursuivrons par son étude au cours du cycle génital, nous permettant de constater la non stabilité de la densité interne du seins.

2.1.1.1. *Structure de base*

La glande mammaire est une glande exocrine[1], tubulo-alvéolaire[2] composée, sécrétant le lait. Ce tissu glandulaire existe aussi chez l'homme et l'enfant mais de manière réduite. On la décrit chez la femme en période d'activité génitale. Le tissu se développe à la puberté et involue à la ménopause (voir figure 1) [**35**].

[1]Glande à sécrétion externe, soit directement en milieu extérieur (par la peau ou par un canal sécréteur) soit au niveau d'une muqueuse.

[2]Composé de petits canaux appelés tubules, ainsi que d'alvéoles.

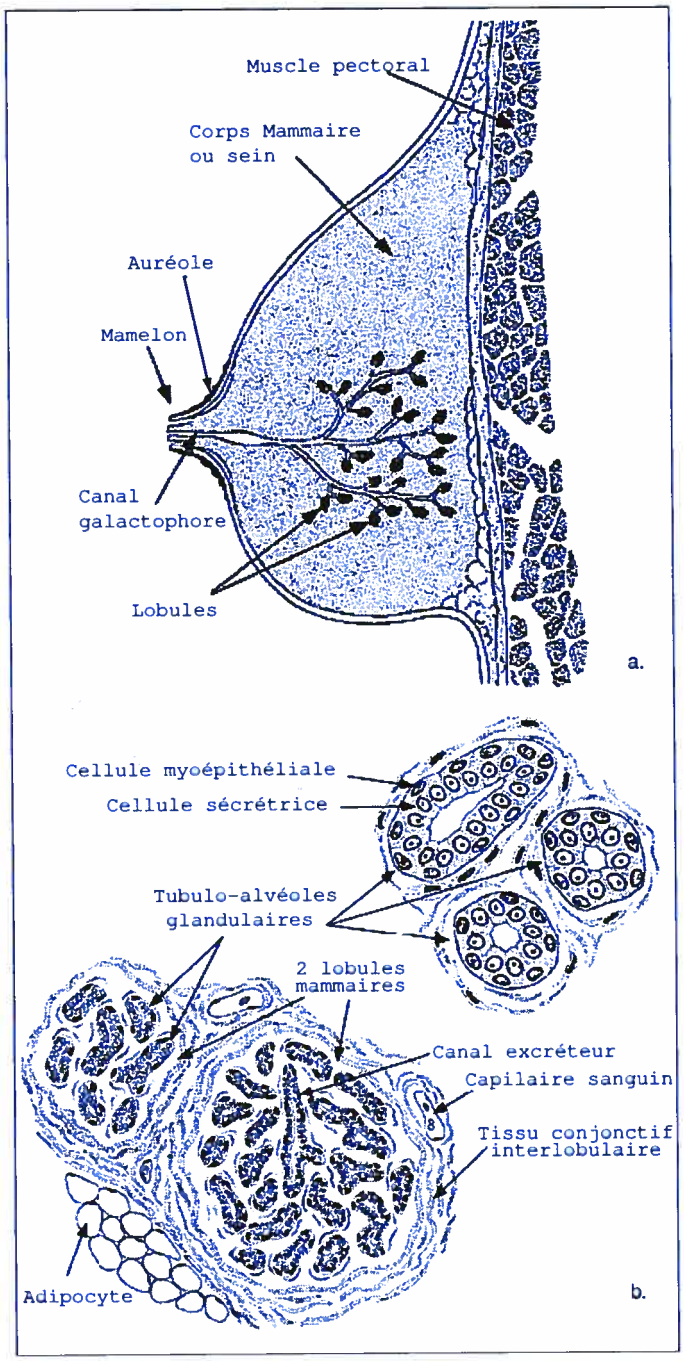


FIG. 1. La glande mammaire : a. Coupe transversale, b. Lobule mammaire [**36**].

La glande est annexée à la paroi antérieure du thorax. Sur le plan profond, nous retrouvons trois muscles, le grand pectoral sur lequel glisse la glande mammaire, le petit pectoral et le sous-clavier. Sur le plan superficiel le *fascia superficialis* s'arrête avant la région du mamelon qui constitue la limite antérieure et

superficielle avec aréole. La limite supérieure est la deuxième ou troisième cote alors que l'extrémité inférieure se trouve sur la sixième ou septième cote.

Le sommet de la glande mammaire est appelé aréole au niveau du quatrième espace intercostal. La partie centrale surélevée de l'aréole est le mamelon où se trouve l'ouverture microscopique de 10 à 20 canaux galactophores qui assurent en cas d'allaitement l'évacuation de sécrétions lactées. Les glandes de Mogani sont la partie grenue de l'aréole. Ce sont des glandes cutanées et sébacées qui s'hypertrophient à la grossesse et prennent alors le nom de tubercules de Montgomery.

Les moyens d'amarre sont le fascia superficialis et les ligaments suspenseurs du sein (travée fibro-glandulaire) dont l'épaississement plus net dans la partie inférieure va tirer la peau en profondeur (sillon sous-mammaire) [**34**].

La glande est constituée de 10 à 20 lobules drainés par des canalicules collecteurs s'ouvrant sur l'aréole. Ces lobules sont eux-mêmes composés de 10 à 100 tubulo-alvéoles. Ces tubulo-alvéoles constituent la partie sécrétrice de la glande et sont entourés de tissu conjonctif dense. Le tissu conjonctif intra-lobulaire (tissu conjonctif palléal) est lui lâche (sa densité varie selon la période du cycle hormonal et l'âge de la personne). Le lobule et le galactophore terminal extralobulaire représentent ce que Wellings surnomma en 1975 le TDLU (Terminal Duct Lobular Unit ou unité terminal ducto-lobulaire). Ce TDLU représente le lieu de développement de la plupart des proliférations épithéliales bénignes et malignes du fait de sa grande hormonoréceptivité [**35**].
En dehors du tissu glandulaire, la glande mammaire est remplie de tissus graisseux.

La vascularisation se fait à l'aide de l'artère subclavière, une branche de l'artère auxiliaire, des rameaux latéraux de la thoracique externe, des rameaux médiaux traversant les espaces intercostaux de l'artère thoracique interne, et des rameaux perforants des artères intercostales en regard de la glande mammaire.

Les veines forment un système parallèle aux artères. Ce système vasculaire forment un cercle anastomotique[3] périmamelonnaire[4] et périaréolaire[5] qui augmente chez les femmes enceintes ainsi que pendant l'allaitement.

[3]Ce dit d'un réseau formé généralement par des artères ou des nerfs reliés sur une certaine longueur. Dans le cas de la glande mammaire, le système vasculaire forme un anneau alimenté par plusieurs artères.

[4]Entourant le mamelon, voir figure 1.

[5]Entourant l'auréole, voir figure 1.

2.1.1.2. *Au cours des étapes de la vie génitale*

Étudions à présent l'évolution des glandes mammaires au cours de la vie génitale [**36**].

Avant la puberté, les tubulo-alvéoles ne sont pas encore développées et le réseau galactophore est encore rudimentaire.

À la puberté, lors des premiers cycles menstruels, l'action des oestrogènes permet le développement des canaux galactophores et est accompagné du développement du tissu conjonctif interlobaire et intralobulaire ainsi que des cellules adipeuses.

En période d'activité génitale, hors période de grossesse quelques tubulo-alvéoles peuvent se développer sous l'effet de la progestérone mais en général ils involuent.

Après la ménopause, une involution lente du réseau galactophore se produit ainsi que des tubulo-alvéoles restants. Les glandes mammaires ont ainsi une tendance à reprendre leur forme pré-pubère.

2.1.1.3. *Pendant la grossesse*

Le but des glandes mammaires est de nourrir et d'assurer les défenses immunitaires du nouveau né par la sécrétion de lait.

Dès les premiers mois de la grossesse, grâce à l'influence des stéroïdes naturels tels que la progestérone, on observe dans les glandes mammaires une prolifération des ramifications terminales du système canaliculaire ainsi que des tubulo-alvéoles glandulaires. La sécrétion de prolactine par le corps provoque la production par les tubulo-alvéoles d'un produit riche en protéine et pauvre en lipide, le colostrum.

L'accroissement de la production de prolactine provoque la lactogènése, ou montée de lait, qui débute dans les jours suivant l'accouchement. On constate alors la transformation des cellules présécrétrices en cellules sécrétrices. La production lactée est alors entre un et deux litres par 24 heures.

2.1.1.4. *Après le sevrage*

La suppression des tétées entraîne l'arrêt de la production réflexe de prolactine ce qui entraîne une involution des glandes mammaires en commençant par une dislocation de l'épithélium mammaire et un démantèlement de la matrice extracellulaire. Au bout de quelques jours, la glande mammaire a retrouvé sa structure de repos sans retrouver sa structure antérieure puisque de nombreux tubulo-alvéoles produits lors de la grossesse ne disparaissent pas entièrement.

La figure 2 nous montre l'évolution temporelle de la structure interne des glandes

mammaires par une coupe transversale. La figure 2.a nous montre une glande mammaire au moment de la grossesse, 2.b au repos et 2.c après la ménopause.

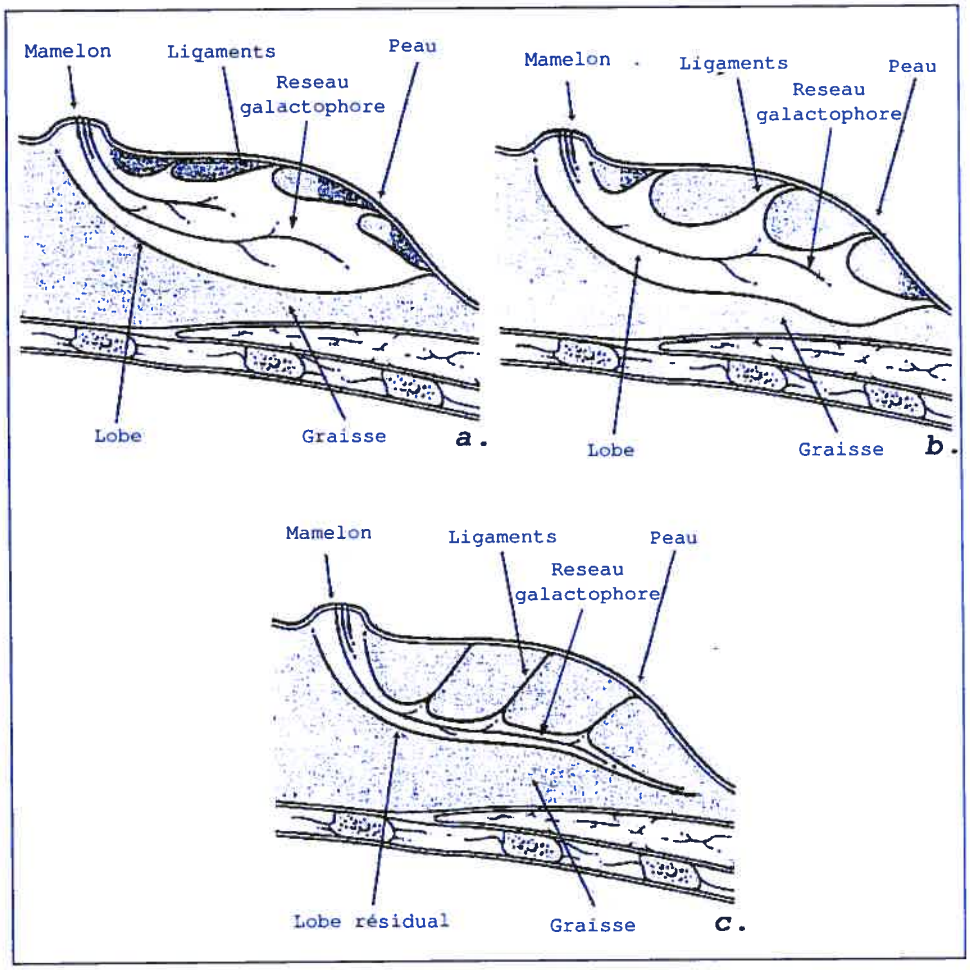


FIG. 2. Evolution temporelle de la glande mammaire (coupes transversales)

Nous pouvons déjà constater que la structure mammaire n'est pas une structure stable et définitive, comme tout le reste du corps humain. Elle est en constante évolution, que ce soit au cours du cycle menstruel, pendant la grossesse et après le sevrage ou encore au moment de la ménopause. Cette évolution étant différente pour chaque individu, il nous sera donc peu probable de prévoir une densité et une *homogénéité* moyenne typique de la glande mammaire. Cette variété oblige donc la présence d'un spécialiste pour l'analyse des résultats de l'analyse des textures composant la mammographie.

2.1.2. Cas d'un sein malade

Un certain nombre de pathologies peuvent se développer dans les glandes mammaires. Attardons nous quelques instants sur les structures bénignes et malignes qui peuvent se développer et apparaître sur une mammographie. En comprenant mieux la composition histologique de ces pathologies nous pourrons ainsi en apprendre plus sur la texture de ces corps étrangers et la manière de les détecter plus efficacement **[35]**.

2.1.2.1. *Les états fibrokystiques*

Les états fibrokystiques sont des regroupements de plusieurs lésions élémentaires dont l'association à degrés divers leurs confère un aspect très hétérogène. Ils sont caractérisés par un remodelage des TDLU[6] lié à un déséquilibre hormonal entre les structures épithéliales et conjonctives spécialisées telles que le manteau. Ils sont généralement de nature bénigne.

Ces lésions élémentaires sont de forme assez variée. Ainsi on peut retrouver, dans les états fibrokystiques, des kystes de tailles variables qui sont multiples et regroupés. La fibrose, lorsqu'elle est importante, se présente comme un placard blanc ponctué de kystes qui apparaît ferme à la palpation. L'adénose consiste en une multiplication des ductules terminaux à partir des lobules ou des petits galactophores. Il est très fréquent d'observer des calcifications dans l'adénose qui sont situées dans la lumière des tubes et ne dépassent pas les 100 microns.

Les statistiques montrent que les états fibrokystiques se développent dans la tranche d'âge allant de 40 à 50 ans, et sont plutôt rares avant 25 ans et après la ménopause. Ils sont généralement bilatéraux mais peuvent aussi se développer unilatéralement. Ils peuvent être localisés et former une masse palpable ou être diffus et masquer de petits cancers infiltrants.

2.1.2.2. *Pathologie inflammatoire*

Dans l'ensemble des pathologies inflammatoires, une inflammation aseptique intéressante est l'adiponécrose. Elle peut être post-traumatique ou se développer sans raison traumatique apparente après la ménopause en raison de problèmes circulatoires locaux parfois rencontrés à cet âge. La lyse du tissu adipeux suscite une réaction inflammatoire résorbative qui se complique de fibroses rétractiles pouvant imiter un cancer infiltrant.

2.1.2.3. *Tumeurs bénignes du sein*

Les tumeurs bénignes sont des lésions généralement bien circonscrites qui ne récidivent pas après exérèse et qui par définition ne métastasent pas. Certaines

[6]Terminal Duct Lobular Unit ou unité terminal ducto-lobulaire.

de ces tumeurs bénignes peuvent imiter le cancer. C'est le cas des fibroadénomes géants, des adénomatoses du mamelon et des tumeurs à cellules granuleuses.

2.1.2.4. *Les cancers du sein*

On appelle cancer toute lésion des cellules ou des tissus résultant d'une prolifération anormale de ces éléments aboutissant généralement à une extension et une diffusion en dehors des zones où ils sont normalement situés.

Le cancer du sein est le cancer le plus fréquent chez les femmes occidentales. Seulement 1% des cancers sont des cancers du sein chez l'homme dans la tranche d'âge allant de 35 à 70 ans.

Dans 98% des cas, le cancer du sein est un adénocarcinome se développant à partir de l'épithélium glandulaire, à la différence des rares sarcomes qui se propagent à l'intérieur des tissus de soutien.

Le carcinome se caractérise par sa capacité à se développer au même endroit après exérèse et à métastaser. L'aspect macroscopique des carcinomes mammaires infiltrants dépend du rapport qualificatif et quantitatif que les cellules cancéreuses entretiennent avec le tissu de soutien intra-tumoral appelé stroma. Certains carcinomes, appelés squirrhes, ont un stroma abondant, dense et souvent rétractile apparaissant à la radiographie sous un aspect dense et stellaire. En revanche, d'autres carcinomes ont une consistance molle et apparaissent de la même manière que des tumeurs bénignes [**35**]. Il existe ainsi un large panel de formes histologiques de carcinomes tels que le carcinome lobulaire in situ, lobulaire infiltrant, galactophorique in situ ou galactophorique infiltrant.

Vu les statistiques précédentes et la taille de notre base de donnée, nous nous attendrons donc à observer, dans la majeure partie des cas malin, un adénome kystique. Possédant maintenant une connaissance de base suffisante sur la forme structurelle bénigne et maligne des glandes mammaires, passons maintenant à la dernière étape permettant de visionner une projection en deux dimensions de ces structures, La mammographie par rayon X. Cette prochaine section nous permettra donc d'avoir tous les éléments permettant de pouvoir juger adéquatement de la texture des mammographies digitales que nous allons étudier par la suite.

2.2. La mammographie

Le but de cette section est d'approfondir nos connaissances en matière de formation d'images radiologiques digitales. Cela nous permettra, entre autre, de mieux pouvoir estimer la notion de projections bidimensionnelles d'un objet complexe en trois dimensions et la discrétisation de la texture de cet objet dans un format numérique.

Nous commencerons donc par faire un rappel succinct sur les principes physiques régissant la radiographie par rayons X. Puis nous étudierons les diverses méthodes utilisées pour numériser l'interaction de ces rayons avec le détecteur après leur passage dans des structures complexes. Nous finirons par une présentation des divers modes de stockage numérique de la radiographie digitale obtenue.

2.2.1. Principes de radiographie

La radiographie a vu le jour en 1895 grâce à Wihelm Conrad Roentgen qui observa les os formant la main de sa femme par fluorescence d'un écran de platinocyanure de baryum après son exposition aux rayons cathodiques émis par un tube de Crookes. Depuis cette époque, beaucoup de progrès dans la compréhension du phénomène de création de rayon X ont permis le développement de la radiographie aux rayons X telle que nous la connaissons actuellement.

Résumons rapidement le modèle de formation d'image d'un système radiographique élémentaire. Commençons par la production des rayons X. Les trois parties principales formant le tube à rayon X sont l'anode, la cathode et le tube. Le filament de la cathode est chauffé à l'aide d'un courant électrique éjectant ainsi des électrons qui sont dirigés vers l'anode dû à l'application d'une différence de potentiel entre l'anode et la cathode (le potentiel variant de quelques à 450 kilovolts)(voir figure 3).

Généralement, une coupole en argent est utilisée pour concentrer le flux d'électrons dans la direction de l'anode. L'anode en tungstène, frappée par les électrons, émet alors une radiation par deux phénomènes distincts, le spectre de raie caractéristique de l'élément composant la cible et un fond continu de rayonnement appelé aussi rayonnement de Bremstrahlung avec une efficacité de 1% (lorsque l'électron passe dans les couches K et L de l'atome de la cible, un photon X est émis. Son énergie est indépendante de la nature de l'atome et du noyau). Les 99% d'énergie restante provenant des électrons sont convertis en chaleur à la surface de la cible en tungstène, la chaleur est ensuite dissipée par le support en cuivre, le bain d'huile dans lequel il repose et le circuit de refroidissement entourant. Le verre épais à base de plomb que constitue le tube sert à prévenir l'émission de radiation de faible intensité et d'exposer inutilement le patient. Il existe différentes sortes de tubes en fonction de leur application tels que le tube panoramique, le tube à longue anode et le tube à anode tournante (utilisé plus couramment en médecine car il possède deux foyers, un étendu pour radiographier de larges surfaces et un quasi-ponctuel pour des zones plus précises).

Les rayons X sortent du tube par une fenêtre pratiquée dans l'enceinte plombée qui entoure le tube, ce qui limite l'émission non isotrope du rayonnement à un cône homogène de 30 à 45 degrés d'ouverture. Les ondes électromagnétiques

n'étant déviées que par des champs magnétiques, les rayons X se propagent en ligne droite à travers l'objet à radiographier. L'intensité du rayonnement est alors atténuée selon la loi $N_o = N_i e^{-\mu x}$ où N_i est le nombre de photons incidents, N_o le nombre de photons restant après un passage de longueur x dans une matière dont le coefficient d'atténuation est μ, ce dernier dépend de la densité de la matière et de sa composition nucléaire.

Le rayonnement peut interagir avec la matière en fonction de l'intensité du rayonnement comme la diffusion Rayleigh, l'effet Compton (entre $1MeV$ et $4MeV$), de création de paire ($> 1.022MeV$) et dépendant aussi de la masse atomique Z de la matière traversée sous la forme d'effet photoélectrique (proportionnel à Z^5).

Le rayonnement atténué est alors détecté de deux manières différentes. Soit par un procédé chimique en activant la fluorescence d'une plaque traitée à cet effet (qui sera fixée sur une plaque photographique ou amplifiée pour un digitalisation de ce signal) ou électroniquement par un détecteur digital à base de Silicium amorphe ou de Sélénium amorphe, comme nous allons l'étudier dans la prochaine section. De manière générale, on peut dire que le radiologue distingue globalement cinq sortes d'opacité sur une radiographie, le métal, l'os, l'eau, la graisse et l'air.

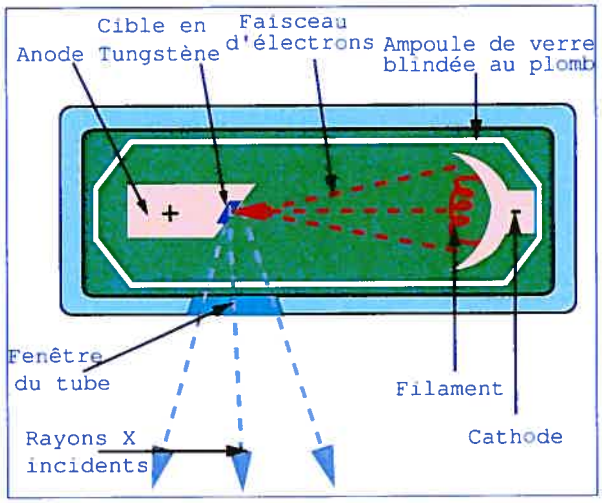


FIG. 3. Ampoule de production de rayons X

2.2.2. Mammographie analogique-digitale

Il existe diverses méthodes de visualisation d'une radiographie [**20**]. Nous concentrerons notre exposé sur les techniques permettant l'obtention d'un fichier

numérique. Nous commencerons donc par les procédés dits indirects ou analogiques/digitaux pour évoluer lentement vers les dernières découvertes permettant la formation directe de fichiers numériques traitables par un ordinateur.

2.2.2.1. *Digitalisation manuelle de radiographie (méthode indirecte)*

La première technique permettant l'acquisition de radiographies numériques consistait à digitaliser à l'aide d'un scanner numérique une radiographie obtenue de manière traditionnelle.

Dans la technique traditionnelle, les photons sortant de l'objet à radiographier passent à travers un scintillateur changeant la longueur d'onde des photons dans la bande spectrale du visible avant le photomultiplicateur ou la fixation sur un film radiographique.

Un scintillateur utilise le phénomène de fluorescence, ce qui signifie que la particule que l'on veut détecter va exciter des atomes ou des molécules dans le scintillateur qui ensuite va émettre de la lumière. Un bon scintillateur doit répondre à plusieurs caractéristiques [**19**] :
- il doit pouvoir générer, pour une particule incidente donnée, suffisamment de lumière pour qu'elle puisse être collectée
- il doit être transparent à sa propre lumière, pour éviter les pertes
- la lumière doit être émise suffisamment rapidement après le passage de la particule
- son indice de réfraction doit être proche de celui des photocathodes des PM, et la longueur d'onde de la lumière émise doit être la plus proche possible de celle du maximum d'efficacité des photomultiplicateurs
- il doit pouvoir être facilement usinable et d'utilisation aisée Le matériel qui répondrait parfaitement à ces cinq qualités n'a pas encore été découvert, on doit donc généralement faire des choix.

Il existe deux grandes classes de scintillateur, les scintillateurs organiques et les scintillateurs inorganiques, toutes deux connues depuis la fin des années 40

Les scintillateurs inorganiques sont le plus souvent des cristaux semi-conducteurs ou isolants. La particule incidente va désorganiser légèrement le cristal et provoquer des excitations des électrons, qui en se désexcitant vont émettre de la lumière. Plus l'énergie d'excitation de ces électrons sera faible et plus, pour une énergie incidente donnée, on aura beaucoup d'électrons donc beaucoup d'excitation et finalement beaucoup de lumière. On quantifie ce phénomène par le nombre de photons émis pour une énergie donnée[7].

[7]Pour améliorer cette caractéristique on rajoute souvent un dopant dans le cristal, par exemple du Thallium pour les cristaux de NaI. De plus les photons émis par la désexcitation des électrons étant moins énergétiques ils vont se rapprocher du domaine visible et être plus

Les différents types de scintillateurs couvrent toute la gamme de détecteurs dont on peut avoir besoin en physique nucléaire. Ils sont aussi bien utilisés pour détecter des ions que des gammas ou des neutrons. Pour les rayons X, on utilise plus communément les scintillateurs inorganiques.

Le but d'un détecteur à base de scintillateurs est de convertir l'énergie qu'une particule incidente laissera dans le scintillateur en lumière visible par un photomultiplicateur ou par le film radiographique.

Le film radiographique est un film photographique spécial muni d'une forte épaisseur d'émulsion sensible, très chargée en halogénure d'argent. Il est généralement bicouche (une demi-émulsion sur chaque face) pour faciliter le développement de l'image et accélérer la prise de mesure afin de diminuer la dose de rayons X à laquelle le patient est exposé. Plus l'énergie du rayonnement est élevée, plus l'oxydation des sels métalliques contenus dans l'émulsion photographique est importante, et le noircissement de la pellicule important. Les photons ont de rares interactions, mais celles-ci produisent des électrons qui sont par contre très actifs pour le noircissement du film. On utilise souvent des films emballés dans une pochette en papier étanche à la lumière ou placés dans une cassette en alliage léger (transparente aux rayons X), munie d'écrans renforçateurs ou au plomb, ou encore dans un emballage plastique sous vide avec des écrans au plomb afin de conserver les films hors de l'humidité et de la lumière. Les écrans renforçateurs (dits salins ou fluorométalliques) comportent un matériau fluorescent qui réduit la résolution spatiale de l'image, mais raccourcit énormément la durée d'exposition nécessaire (jusqu'à 24 fois). Les écrans au plomb sont faits d'une feuille de carton recouverte d'une fine épaisseur de plomb qui atténue l'effet des rayons diffusés et renforce l'image au dessus de 100 kV par émission d'électrons (le gain sur la durée d'exposition est d'environ 3 fois).

Les systèmes indirects sont basés sur le principe de lumière sur film pour créer une image, ce qui nécessite plusieurs étapes :
- les photons sortent de la structure à étudier, le matériau phosphorescent des écrans intensificateurs absorbe l'énergie des rayons X incidents et devient fluorescent durant l'exposition.
- La lumière émise expose l'émulsion du film créant ainsi l'image latente.
- L'image latente est rendue visible par le développement du film (développeur et

---

proches des gammes d'entrée des photomultiplicateurs. Pour rentrer plus dans le détail, les cristaux semi-conducteurs ont un gap entre la bande de valence et la bande de conduction. Lorsque l'on dope ces cristaux on peut voir apparaître, au niveau des impuretés, des bandes d'énergie dans le gap, et quand un électron excité se trouve sur la bande de conduction tombe sur un de ces sites, il y est piégé puis décroît vers la bande de valence en émettant la lumière voulue

fixateur).

Une fois que nous avons obtenu cette image radiographique, il nous suffit de la numériser avec un scanner permettant l'obtention d'image dans un format compatible aux normes DICOM (Digital Imaging and Communications in Medicine).

2.2.2.2. *Caméra CCD (Charge Coupled Devices)(méthode indirecte)*

Une autre technique de radiographie numérique indirecte consiste à utiliser une caméra CCD. Le principe est basé sur celui du précédent. Les rayons X incidents sortent de l'objet pour entrer dans le scintillateur généralement composé d'oxysulfure de gadolinium. Les rayons lumineux sortant sont concentrés par des optiques (lentilles, fibres, etc.) pour entrer dans la caméra CCD pour digitalisation [**20**].

Le CCD est utilisé comme composante pour l'acquisition d'image dans des appareils photographiques digitaux et caméra vidéo. Les systèmes digitaux CCD de radiographie utilisent ce dispositif à la fois pour minimiser les optiques permettant de visualiser la lumière émise par le scintillateur, mais aussi comme écran intensificateur pour la visualisation directe de films radiographiques.
Le principal avantage des CCD est leur petite taille (entre 2 et 3 cm^2) ce qui est généralement plus petit que la zone d'intérêt. On utilise généralement de la fibre optique pour acheminer la lumière sortant du scintillateur jusqu'aux CCD. L'intensité de la lumière parvenant aux CCD est généralement plus faible que celle issue du scintillateur ce qui entraîne une perte de définition de l'image. La technologie des CCD permet donc la numérisation d'une image radiographique à un prix très abordable au coût d'une qualité moindre de l'image qu'elle produit.

2.2.2.3. *Détecteurs plats (méthode indirecte)*

La technique des détecteurs plats utilise des scintillateurs au Iodure de césium ou Oxysulfure de Gadolinium à la sortie desquels les photons rentrent dans des photodiodes en silicium ou silicium amorphe.

Les photodiodes sont des jonctions p-n créées spécifiquement pour optimiser leur photosensibilité inhérente. Elles peuvent être utilisées de deux manières, soit dans un rôle photovoltaïque, soit dans un rôle photoconducteur. Ici, on utilise leur capacité à devenir une source de courant lorsqu'elles sont illuminées (photovoltaïque). Les photodiodes en silicium sont construites à partir d'un même et unique cristal (wafers) que ceux utilisés dans les circuits intégrés, mais de pureté supérieure.
Lorsque la lumière est absorbée par la partie active une paire électron-trou se forme alors. Les électrons et les trous se séparent alors, les électrons passant dans la région n et les trous dans la région p. De ceci résulte un courant généré par la lumière. La migration des trous et des électrons dans leurs zones respectives est appelée *effet photovoltaïque.*

Le courant nouvellement généré par la photodiode est alors transmis à une plaque de transistors fins (ou Thin Film Transistor TFT en anglais).
Le TFT est un interrupteur électronique communément constitué de silicium amorphe. Il permet aux charges collectées à chaque pixel (un transistor correspond à un pixel d'image, c'est-à-dire au plus petit élément que compose une image digitalisée) d'être transférées indépendamment les uns des autres à une électronique extérieure où elles seront amplifiées et quantifiées et dont l'ensemble formera l'image radiologique digitalisée escomptée.

2.2.2.4. *Détecteurs plats (méthode directe)*

Le système de conversion directe des rayons X en format numérique est composé d'une fine couche de photoconducteurs sensible aux rayons X adjacente au TFT et aux capacitances de stockage de charges.

Du sélénium amorphe est utilisé comme matériau de photoconduction pour ses excellentes propriétés de détection des rayons X et pour son extrême résolution spatiale. Avant l'exposition, un champ électrique est appliqué à travers la couche de sélénium amorphe à l'aide d'une électrode recouvrant la surface supérieure de cette couche. À chaque absorption de rayon X, des charges électriques sont tirées le long des lignes de champ jusqu'aux électrodes des capacités de stockage. Les charges collectées sont alors magnifiées et quantifiées en pixels pour ainsi former l'image digitale de la radiographie.

L'usage d'un détecteur direct signifie donc :
- Les rayons X sortent de la structure à radiographier et sont capturés directement sous forme de signaux digitaux par les détecteurs, puis stockés dans un ordinateur pour traitement (visualisation, impression, etc.).
- Pas de phosphore.
- Pas de scintillateur.
- Pas d'étape intermédiaire.
- Pas de lumière à diffuser ou à amplifier.
- Seul le bruit du détecteur dégrade les signaux après l'arrivé des rayons X sur le détecteur.

Cette méthode de formation d'images radiographiques n'a pas le défaut d'empattement du signal de sortie des autres techniques analogiques (voir figure 4), mais a l'inconvénient de la précision due à la capacité limitée de stockage de charge.

Les différents types de signaux de sortie sont résumés dans le graphique 4. Malgré le fait que l'oeil humain ne distingue pas la différence de précision entre une radiographie numérique prise à l'aide de caméra CCD et une radiographie prise

par une matrice de détecteurs plats à sortie numérique directe, il est intéressant de constater qu'un ordinateur permet une résolution visuelle au moins quatre fois supérieure à celle de l'homme, lui permettant de mieux l'assister lors des diagnostics médicaux.

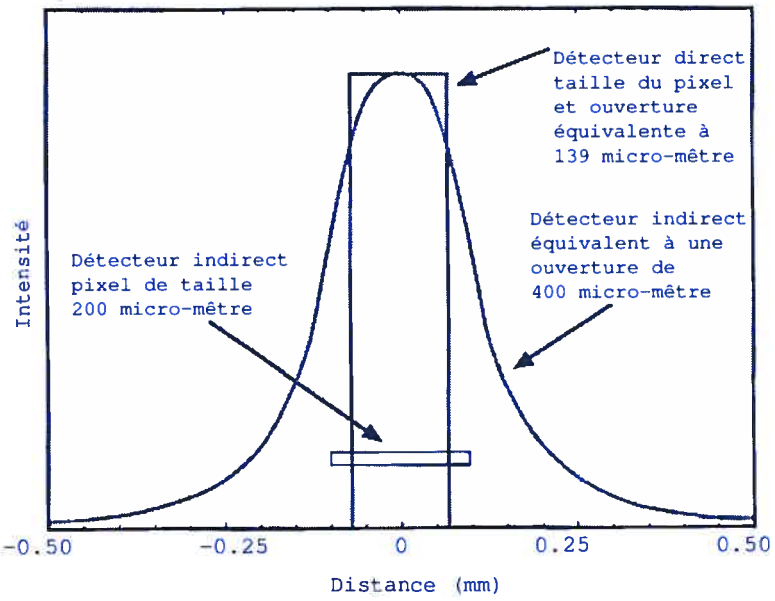


FIG. 4. Précision du signal de sortie des détecteurs radiographiques digitaux direct et indirect [**20**]

Puisque nous avançons dans un monde où toutes les informations semblent de plus en plus stockées et traitées par les ordinateurs, prenons quelques instants pour nous concentrer sur les formats de stockage et de transfert des informations médicales afin de connaître un peu mieux la structure des données que nous allons exploiter.

2.2.3. Format numérique de sortie

Malgré toutes les techniques différentes que nous pouvons utiliser pour obtenir une image radiographique numérique, il reste cependant le problème de stockage de ces informations digitales. Le but de cette section est de déterminer les manières dont les images numériques vont être stockées afin qu'elles ne soient pas corrompues. Ceci nous permettra par la suite d'estimer la perturbation inhérente à la prise de mesure et son influence par rapport aux données.

2.2.3.1. *Termes et définitions*

Commençons par définir les quelques termes importants qui vont permettre la compréhension de cette section.

Il faut tout d'abord rappeler qu'une image numérique se mesure en terme de pixels (picture element). Ce pixel est directement lié aux caractéristiques des écrans d'ordinateur et des cartes vidéo qui permettent cet affichage.

En imagerie digitale on parle souvent de la profondeur de l'image en terme de bits. Elle correspond à la précision de la palette de couleurs ou de niveau de gris que vous utilisez pour décrire votre image. Par exemple, une image de profondeur 1 byte correspond à une image ou chaque pixel a une valeur de 0 (noir) ou de 1 (blanc). Autre exemple, une image en couleur de 32 bytes sera moins réaliste qu'une image couleur de 128 bits.

Une *byte* est une séquence adjacente de bits (signal électronique ayant la valeur 0 ou 1), usuellement de huit bits, considérée comme une unité. Dans la littérature pré 1970, le mot 'byte' référait à une variable (length bit string). Depuis cette époque, on fait uniquement référence au chaîne de 8-bit ou octet, dans les écrits concernant les ordinateurs et transferts de données.

2.2.3.2. *Les types d'image*

Sans vouloir particulièrement la comparer aux chefs d'oeuvre du quattrocento, l'imagerie numérique possède toutes les caractéristiques inhérentes à une peinture ou une photographie classique. Comme nous allons le voir dans cette section, il existe différents formats d'images numériques et donc différentes manières de stocker les informations qu'elles contiennent. Nous allons pouvoir ainsi mettre l'accent sur la manière d'apprécier la qualité des mammographies digitales en terme de texture et de profondeur dépendant du format de stockage que l'on aura choisi.

Il existe actuellement quatre grandes familles d'images numériques qui ont toutes leur avantage en fonction de l'usage auquel on les destine.

Les images indexées sont des matrices de m par n pixels (où m, n et $p \in \mathbb{Z}^{*+}$) de nombres en format double, 8 bits ou plus dans l'intervalle $[0, p]$. Ces nombres font référence à une palette de couleurs qui est une matrice de p par 3 de valeurs flottantes comprise entre $[0, 1]$. Chacun de ces trois tableaux de longueur p correspond à une couleur primale, vert, bleu et rouge dont le mélange donne la couleur désirée. L'intérêt de cette méthode est qu'elle permet de conserver un minimum de données à l'intérieure de la matrice puisque la palette est stockée sur l'ordinateur et ainsi d'optimiser l'espace disque.

Les images en ton de gris (ou intensité) fonctionnent sur le même principe que les images indexées puisqu'il s'agit d'une matrice de m par n pixels dont les valeurs font référence à une palette de couleurs qui est, elle, en ton de gris.

Typiquement les valeurs contenues dans l'image dépendent de la profondeur de l'image, $[0,1]$, $[0,255]$, $[0,65535]$, etc. Ce format est le plus utilisé en radiographie puisque, comme nous le savons, les radiographies sont faites à partir de rayons de spectre très étroit.

Les images en couleurs vraies (RGB ou Truecolor) fonctionnent sur un principe différent. Au lieu de faire référence à une palette, le format RGB permet de voir directement le pourcentage des trois couleurs primales verte, bleue et rouge. Il s'agit d'une matrice de m par n par 3 pixels. Chaque couche de m par n pixels représente une couleur des trois couleurs avec une valeur i dépendant de la profondeur de l'image, $[0,1]$, $[0,255]$, $[0,65535]$, etc. Le pixel affiché est alors une superposition de ces trois couleurs avec une intensité i différente pour chacune des couleurs.

Le dernier type de format ressemble au précédent, il s'agit des images en format d'impression (CMYK) composé d'une matrice de m par n par 4 pixels, chaque couche de m par n pixels représentant l'intensité attribuée aux couleurs primaires cyan, magenta, jaune et noire.

De tous ces formats numériques et méthodes d'acquisition, il est important de retenir qu'actuellement les images en ton de gris sont le format standard de stockage en radiographie et qu'elles sont obtenues par un nombre croissant d'appareils radiologiques numériques directs.

2.3. Textures en mammographie

Dans cette section, nous traiterons de la texture de la mammographie digitale en commençant par nous poser la question de l'origine de l'approximation multifractale de la mammographie. Nous continuerons par la nature des différentes sortes de bruits visibles sur une radiographie et les origines probables de la texture dans les mammographies. Nous finirons par exposer les problèmes et les enjeux que représente un programme de caractérisation de texture.

2.3.1. Pourquoi une mammographie serait-elle multifractale ?

Depuis l'introduction de la notion de fractal par Mandelbrot, on peut chercher à représenter les objets qui l'entourent sous forme fractale. Nous allons voir ici que l'approximation multifractale peut s'appliquer au domaine de la radiologie et en particulier à celui de la mammographie.

En étudiant plus précisément la structure interne de la glande mammaire, nous pouvons apercevoir un début d'architecture multifractale. La radiographie ne permettant que la distinction entre les cinq éléments que sont l'air, l'eau, la graisse, les os et le métal, nous pouvons donc affirmer qu'une mammographie d'un sein

sain ne nous permet pas de différencier le réseau sanguin, la graisse et le réseau galactophore. Ainsi une structure fractale, sur un nombre d'échelles limitées, apparaît naturellement par l'entremise des ramifications sanguines de la glande mammaire. Les artères et veines se subdivisant en canaux de plus en plus fins pourraient nous faire penser à un agencement fractal tridimensionnel. Il en est de même pour le réseau galactophore qui par l'intermédiaire des lobes se composant de lobules eux-mêmes se divisant en un agencement de tubulo-alvéoles nous fait penser aussi à un arrangement fractal volumique.

Le rassemblement de ces deux arrangements peut donc faire penser à un objet pourvu de propriétés multifractales sur un nombre d'échelles limitées. Il ne faut cependant pas oublier que la mammographie est une projection bidimensionnelle d'une structure tridimensionnelle.

Nous ne prétendrons pas que les images mammographiques, voire même la physiologie des glandes mammaires, aient une structure multifractale. Nous appliquerons la méthode thermodynamique des multi-fractales dans le formalisme des ondelettes afin d'étudier le coefficient de texture locale. Nous partirons donc ici du principe que tout objet peut être assimilé à un objet multifractal sur un nombre d'échelles limité ou non.

2.3.2. Le bruit en mammographie

Les images mammographiques n'étant qu'une projection à l'aide de rayons X, nous pouvons constater que selon les techniques employées l'image est plus ou moins bruitée permettant une vision plus ou moins claire de la structure interne de la glande mammaire.

2.3.2.1. *Origine quantique du bruit*

La production de rayons X étant faite par effet Bremstrahlung le cône de diffusion de ces rayons ne se fait donc pas de manière homogène mais discrète. Il en résulte alors une granulosité de la radiographie obtenue que l'on peut interpréter visuellement comme la texture caractéristique de l'appareillage utilisé ou un bruit ajouté à l'image.

2.3.2.2. *Origine technique du bruit*

Le bruit quantique n'est pas la seule composante venant perturber l'image radiographique numérique que le médecin étudie. Ainsi, selon la méthode de numérisation utilisée, un bruit supplémentaire plus ou moins important vient s'ajouter.

Le signal le plus bruyant est obtenu par l'emploi de la caméra CCD qui amplifie de manière optique la lumière déjà grossière émergeant du scintillateur.

Cette lumière est par la suite quantifiée par les détecteurs CCD. Un bruit électronique vient donc se rajouter au bruit du scintillateur et des optiques pour finir de perturber l'image radiographique.

Le meilleur rapport signal sur bruit s'obtient évidement par l'utilisation de la méthode directe de numérisation puisque seul le bruit électronique dû au détecteur vient s'ajouter.

Selon la technique de prise des mesures analogiques, CCD ou digitales, l'image aura donc une texture supplémentaire spécifique à la technique de numérisation utilisée qui viendra perturber la visualisation de l'image radiographique.

2.3.3. Origines probables de la texture en mammographie

Il est tout à fait légitime, à ce niveau de l'exposé de se poser la question de l'origine de texture des mammographies que nous allons étudier. Le but de ces prochains paragraphes n'est pas de répondre à la question de la provenance texturale de l'image, mais de poser des hypothèses et des méthodes de vérification quant à l'origine et l'évolution temporelle du coefficient de texture afin de mieux cerner sa provenance. Ces hypothèses nous permettront d'établir de manière plus efficace les problèmes et les enjeux que contient la détection de texture en mammographie.

2.3.3.1. *Origines anatomiques et technologiques*

Comme pour sa nature multifractale, la mammographie trouve sa texture dans le mélange et le rapport entre tissu adipeux, réseau galactophore et sanguin. Cette fusion entre coefficients d'atténuation et projection de volume fait l'unicité de chacune des mammographies. Il est ainsi quasiment impossible d'obtenir deux mammographies identiques.

Cette unicité est tout d'abord due au fait que la mammographie se fait par compression de la glande mammaire qui par son élasticité dispose de manière unique son architecture interne avant sa projection pour l'obtention de l'image radiographique. Cette souplesse dans sa composition permet de prendre des mammographies sous plusieurs angles (projections céphalo-codales et transversales dans la majorité des cas) afin de mieux localiser spatialement les singularités.

Il est ensuite important de se rappeler que la structure mammaire est en constante évolution au court de la vie de la femme. Que ce soit lors de sa construction au moment de la puberté, de sa réorganisation et spécialisation au moment de la grossesse ou de son involution après la ménopause, les rapports de masse entre réseaux galactophores, sanguins et quantités et répartitions des tissus adipeux font que les mammographies d'un même sein ne sont jamais identiques et de textures différentes. Les transformations structurelles influenceront donc plus les

coefficients de texture des parties denses que dans les parties adipeuses. La surface couverte et le rapport graisse sur structure étant nettement plus stable dans le cas des seins adipeux, il est donc à prévoir de forte variation du coefficient de texture pour des seins plus denses tout en restant dans les normes établies par Kestener et al **[13]**(voir figure 5).

Le facteur technologique n'est pas non plus à négliger dans ces considérations texturales. Fort est de constater l'évolution et l'amélioration du matériel permettant la prise de mesures. Si le temps d'exposition a diminué, il est essentiel de percevoir la nette amélioration de la définition des radiographies qui sont faites de nos jours. Il est ainsi très difficile de comparer la texture de mammographies prises à dix ans d'intervalle et il est préférable de commencer une analyse mammaire par la comparaison et la symétrie des structures entre les deux seins.

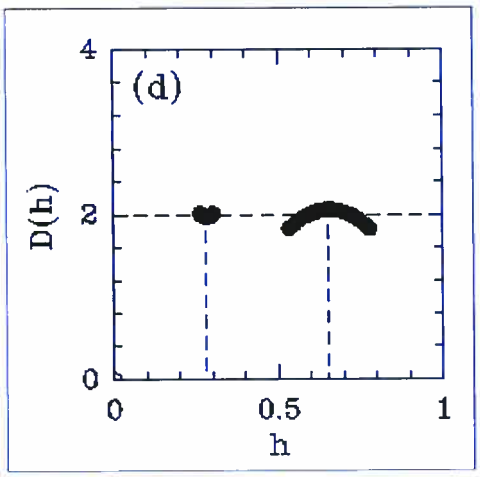


FIG. 5. Spectre des singularités en fonction du coefficient de Hurst, à $h = 0.3 \pm 0.1$ les tissues mammaires sont adipeux et à $h = 0.65 \pm 0.1$ les tissues mammaires sont denses.

2.3.3.2. *Evolution du coefficient de texture*

Afin de pouvoir vérifier l'évolution du coefficient de texture, nous pourrions choisir une région localisée au même endroit sur les mammographies du même sein sur plusieurs années, puis calculer le coefficient de texture et voir son évolution au cours du temps.

Nous pouvons déjà prévoir que la stabilité du coefficient au cours du temps prouverait qu'il ne dépend pas d'une structure unique. Si au contraire on observe une évolution du coefficient vers une valeur plus faible, cela prouvera qu'il dépend d'architecture en involution. Le réseau galactophore serait une bonne hypothèse

dans ce cas puisque l'on observe après la ménopause une nécrose du réseau galactophore donnant ainsi une texture plus fibreuse à la mammographie.

Les résultats finaux escomptés quant à la nature dense ou adipeuse du sein dans le cadre de la mammographie sont montrés par la figure 5.

2.3.3.3. *Problèmes et enjeux*

C'est avec le développement de la radiographie digitale que le besoin de programme de prétraitement des images médicales s'est fait ressentir. Les matériels informatiques variant avec les constructeurs et les tendances il est important de posséder des outils automatiques pour seconder les méthodes traditionnelles que sont la symétrie et la comparaison du sein gauche et droit , ainsi que de complémenter l'expertise médicale et compenser la fatigue de l'oeil humain.

L'évolution de la radiographie digitale permet d'observer avec plus de précision des structures mammaires qui étaient avant invisibles pour le médecin. La contre partie de cette évolution technologique est le doute s'installant dans l'esprit du médecin lorsqu'il se trouve confronté à des masses suspectes qui pourraient être malignes ou tout simplement une restructuration de la glande mammaire.

Le but du calcul du coefficient de texture est donc de pouvoir détecter les singularités telles que les micro-calcifications mais aussi de pouvoir éventuellement classifier ces nouvelles structures de manière à simplifier le rôle du médecin lors du diagnostic.

Chapitre 3

IMPLÉMENTATION DES MÉTHODES DE CALCUL

This is the age of the push buttons (...) programmed correctly, a computer can answer all kind of questions on finance, science, even military strategies and politics.
The Cybernauts, Philip Levene and Sydney Hayers (1965).

L'objectif de cette section est de présenter le programme ayant pour but premier le calcul du coefficient de texture. Nous allons donc suivre le chemin naturel de la mammographie complète à travers l'ensemble de ces programmes nommé Waveplz afin de calculer les différents coefficients de texture qui la composent.

Nous commencerons ce chapitre par une introduction rapide sur les différents langages utilisés par Waveplz ainsi que les diverses librairies de programmation essentielles. Par la suite, nous exposerons les interactions avec l'utilisateur de même que le format des données qui sera utilisé pour le calcul des textures. La fin de ce chapitre concernera essentiellement l'implémentation numérique, la sauvegarde et la visualisation des résultats. Nous débuterons par l'implantation des ondelettes continues avec les approximations qu'il a été nécessaire de faire pour adapter la théorie à la réalité de l'informatique. Nous poursuivrons par la thermodynamique avec le calcul de la fonction de partition, de l'exposant de Hölder et la dimension de Hausdorff. Nous terminerons enfin cette section par la méthode employée pour la sauvegarde des résultats, leur format de stockage ainsi que les diverses méthodes qui ont été développées pour aider à leur visualisation.

3.0.4. Langages de programmation et librairies

Le langage de programmation utilisé pour l'étude de texture du signal est Python (*http ://www.python.org*). Ce langage allie la facilité d'une écriture orientée objet telle que Java avec la simplicité du style de programmation du logiciel Matlab. L'avantage majeur de Python est sa stabilité et sa compatibilité avec les autres langages. Son inconvénient est le manque de documentation sur ses librairies ce qui implique la multiplicité d'une même fonction dans diverses librairies comme par exemple pour un graphique où il est possible d'utiliser la librairie GNU-Plot, DISLIN, gplot, voir même d'importer celle de matlab.

Les librairies utilisées sont issues des librairies standard de Python, C, C++. Afin de faire fonctionner le programme de calcul de texture, il vous faudra cependant installer dans Python les librairies Numeric, Tkinter et dislin (optionnelle) dont les liens se trouvent sur le site de Python.

Afin de pouvoir rendre compatible les programmes avec tous les types de systèmes opérateurs (Windows, Linux, Unix, etc.), il vous faut spécifier la localisation exacte sur votre ordinateur des librairies python et les importer dans la partie vive de votre mémoire au moment du démarrage des programmes. L'exemple ci-dessous vous montre comment effectuer cela pour un ordinateur personnel configuré pour Linux.

```
#! /usr/local/lib/python2.2 python

import sys, os, string, fpformat, os.path, dircache, shutil, time
import math, Tkinter, Numeric
from string import split
from Tkinter import *
from Numeric import *
from FileDialog  import *
```

Maintenant que le langage Python et ses librairies complémentaires sont installés, passons à l'étape d'installation des différents programmes et répertoires que composent Waveplz.

3.0.5. Installation des programmes

Les calculs entropiques qui sont effectués sur différents types de signaux (une dimension ou deux dimensions) à l'aide d'un ensemble de programmes nommés *waveplz* (version 2.2 ou supérieure). Waveplz se compose des programmes :
- waveplz.py,
- guiWaveplz.py,
- wavelet1d,
- wavelet2d,
- Chain1d,
- Chain2d,
- basilik40.py.

Afin d'être sûr du bon fonctionnement de Waveplz commençons par créer quelques répertoires de travail.

Pour plus de facilité nous considérerons que votre répertoire de travail courant se nomme :

home/vous/

A l'intérieur de ce répertoire créez un répertoire waveplz, à l'intérieur duquel vous générerez les répertoires exec/, step0/, step1/, matlab/ et work/ afin d'obtenir l'architecture suivante :

home/vous/waveplz/exec/step0/
home/vous/waveplz/exec/step1/
home/vous/waveplz/matlab/
home/vous/waveplz/work/

Maintenant il vous faut placer le programme waveplz.py (annexe B) dans le répertoire waveplz/. Puis copier guiWvplz.py (annexe C) et wave.i (annexe A) dans le répertoire step0/. Classer wavelet1d (et .exe), wavelet2d (et .exe), Chain1d (et .exe), Chain2d (et .exe) et basilik40.py (annexe D) dans le répertoire step1/. Finalement il ne vous restera plus qu'à ranger les fonctions matlab bVisu4.m (annexe G), bVisu6.m (annexe H), showIm.m, showLine.m et ShowSurf.m (annexe I) dans le répertoire matlab/. Si ces étapes ont été faites correctement vous devriez avoir sur votre disque dur la structure suivante :

home/vous/waveplz/waveplz.py
home/vous/waveplz/exec/step0/guiplz.py
home/vous/waveplz/exec/step0/wave.i

home/vous/waveplz/exec/step1/wavelet1d
home/vous/waveplz/exec/step1/wavelet1d.exe

home/vous/waveplz/exec/step1/wavelet2d
home/vous/waveplz/exec/step1/wavelet2d.exe
home/vous/waveplz/exec/step1/Chain1d
home/vous/waveplz/exec/step1/Chain1d.exe
home/vous/waveplz/exec/step1/Chain2d
home/vous/waveplz/exec/step1/Chain2d.exe
home/vous/waveplz/exec/step1/basilik40.py

home/vous/waveplz/matlab/bVisu4.m
home/vous/waveplz/matlab/bVisu6.m
home/vous/waveplz/matlab/showIm.m
home/vous/waveplz/matlab/showLine.m
home/vous/waveplz/matlab/showSurf.m

home/vous/waveplz/work/

Il ne nous reste plus, pour démarrer les programmes, qu'à activer le programme waveplzXY.py. Un algorithme résumant le fonctionnement complet ainsi que les répertoires et les fichiers que compose waveplz se trouvent dans la figure 1.

3.1. Formatage des données et interactions avec l'utilisateur

Le programme waveplz est du type semi-automatique puisqu'il nécessite des réglages par l'utilisateur. Cependant le programme est fait de telle sorte que ce paramétrage reste optionnel puisque des valeurs sont intégrées par défaut permettant de faire une analyse complètement automatique si nécessaire. Ces réajustements permettent toutefois d'éliminer plus finement les effets de bord de l'image par l'usage de filtres prédéterminés.

3.1.1. Choix des fichiers de données

Le programme débute lorsque vous exécutez le programme waveplz.py. Le programme débute par l'apparition d'une fenêtre vous confirmant le fait que vous exécutez le traitement de signal waveplz. Par la suite, il vous sera alors demandé d'entrer la localisation du signal à analyser. Cette localisation se fait par l'intermédiaire d'un module se nommant *class FileChoose* (voir annexe B pour la syntaxe exact du procédé).

Dans sa forme actuelle, waveplz ne traite que les fichiers de formats numériques en forme de tableau. Dans une prochaine version, des codecs (programmes de transformation automatique de formats compressés en formats numériques ASCII) seront implémentés afin de pouvoir effectuer le traitement d'images de toutes sortes pour plus de commodité pour l'utilisateur. Il est donc important que votre fichier de données soit un tableau en format ASCII contenant des valeurs en format *DOUBLE*.

Une fois que vous aurez sélectionné le fichier de données, le programme créera de manière automatique un répertoire dont le nom commencera par le nom de votre fichier de données suivit de l'heure du début du traitement du signal. Ce nouveau répertoire sera localisé dans le répertoire work/.

Une copie du fichier de données sous le nom data.m est alors faite dans le nouveau répertoire de stockage. Ceci permet de pouvoir travailler sur le fichier sans détruire les données d'origine en toute sécurité par l'intervention de la classe *bLoad*.

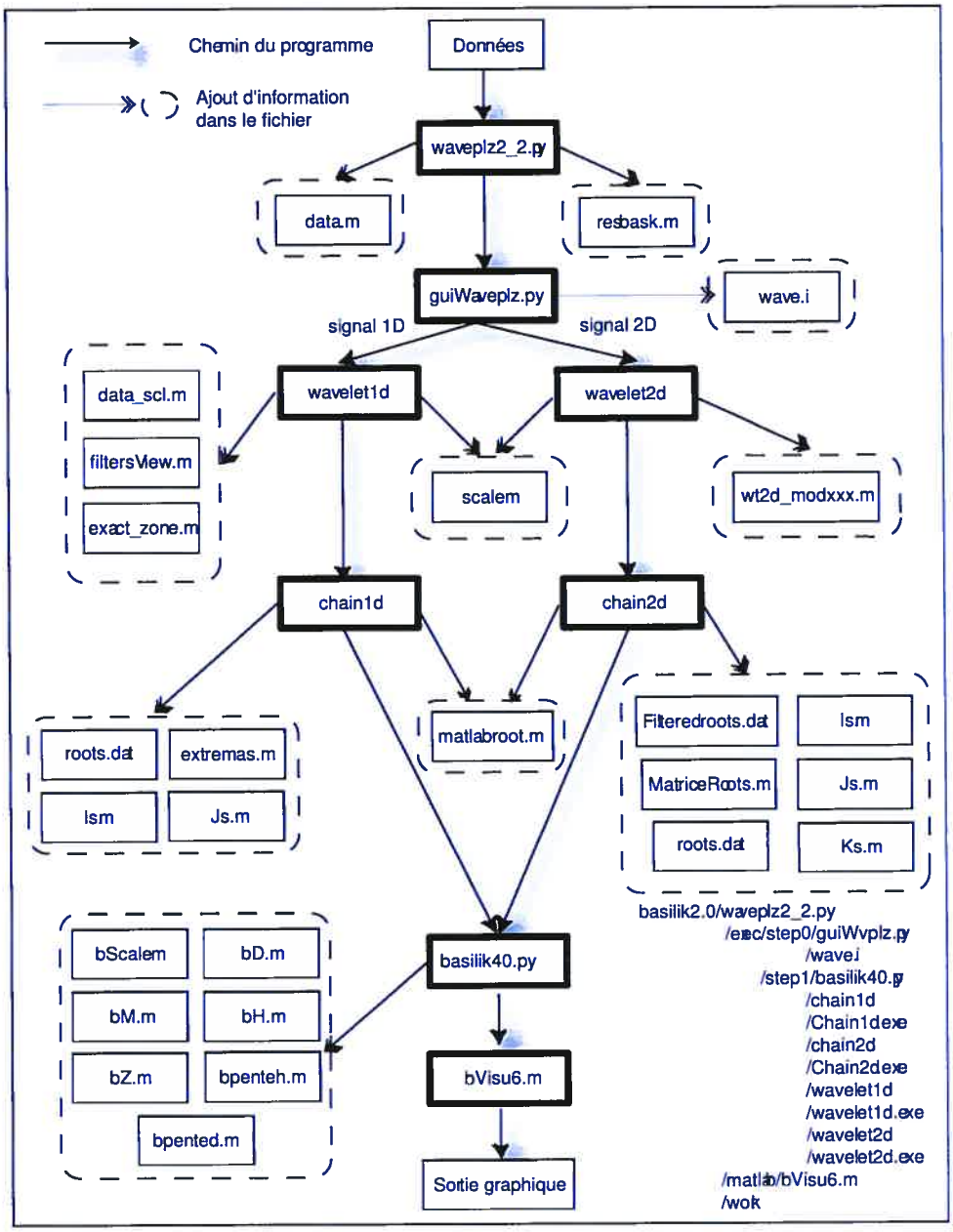


FIG. 1. Algorithme pour l'obtention du coefficient de texture d'une image

Le programme crée alors un répertoire tampon temp/ où tous les calculs seront effectués. Une copie des exécutables est alors réalisée dans le répertoire temp/. Les exécutables se trouvent normalement dans le répertoire exec/.

A partir de ce moment, l'ordinateur calcule automatiquement la taille du tableau de données contenu dans data.m afin de déterminer la dimension du signal à étudier et faire un premier réglage des paramètres qui seront utilisés par la suite. Ces paramètres pourront être modifiés par l'utilisateur à cet instant.

3.1.2. Interactions avec l'utilisateur

A ce stade, nous commençons à connaître quelques caractéristiques de notre signal et Waveplz en tient compte alors qu'il crée la structure du fichier wave.i. Ce fichier contient toutes les informations inhérentes au signal telles que la dimension du signal et les particularités du traitement qu'il va subir telles que le type de filtre à utiliser, le nombre de voix et d'octaves qui va être utilisés (c'est-à-dire le nombre d'échelles considéré).

Ces informations contenues dans le fichier wave.i (annexe A) peuvent être suffisantes mais peuvent toutefois être affinées par une intervention de l'utilisateur. Afin que cette interaction soit possible waveplz.py appelle le programme guiWvplz.py (annexe C). L'activation de guiWvplz se fait grâce à la commande *os.name* importée de la librarie os.path qui renvoie le type de plateforme utilisé sous les différents noms *posix* (toutes les plateformes Unix ou Linux), *winnt* (toutes les plateformes Windows) et *mac* (plateforme Mac).

Cette interaction entre l'utilisateur et le programme est possible par l'importation dans guiWvplz des librairies Tkinter, tkSimpleDialog et TkMessageBox et par le développement de la classe Dialog(). Dialog() est assez malléable pour pouvoir ajouter autant d'informations nécessaires pour le bon déroulement au cours de l'évolution du programme. Elle permet d'envoyer les données directement à la classe Maxima() de waveplz.py et de basilik40.py (équivalent de la classe Main dans un programme conventionnel) ainsi qu'aux programmes de transformées en ondelettes et de chaînage.

Pour plus de simplicité et de rapidité dans l'exécution des programmes, toutes les informations nécessaires sont demandées en une seule fois évitant ainsi les multiples interventions que nécessiterait par exemple une grande mammographie.

Lors de l'exécution du programme guiWvplz.py deux fenêtres doivent apparaître. Une première vous souhaitera la bienvenue sous basilik (elle permet de constater l'activité des programmes), l'autre vous permettra d'entrer les valeurs pour estimer plus précisément la rugosité de votre signal.

Waveplz.py active alors le programme guiWaveplz.py dans votre répertoire de travail et lit le format de vos données (une ou deux dimensions) et vous propose différentes options d'action sur vos données. Il vous faudra d'abord choisir les paramètres pour les calculs des coefficients d'ondelettes selon qu'il s'agit d'un signal

une ou deux dimensions.
Pour plus de facilité, voici la correspondance des divers champs (voir aussi figure 2).

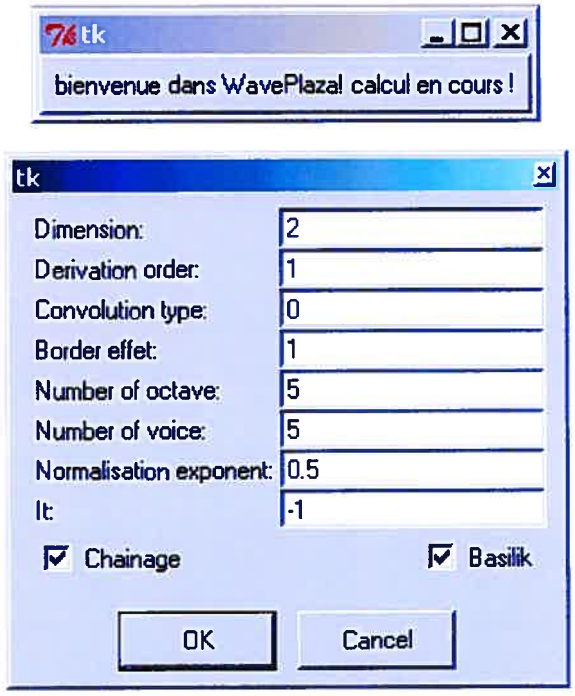


FIG. 2. Interface d'affinage de calcul de waveplz

3.1.2.1. *La dimension*

Le champ correspondant à la dimension est calculé de manière automatique par le programme à partir de la copie du fichier de données dont vous avez saisi précédemment la localisation et qui est sauvegardé dans le répertoire de stockage des résultats sous le nom data.m.

Seules les valeurs 1 et 2 peuvent être comprises par le programme qui les sauvegardera dans le fichier wave.i sous les formes suivantes :

```
dimension = 1 Wavelet1D
            2 Wavelet2D
```

3.1.2.2. *L'ordre de dérivation*

Cette valeur correspond à l'ordre de la dérivation de la fonction gaussienne qui va être utilisée comme noyau pour le calcul de la transformée en ondelettes. Dans l'état actuel du programme, seules la première et la seconde dérivée sont possibles par l'intermédiaire des valeurs numériques 1 et 2. Les tests pour les signaux unidimensionnels et bidimensionnels ont été fait respectivement avec la première et la seconde dérivée. Ce paramètre se sauvegardera dans le fichier wave.i sous la forme :

```
Derivation order : 1
                   2
```

3.1.2.3. *Les différents types de convolutions*

Ce champ permet de déterminer la nature de la convolution qui va être utilisée pour le calcul des coefficients d'ondelettes. La valeur zéro correspond à une convolution directe qu'il est recommandé d'utiliser pour dans le cas de l'analyse unidimensionnelle. La valeur 1un permet une convolution plus rapide par l'usage des transformées rapides de Fourier (Fast Fourier Transform) [**22**]. La convolution par les transformées rapides de Fourier est fortement conseillée dans le cas d'images volumineuses telles que les mammographies. Ce paramètre se sauvegardera dans le fichier wave.i sous la forme :

```
Convolution Type : DIRECT_CONV
                   FFTW_CONV
```

3.1.2.4. *Les filtres : effets de bords*

Lors du calcul des coefficients d'ondelettes par ordinateur, on constate l'apparition d'effet de bord causé par les approximations numériques des différentes convolutions effectuées sur les mesures du signal. Pour atténuer ce phénomène et éviter une trop grande perturbation des résultats, on utilise différentes sortes d'effets que l'on convolue aux mesures afin de réduire cet effet de bord. Quatre options sont disponibles, représentées ici dans ce champ par les valeurs entières 0, 1, 2 et 3. Ces chiffres correspondent respectivement au filtre périodique qui étend les mesures du signal par l'ajout du même signal de chaque côté, à l'effet miroir qui réplique de chaque côté du signal le même signal mais inversé à la manière d'un miroir, et aux filtres padding et 0 padding. Par défaut, ce paramètre est réglé sur l'effet miroir. Ce paramètre se sauvegardera dans le fichier wave.i sous la forme :

```
BorderEffect : CV1D_PERIODIC
               CV1D_MIRROR
               CV1D_PADDING
               CV1D_0_PADDING
```

3.1.2.5. *Le nombre d'octaves*

Comme cette analyse est multi-échelles, il faut régler ce paramètre en sachant que la taille minimale d'une image peut être de quatre pixels et que le nombre de voix qui va subdiviser chaque octave doit toujours être égal ou supérieur au nombre d'octaves. Ainsi une image va se réduire dans les échelles par un facteur 2 N fois, où N est un entier réel strictement positif correspondant au nombre d'octaves. Par défaut cette valeur est réglée sur cinq.

3.1.2.6. *Le nombre de voix*

Chaque octave est divisée par un nombre de voix. Dans l'implémentation, ce nombre est toujours supérieur ou égal au nombre d'octaves. Cette valeur est paramétrée à cinq par défaut.

3.1.2.7. *L'exposant de normalisation*

L'exposant de normalisation s'applique à l'ensemble des coefficients du scalogramme afin que les propriétés de l'ondelette et des transformées de l'ondelette soient conservées. Ce champ est initialement défini pour une valeur de 0.5 et n'a rien à voir avec le facteur de normalisation de la transformée en ondelettes (voir équation 38). Suite à une erreur dans l'implantation de cet code par le programmateur des ondelettes, ce paramètre n'a aucune influence sur le résultat final de l'analyse thermodynamique du signal.

3.1.2.8. *L'itération*

Le champ concernant l'itération permet de définir une analyse multi-échelles sur un nombre de niveaux déterminé. Ainsi, si N_e est le nombre d'échelles que l'on obtient en multipliant le nombre d'octaves par le nombre de voix, la valeur inscrite dans ce champ doit être un entier inférieur ou égal à N_e. La valeur de -1 inscrite par défaut signifie que le programme va itérer sur tous les niveaux.

3.1.2.9. *L'activation du chaînage*

Ce champ, coché par défaut, permet l'activation automatique du programme de chaînage des maxima des transformées en ondelettes à la suite de l'exécution du calcul de ces coefficients.

3.1.2.10. *L'activation de la thermodynamique*

Ce champ, coché par défaut, permet l'activation du programme basilik40.py permettant le calcul des textures du signal uni ou bidimensionnel désiré.

A partir de ce moment, le programme possède toutes les informations nécessaires pour le traitement du signal et termine l'exécution de guiWvplz par la sauvegarde de ces données dans le fichier wave.i.

3.1.3. Formatage des fichiers de données

Le programme waveplz adapte, à partir de cet instant, le reste du traitement en reconnaissant le format des données à traiter par le calcul rapide de la taille du tableau. Ce calcul se fait de manière classique à travers la classe bLoad(), par comptage du nombre de lignes et de colonnes formant le tableau. Si X et Y sont respectivement la largeur et la hauteur du signal et Textfile() la classe permettant la lecture de fichier en format ASCII, le calcul de X et Y s'écrit en Python de la manière suivante :

```
class bLoad:

    def __init__(self, name):

        X = self.words(name)
        Y = self.lines(name)

        ...

    ...

    def lines(self,name):
        lines = 0
        #calcul de la hauteur du tableau de donnees
        for line in TextFile(name):
            lines = lines + 1
        return lines

    def words(self,name):

        lines = 0
        words = 0
        max = 0
        #calcul de la largeur max du tableau de donnees
        for line in TextFile(name):
```

```
            words = len(split(line))
            if (max - words)<0:
                max = words
        return max
```

Cette classe permet de déterminer la longueur maximale et la largeur maximale du fichier au cas où, pour économiser de l'espace sur le fichier, les zéros de fin de ligne auraient été supprimés.

Puisque nous connaissons la taille du signal, waveplz peut alors préparer le répertoire de stockage des résultats. En effet selon les recherches faites par Decoster sur l'analyse multifractale d'images de surfaces rugueuses [**22**], un signal bidimensionnel doit avoir une taille minimale de 512 par 512 pixels afin de pouvoir appliquer la transformation en ondelettes et obtenir un résultat correct. A partir de ce résultat nous devons donc adapter notre technique de recherche de textures. Dans cette optique, waveplz va morceler les données en images de 512 par 512 pixels se chevauchant de 256 pixels de chaque côté pour avoir une analyse complète de l'image malgré les effets de bord. Chaque morceau est ainsi numéroté de 000000 à 999999 divisant, telle une mosaïque, l'image d'origine au maximum en 1000 colonnes et 1000 lignes d'images. Le numéro de cette image se retrouve sur le nom du répertoire qui contiendra ses résultats.

Une première approximation est ici faite puisque ce fractionnement du signal original implique que nous n'aurons pas une valeur précise de la texture d'un cadre entourant notre image d'une largeur de 25% du coté de l'image (à cause des effets de bord perturbant les images analysées).

Une itération permet ici à waveplz de faire l'analyse en ondelettes et thermodynamique de chaque parcelle de l'image par l'implantation du code suivant dans waveplz :

```
#test signal 2D:
if ((self.nbLigne - 1) > 0) & ((self.nbCol - 1) > 0) :

     #test signal > 512*512
     if ((self.nbLigne - 512) > 0) | ((self.nbCol - 512) > 0) :
         Nx = int(self.nbLigne/256)
         Ny = int(self.nbCol/256)

         if Nx<1:Nx=1
         if Ny<1:Ny=1
```

```
for i in range(Nx-1):
    for j in range(Ny-1):
        ...
```

3.2. Implantation numérique des ondelettes

L'implantation numérique des ondelettes continues et leur chaînage, tels que compilés dans les programmes wavelet1d, wavelet2d, Chain1d et Chain2d, ont été programmés par Basile-Bellavance. Les programmes utilisés ici sont des compilations des librairies CRM_wtd1, CRM_wt2 et CRM_chaining programmées en langage C++. Ces librairies sont largement inspirées de la librairie *xmurf* développée par Decoster de l'université de Bordeaux I dans le cadre de l'analyse multifractale d'images de surfaces rugueuses.

Le but de cette section n'est pas d'expliquer dans le détail les librairies dont ces programmes sont issus, mais plus simplement de décrire les principes fondamentaux et les approximations qu'impose l'implantation numérique aux théories des Ondelettes.

Nous commencerons par expliquer le but des programmes wavelet1d et wavelet2d et leurs algorithmes de fonctionnement. Puis nous examinerons le fonctionnement du chaînage des maxima des coefficients calculés précédemment qui nous permettront l'analyse thermodynamique des signaux.

3.2.1. Implantation numérique des transformées en ondelettes

Les programmes wavelet1d et wavelet2d sont des outils d'analyse permettant de réaliser une série de transformées en ondelettes continues pour un signal respectivement à une ou deux dimensions. Le noyau de l'ondelette peut être une gaussienne de dérivées première ou seconde en fonction de la sélection faite au début du programme waveplz. Il est essentielle de faire remarquer ici aux lecteurs que ces deux programmes, développés indépendamment à cette maîtrise, produisent une erreur systématique dans la valeur des coefficients en ondelettes. Cette erreur peut toute fois être corrigée à la visualisation des résultats (voir section 4.1).

Les coefficients en ondelettes sont calculés à l'aide des programmes wavelet2d et wavelet1d, spécifiques au type de signal à étudier (1D ou 2D). Ces programmes sont activés automatiquement à partir du programme guiWaveplz.py, mais peuvent aussi être appelés de manière manuelle sans l'intervention du programme waveplz par l'activation des commandes suivantes :

wavelet1d
(si votre image est un signal 1D)

ou

wavelet2d
(si votre image est un signal 2D)

Le programme wavelet1d gère le calcul et la mémoire pour réaliser la transformée en ondelettes pour des signaux une dimension. Ces deux fonctions utilisent une seule transformée en ondelettes dans chaque orientation en deux dimensions à chaque échelle (voir section 1.3.3). Le signal est convolué avec l'effet choisi pour atténuer les effets de bord puis à la dérivée de la fonction gaussienne choisie. Tous les coefficients des transformées sont stockés dans un même fichier qui servira par la suite au chaînage.

Le programme wavelet2d effectue le même travail que son homologue wavelet1d mais sur des signaux bidimensionnels tels que des mammographies. Les coefficients sont stockés dans des fichiers numérotés correspondant à leurs échelles respectives. Comme dans le cas unidimensionnel, l'ensemble de ces fichiers permettra la création des lignes de maxima.

Il est utile de noter que, comme il est décrit plus haut, l'ondelette analysatrice utilisée en deux dimensions au cours de ces recherches a été le chapeau mexicain (voir figure 3). Les langages utilisés pour le développement des librairies sont le C, C++ [**38**].

3.2.2. Implantation numérique du chaînage des maxima

Les lignes de maxima sont ensuite chaînées à l'aide des programmes Chain1d et Chain2d à partir des maxima calculés. Chain1d et Chain2d permettent le chaînage des points dans un espace 2d ou 3d correspondant respectivement à l'analyse multi-échelles des signaux une et deux dimensions. Ces points sont sélectionnés dans une matrice de données selon un critère défini et sont ensuite chaînés.

Ainsi, la gestion de la sélection des points à chaîner se fait par une recherche d'extrema. Le chaînage des extrema se fait suivant la direction verticale pour les signaux 2D et horizontale pour les signaux 1D. Une fois le chaînage terminé, un algorithme de recherche des bifurcations entre en action. Ces bifurcations permettent de préciser la structure de l'arbre initialement chaîné. Ainsi, en connaissant le lieu où se trouvent les bifurcations dans les branches de l'arbre, il est possible de décrire exactement tous les éléments de l'arbre en énumérant les bifurcations et les descendants des racines issues de chacune de ces bifurcations. Ces informations sont écrites à la toute fin dans des fichiers de sorties appropriés [**38**].

Le schéma de fonctionnement et les fichiers de sortie qui en résultent sont résumés sur un diagramme décrivant tout l'algorithme de fonctionnement de Waveplz et se trouvant dans la figure 1.

Une activation manuelle des programmes peut aussi être possible à l'aide des commandes :

chain1d wave.i
(pour un signal 1D)

ou

chain2d wave.i
(pour un signal 2D)

Si le processus s'est déroulé correctement vous obtiendrez en final les deux fichiers essentiels pour le calcul des textures *matlabroot.m* et *wave.i*.

3.3. Implantation numérique de la thermodynamique

Le calcul et le stockage des lignes de maxima ayant été exécutés, waveplz peut maintenant commencer à appliquer les concepts de thermodynamique pour le calcul des coefficients de texture. Le but de cette section et de voir les différentes méthodes numériques employées pour arriver à cette fin. Nous commencerons par justifier la nécessité de la régularisation des lignes de maxima avant d'effectuer le calcul des fonctions de partition. Nous terminerons par l'obtention du coefficient de Hölder et de la dimension de Hausdorff.

Comme on peut le constater en étudiant le code de basilik40.py (voir annexe D), le programme se sépare en plusieurs classes et une commande permettant l'intégration de fichiers de données dans un format compressé.

```
# Use the gzip module for Python version 1.5.2 or higher
gzip = None
try:
    _version = map(string.atoi,
                   string.split(string.split(sys.version)[0], '.'))
    if _version >= [1, 5, 2]:
        try:
            import gzip
        except ImportError:
            gzip = None
except:
    pass
```

```
max = Maxima()
print "fichiers sauvegardes"
```

Ces commandes permettent, entre autres, l'activation de la classe principale Maxima.

3.3.1. Linéarisation des lignes de maxima

Les informations étant maintenant disponibles pour le programme, basilik40 peut commencer la sélection des lignes de maxima. Préalablement, le programme a stocké toutes les lignes de maxima en mémoire en séparant les valeurs des maxima et de leur localisation spatiale (sur le signal mais aussi dans les échelles), ainsi que la longueur des lignes de maxima.

La sélection des lignes de maxima se fait par la fonction même calcNbLign(). Cette fonction prépare d'abord les valeurs des coordonnées des bornes à l'intérieur desquelles les lignes de maxima seront pertinentes. Ainsi on sélectionnera les lignes les plus grandes partant de l'échelle la plus fine et qui ne sont pas localisées près du bord à cette même échelle (soit une bande 25% de chaque côté du signal).

La fonction détermine le type du signal (signal 1D ou 2D) à partir des informations données par l'objet self.image. Cette partie du code permet donc à basilik de ne pas être *bloqué* par la dimension du signal étudié. Par la suite, pour chaque ligne de maxima, le programme calcule leur longueur et assimile leurs coordonnées. Si la ligne de maxima se localise dans le bloc de données pertinentes et si elle est de longueur suffisante (> 2) son numéro est noté dans un tableau qui permettra plus tard de retrouver plus facilement cette ligne de maxima et ainsi économiser la mémoire de l'ordinateur.

Après avoir sélectionné les bonnes lignes de maxima et affiché leur nombre sur l'écran, basilik passe maintenant à leur *régularisation*. La fonction self.regulRoot permet de remettre de l'ordre dans chaque ligne de maxima et ainsi avoir toujours la plus grande valeur absolue en haut de la ligne de maxima répondant ainsi à la contrainte stipulée dans l'équation 60 et évitant une contradiction avec la deuxième loi de la thermodynamique. Après cette régularisation, les données sont stockées dans la variable self.M et dans le fichier bM.m.

3.3.2. Calcul de la fonction de partition

Après la sélection des coefficients q (par défaut q varie de -4 à +4), la fonction self.partition() permet de calculer l'énergie libre $Z(a, q)$ répondant aux lignes de maxima telle que décrite dans l'équation 62 à l'échelle donnée a. Cette fonction

est assez simple puisqu'elle fait une somation des q_{ime} puissances des amplitudes des maxima à une échelle a. La fonction self.partition() s'écrit de manière triviale :

```
def partition(self):
    q = self.q
    cst = len(q)/2
    levels = self.lgnMax
    nbLines = len(self.lignes)
    M = self.M
    Mtemp = zeros([nbLines,levels])
    Mtemp = Mtemp.astype(Float64)
    Z = zeros([len(q),levels])
    Z = Z.astype(Float64)

    for i in q:
        for j in range(nbLines):
            for k in range(levels):
                if (M[j,k]- 1e-100) <= 0 :
                    Mtemp[j,k] = 0
                else:
                    Mtemp[j,k] = self.puiss(M[j,k],i)
        temp = sum(Mtemp,axis=0)
        for m in range(len(temp)):
            Z[(i+cst),m] = temp[m]
    ...
```

3.3.3. Calcul du poids de Boltzmann

Le calcul se fait à l'appel de la fonction au fur et à mesure qu'elle est nécessaire dans les autres fonctions. Dépendant de trois variables distinctes, il serait trop volumineux du point de vue espace mémoire de vouloir le calculer au complet en une seule fois. Sa syntaxe permet d'éviter les erreurs d'approximation dues aux précédents calculs en forçant à zéro les valeurs exubérantes.

```
def W(self,M,Z,q):
    # Calcul du poids de Boltzmann
    if ((Z - 1e-100) <= 0 ) | ((M - 1e-100) <= 0 ):
        w = 0
    else :
        w = float((abs(M))**q)/float(Z)
    return w
```

3.3.4. Calcul de l'exposant de Hölder

À ce stade, le programme peut calculer les exposants de Hölder grâce à l'implémentation de l'équation 63 dans la fonction self.h(). Toutefois, une particularité du code est à noter. Afin d'éviter le logarithme de zéro (si la valeur M est nulle ou très petite), le programme identifie la valeur de M à utiliser. Si la valeur de M est nulle, voire petite, alors cet exposant particulier est égal a zéro. Dans le cas où M n'est pas nul, la fonction self.W() permet de calculer le poids de Boltzmann selon l'équation 61 sans avoir l'inconvénient de stocker toutes ces valeurs (ce qui correspondrait à un tableau de trois dimensions et serait donc très coûteux en mémoire).

3.3.5. Calcul de la dimension de Hausdorff

Basilik peut maintenant calculer les coefficients correspondants à la dimension de Hausdorff grâce à l'implantation de l'équation 64 dans la fonction self.d(). La même technique de détection des valeurs nulles pour éviter le logarithme de zéro est employée ici.

Sur le même principe, on calcule les pentes de $h(a, q)$ et $D(a, q)$ sur le modèle des équations 63 et 64 grâce aux fonctions self.pente-holder() et self.pente-d() dont les valeurs sont stockées respectivement dans les matrices self.h-pente et self.d-pente.

Après une confirmation de la sauvegarde des fichiers le programme s'arrête. Il ne nous reste plus qu'à visualiser les résultats.

3.4. Sauvegarde et visualisation

3.4.1. Extraction et sauvegarde des données

Afin de pouvoir palier aux problèmes d'installation de la librairie *Scientific* sous Linux, basilik contient la classe TextFile empruntée directement de la librairie scientific/IO. Cette classe permet ainsi d'ouvrir n'importe quel fichier même compressé afin d'en lire le contenu. Afin de pouvoir sauver n'importe quelle variable telle qu'un tableau de données, il a tout de même fallu développer la classe SauveInfo.

```
class SauveInfo:


    def readMatrix(filename):
        rows = []
```

```
        for line in TextFile(self,filename):
            columns = []
            for number in string.split(line):
                columns.append(string.atof(number))
            rows.append(columns)
        return Numeric.array(rows)

    def writeMatrix(self,a, filename):
        file = TextFile(filename, 'w')
        for line in a:
            for number in line:
                file.write(`number` + ' ')
            file.write('\n')
        file.close()
```

L'appel de la fonction writeMatrix de la classe SauveInfo nous permet de sauvegarder les informations utiles dans le répertoire tampon. Son appel à l'intérieur du programme se fait ainsi :

```
self.sauve = SauveInfo()
self.sauve.writeMatrix(nomdelavariable,'bnomdelavariable.m')
```

Cette version de basilik étant encore en développement, tous les tableaux de variables sont sauvegardés au fur et à mesure de leur calcul au cours du programme et commencent tous par la lettre b. De ce fait, à la fin de l'exécution de basilik40.py, on doit avoir les fichiers suivants dans chaque répertoire de chaque image composant la mosaïque de l'image globale :

- **bD.m** : la matrice de stockage des valeurs de la dimension de Hausdorff (eq. 64) en fonction de q (par défaut $q = -4, ..., 0, ..., +4$) et du niveau a.

- **bH.m** : la matrice de stockage des valeurs de l'exposant de Hölder (eq. 63) en fonction de q (par défaut $q = -4, ..., 0, ..., +4$) et du niveau a.

- **bM.m** : la matrice de stockage des valeurs normalisées des maxima sur les lignes de maxima (eq. 60) en fonction des lignes de maxima pertinentes (qui ne sont pas sujettes aux effets de bord après une sélection par l'utilisateur de la largeur du bord à négliger)

- **bpented.m** : la matrice de stockage des valeurs de la pente de la dimension de Hausdorff (eq. 66) en fonction de q (par défaut $q = -4, ..., 0, ..., +4$) et du niveau a (on peut ainsi voir l'évolution de $D(q)$ plus le niveau a tend vers 0^+)

- **bpenteh.m** : la matrice de stockage des valeurs de la pente de l'exposant de Hölder (eq. 65) en fonction de q (par défaut $q = -4, ..., 0, ..., +4$) et du niveau a (on peut ainsi voir l'évolution de $h(q)$ plus le niveau a tend vers 0^+)

- **bScale.m** : la matrice de stockage des valeurs exactes des niveaux a utilisés pour les calculs et qui servirons pour les représentations graphiques.

- **bZ.m** : la matrice de stockage des valeurs de la fonction de partition (eq. 62) en fonction de q (par défaut $q = -4, ..., 0, ..., +4$) et du niveau a.

Le calcul pour obtenir la texture de chaque sous-image se fait de façon répétée et identique pour toutes les images. Au moment de la fin de l'exécution du programme, waveplz efface les variables de la mémoire vive ainsi que le répertoire tampon qui a permis le stockage temporaire de tous ces calculs.

3.4.2. Visualisations graphiques

On peut observer les graphiques de deux manières différentes, par la libraire DISLIN ou par le programme matlab.

3.4.2.1. *Visualisation grâce à la classe plot*

La visualisation des résultats par la librairie DISLIN se fait de manière automatique lorsque vous activez le programme bVisu.py (voir annexe E). Comme on peut le constater, les graphiques sont limités au strict nécessaire. Vous pouvez observer, à la suite des calculs d'entropie, les graphiques :

- des maxima en fonction des échelles (notés *Scale versus M*)

- des valeurs des fonctions de partition en fonction de q et de l'échelle (notées *Scale versus Z*)

- des valeurs des exposants de Hölder en fonction de q et de l'échelle (notées *Scale versus H*)

- des valeurs des coefficients de Hausdorff en fonction de q et de l'échelle (noté *Scale versus D*)

Pour des graphiques plus pertinents et exacts, il est recommandé d'utiliser les programmes bVisu4.m et bVisu6.m sous matlab.

3.4.2.2. *Visualisation ciblée avec bVisu42.m*

Le code bVisu42.m est un programme spécifiquement développé pour la visualisation des résultats générés par waveplz. Les graphiques qui en résultent sont identiques que pour le programme bVisu.py. Une différence cependant notable est qu'il fonctionne sous matlab ce qui permet la sauvegarde des graphiques sous un format plus convivial tel qu'une image.

Note :une amélioration due au calibrage de waveplz a été effectué dans la version bVisu43.m permettant la visualisation directe des résultats de l'analyse des signaux bidimensionnels en tenant compte de l'erreur systématique (voir figure 3 et chapitre suivant).

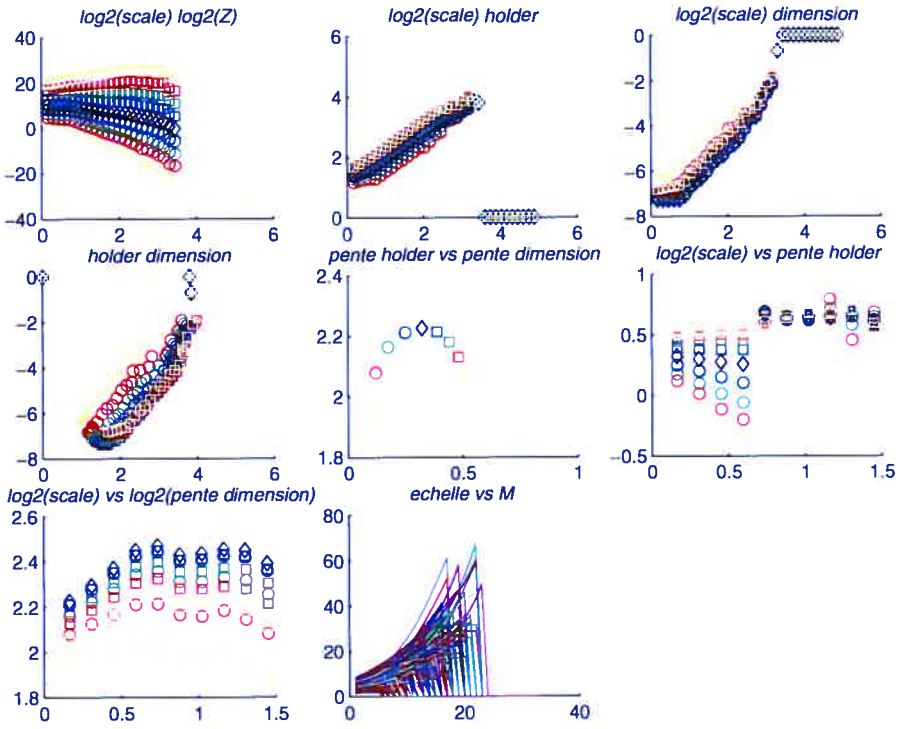


FIG. 3. Visualisation des résultats par bVisu43.m d'un brownien fractionnaire 2D de $h = 0.3$ pour $q \in [-4, 4]$.

3.4.2.3. *Visualisation globale avec resbask.m et bVisu6.m*

Il est cependant possible d'avoir une vision globale de tous les résultats que contient chaque parcelle de l'image complète. A l'exécution de waveplz, les informations concernant la pente de dimension de Hausdorff (pour les petites échelles)

ainsi que le coefficient de texture (pour les petites échelles) ont été récoltés au fur et à mesure de leur calcul. Toutes ces informations classées sous la forme d'un tableau représentant la subdivision de l'image principale se nomme resbask.m. bVisu6 permet la visualisation de ce tableau sous la forme de colonnes avec, en parallèle, une miniature de l'image permettant la localisation de ces valeurs. L'échelle des couleurs se fait automatiquement par matlab avec le bleu foncé pour le coefficient de texture la plus basse et rouge foncé pour la plus haute. Il est ainsi possible de repérer plus aisément les singularités texturales présentes dans votre image (voir figure 4).

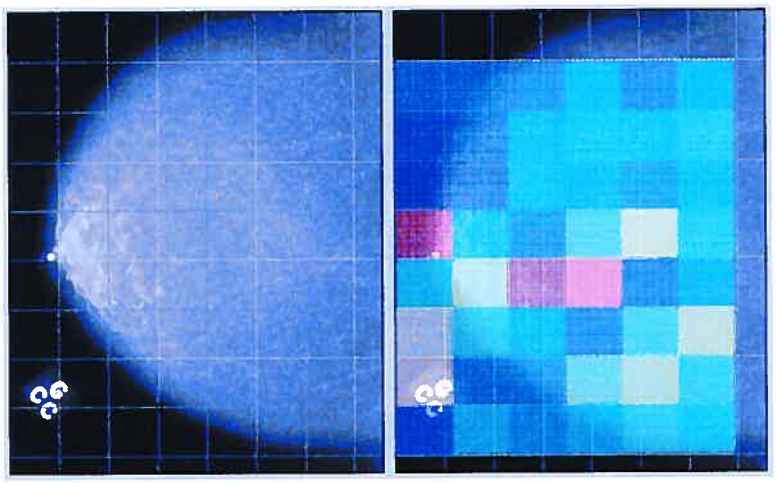


FIG. 4. Visualisation par bVisu6.m de la détection de micro-calcifications et de textures cancéreuses dans une mammographie

Chapitre 4

EXPÉRIMENTATIONS NUMÉRIQUES

La Méthode des Maxima des Transformées d'Ondelettes (MMTO) a été utilisée au cours des dernières années dans divers domaines. Développée à l'origine pour la détection de signaux sismiques, cette méthode s'applique maintenant dans des domaines tels que la finance, l'analyse de chaînes d'ADN, la différentiation de types de nuages ou l'analyse de turbulences [7].

Dans ce chapitre, nous allons exposer les résultats de l'analyse thermodynamique dans la représentation par la MMTO, dans le domaine de la mammographie. Ainsi, par étapes successives, nous progresserons de la validation de ce procédé par l'analyse de signaux mono-fratals de une et deux dimensions permettant la validation et la calibration du programme waveplz, à la segmentation des parenchymes mammaires en deux catégories selon leur texture tel que l'expose Cadwell [1]. Cet exposé se terminera par l'étude des capacités et des limitations de ce procédé dans le cadre pour la pré-détection de micro-calcifications et de textures cancéreuses.

Dans ce chapitre, nous numéroterons les différents graphiques par un indice pour simplifier la présentation des divers résultats. Nous prendrons les conventions suivantes :

- Graphique 1 : le logarithme en base 2 de la fonction de partition $Z(a, q)$ par rapport au logarithme en base 2 de l'échelle a,

- Graphique 2 : le coefficient de Hölder $H(a, q)$ par rapport au logarithme en base 2 de l'échelle a,

- Graphique 3 : la dimension de Hausdorff $D(a, q)$ par rapport au logarithme en base 2 de l'échelle a,

- Graphique 4 : la dimension de Hausdorff $D(a, q)$ par rapport au coefficient de Hölder $H(a, q)$,

- Graphique 5 : la pente de la dimension de Hausdorff $d(q)$ par rapport à la pente du coefficient de Hölder $h(q)$,

- Graphique 6 : la pente du coefficient de Hölder $h(q)$ par rapport au logarithme en base 2 de l'échelle a,

- Graphique 7 : la pente de la dimension de Hausdorff $d(q)$ par rapport au logarithme en base 2 de l'échelle a,

- Graphique 8 : l'évolution des valeurs des maxima $M(a)$ des différentes lignes de maxima par rapport à l'échelle a.

Nous prendrons aussi comme convention que la variation de l'indice q prend les valeurs -4, -3, -2, -1, 0, +1, +2, +3 et +4. Les différentes valeurs ayant comme paramètre q seront toujours représentées par la même convention de symbole comme stipulé dans la figure 1.

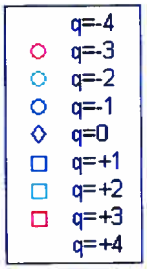


FIG. 1. Légende des symboles et des valeurs de q correspondantes

4.1. CALIBRATION SUR DONNÉES SIMULÉES

Afin de pouvoir vérifier l'exactitude des résultats obtenus, nous effectuons une série de tests sur des signaux 1D et 2D dont nous connaissons les caractéristiques (dimension Hausdorff et exposant de Hölder). Afin d'obtenir ces signaux, nous avons utilisé le programme Fraclab élaboré par l'équipe du projet FRACTALES de l'INRIA (logiciel téléchargeable gratuitement sur le site *http ://www.irccyn.ec-nantes.fr/herbergement/FracLab/*) qui permet la construction de signaux et d'images de type brownien fractionnaire.

La raison pour laquelle nous avons utilisé ici des browniens fractionnaires pour la vérification du fonctionnement de waveplz et son calibrage est que ces signaux possèdent un même coefficient fractral à travers toutes les échelles (mono-fractales, voir section 1.2). Cela nous a permis d'observer les résultats générés par waveplz et de les corriger le cas échéant.

Quelques exemples de browniens fractionnaires à deux dimensions se trouvent sur les figures 2, 3 et 4. On peut ainsi remarquer que le brownien fractionnaire de coefficient $h = 0.2$ semble plus rugueux que celui de coefficient mono-fractal $h = 0.9$.

Dans cette section, nous allons exposer les différents résultats obtenus du calcul du coefficient de texture avec waveplz. Nous commencerons donc par présenter le spectre des expériences effectuées en une dimension ainsi que leur analyse. Nous poursuivrons par des tests sur des browniens fractionnaires de deux dimensions de coefficients de Hölder (h) variant de 0.1 à 0.9. Nous finirons enfin cette partie sur l'analyse de ces résultats qui nous ont permis la calibration du système de visualisation de waveplz.

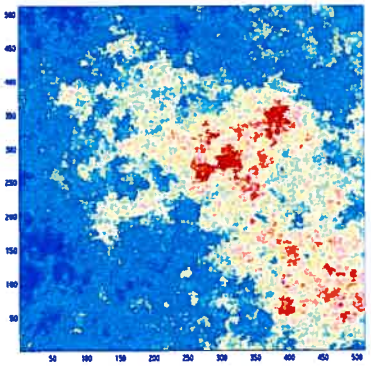


FIG. 2. Exemple d'un signal à 2 dimensions d'un brownien fractionnaire avec $h = 0.2$

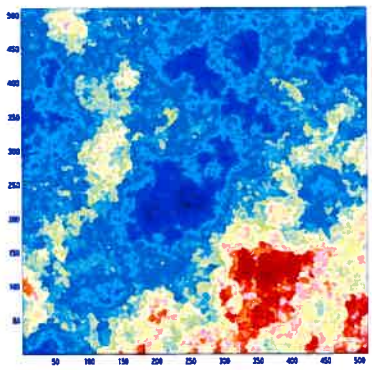


FIG. 3. Exemple d'un signal à 2 dimensions d'un brownien fractionnaire avec $h = 0.5$

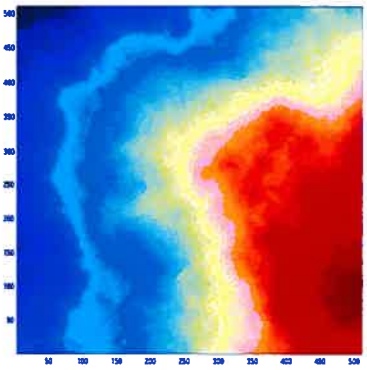


FIG. 4. étude d'un signal une dimension d'un brownien fractionnaire avec $h = 0.9$

4.1.1. Tests sur Browniens Fractionnaires à une dimension

D'après Arnéodo et al [**6**], les signaux que l'on peut étudier avec la MMTO doivent avoir certaines caractéristiques. Le signal artificiel doit avoir un coefficient de Hölder compris entre $h = 0.1$ et $h = 0.9$ pour que la méthode fonctionne correctement. L'analyse doit aussi être faite sur un minimum de 32 échelles, soit quatre octaves et sept voix, afin d'obtenir une statistique suffisante et des résultats probant [**22**].

Nous pouvons observer les résultats des tests effectués sur quelques signaux monofractals avec $h = 0.3$, $h = 0.5$ et $h = 0.9$. Les graphiques en résultant se trouvent dans les figures 5, 6 et 7.

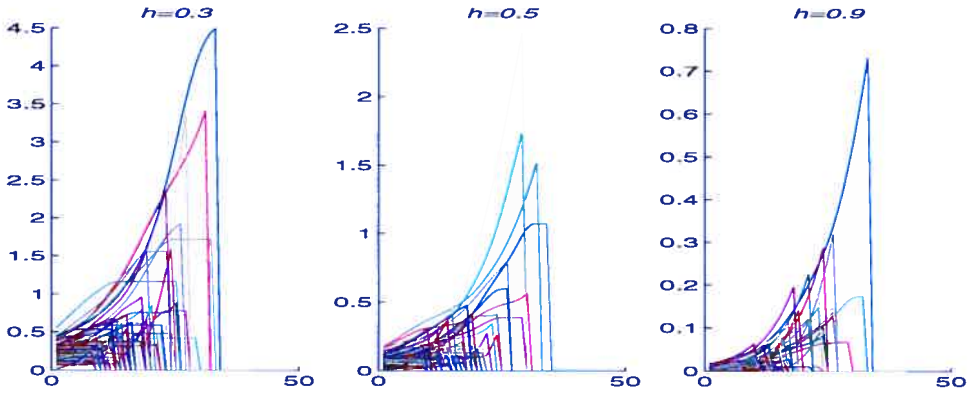


FIG. 5. Module du maxima $M(a)$ de chaque ligne en fonction de $log_2(a)$ d'un signal une dimension d'un brownien fractionnaire $h = 0.3, 0.5, 09$.

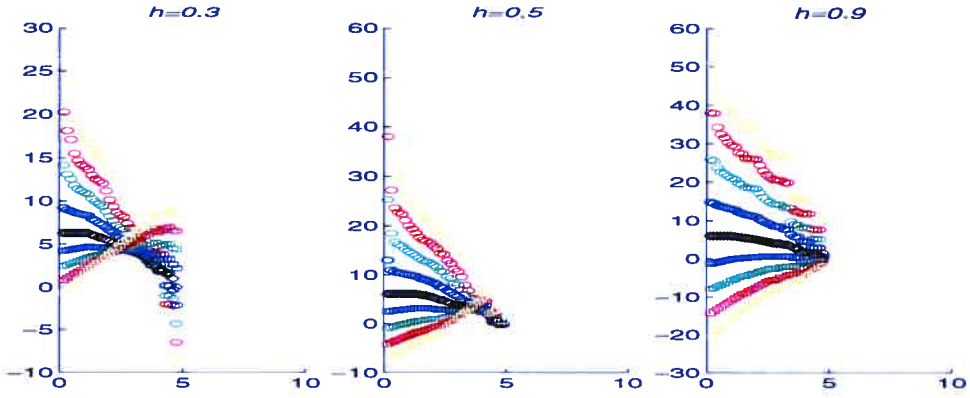


FIG. 6. Fonction de partition $Z(a.q)$ en fonction de $log_2(a)$ d'un signal une dimension d'un brownien fractionnaire $h = 0.3, 0.5, 0.9$.

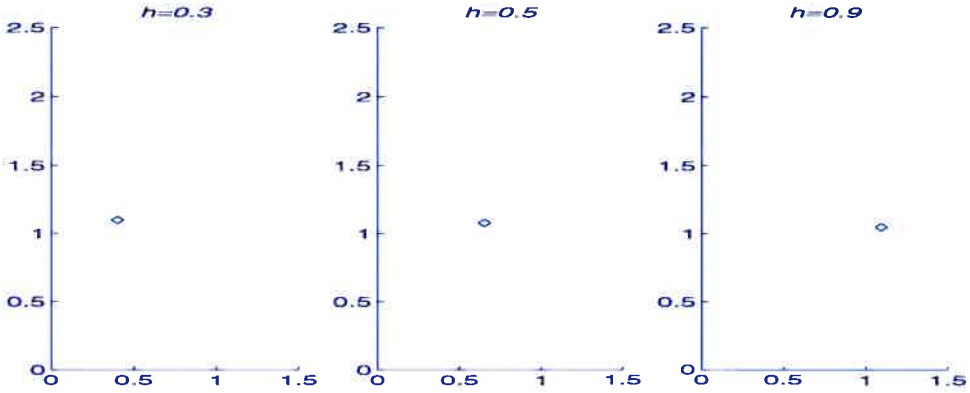


FIG. 7. Le coefficient de texture en fonction de la dimension de Hausdorff d'un signal une dimension d'un brownien fractionnaire $h = 0.3, 0.5, 0.9$.

Une première analyse des graphiques nous confirme le bon fonctionnement de waveplz dans le traitement des données. Nous constatons que la linéarisation des maxima pour conserver la plus grande valeur de maxima au sommet de la ligne (equation 60) se fait correctement comme nous le montre les graphiques 8 des figures 5.

Nous pouvons constater sur les figures 6 le comportement linéaire de la fonction de partition $Z(a, q)$ dans les échelles a.

En effectuant un balayage du spectre plus large pour des coefficients mono-fractals

allant de $h = 0.1$ à $h = 0.95$, nous observons que la valeur de la pente de coefficient de Hölder $h(q)$ pour les échelles a allant vers zéro a en moyenne une valeur supérieure de 0.125 ± 0.005 par rapport à ce que nous prévoyions (voir figures 7).

4.1.2. Tests sur Browniens Fractionnaires à deux dimensions

Maintenant que nous avons la confirmation du bon fonctionnement du programme pour des signaux artificiels à une dimension, abordons cette même série de tests pour des images mono-fractales. Comme pour le cas unidimensionnel, nous générons les browniens fractionnaires grâce au programme Fraclab. Afin de suivre les suggestions de Decoster et al **[22]**, nous utiliserons des signaux de 512 par 512 pixels en effectuant une analyse sur 32 échelles (4 octaves et 7 voix) afin d'avoir assez de statistiques. Nous limiterons le spectre de cette étude à l'analyse d'images de coefficient mono-fractal de $h = 0.1$ à $h = 0.9$ **[8]**.

Les graphiques des figures 8, 9 et 10 nous montrent les résultats que l'on obtient pour des browniens fractionnaires respectivement de valeur mono-fractale $h = 0.2$, $h = 0.5$ et $h = 0.9$.

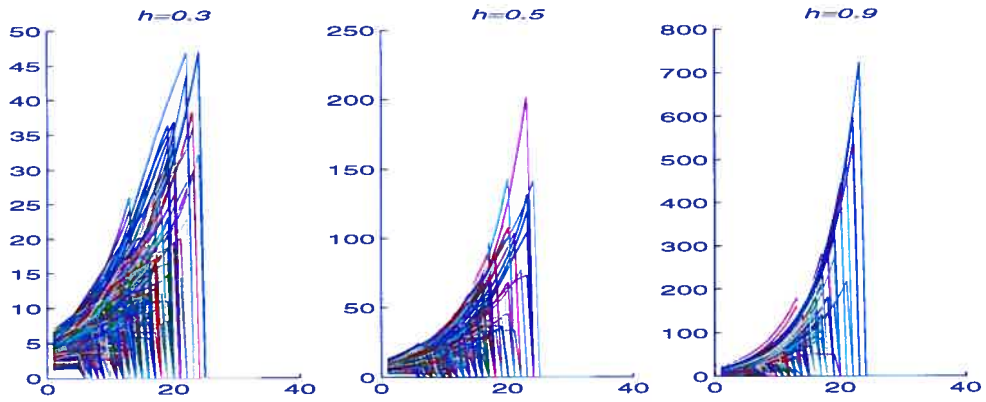


FIG. 8. Module du maxima $M(a)$ de chaque ligne en fonction de $log_2(a)$ d'un signal deux dimensions d'un brownien fractionnaire $h = 0.3, 0.5, 09$.

Ces différents graphiques nous montrent la stabilité du code pour une analyse thermodynamique bidimensionnelle, entre autre par la linéarité des fonctions de partition $Z(a, q)$ à travers les échelles a (voir figures 9).

Comme pour le cas unidimensionnel, le graphique 5 des figures 10 montre une variation du coefficient de texture $h(q)$ proportionnel au coefficient mono-fractal du signal. Nous constatons aussi l'addition systématique de la valeur 0.25 à chaque coefficient de texture.

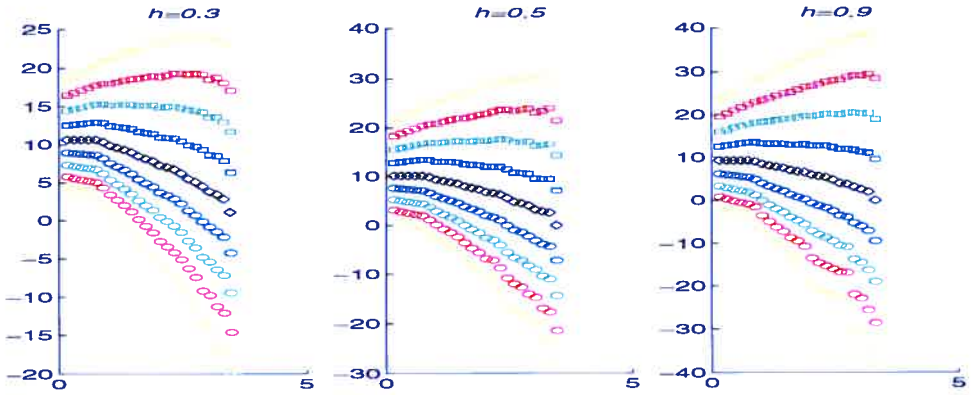


FIG. 9. Fonction de partition $Z(a.q)$ en fonction de $log_2(a)$ d'un signal deux dimensions d'un brownien fractionnaire $h = 0.3, 0.5, 0.9$.

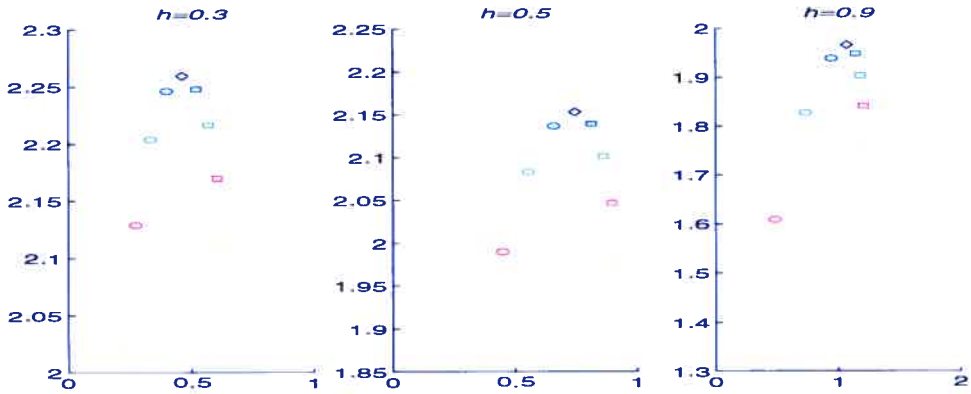


FIG. 10. Le coefficient de texture en fonction de la dimension de Hausdorff d'un signal deux dimensions d'un brownien fractionnaire $h = 0.3, 0.5, 09$.

4.1.3. Calibration

Grâce aux résultats obtenus dans la section précédente, nous pouvons maintenant calibrer les différents coefficients de texture que nous obtiendrons sur des images réelles de mammographie.

D'après les figures 7, nous avons constaté qu'une correction sur le coefficient de texture $\Delta_{1D} = 0.125 \pm 0.005$ était nécessaire pour l'analyse des signaux d'une dimension.

En utilisant les graphiques 5 des figures 10 de l'analyse d'images mono-fractales,

nous pouvons établir que la correction $\Delta_{2D} = 0.25 \pm 0.005$ vient s'additionner à la valeur réelle du coefficient de Hölder lorsque l'échelle a tend vers zéro. Les pentes des coefficients de Hölder et de la dimension de Hausdorff nous donne donc directement le coefficient de texture et la dimension multifractale du signal tels que décrits dans les équations 65 et 66.

Il est intéressant de noter, à ce niveau de l'analyse, le comportement thermodynamique des signaux artificiels étudiés. En effet, dans les graphiques 2, 3, 6 et 7 des figures 11,12 et 13, nous pouvons remarquer que pour des températures élevées (c'est-à-dire pour une valeur faible de q). Les signaux se comportent de manière quasi chaotique similairement à un gaz dont le mouvement moléculaire deviendrait erratique avec l'augmentation de la chaleur.

Nous pouvons ainsi adapter le système de visualisation pour obtenir directement la bonne valeur de texture afin de vérifier les résultats des travaux de Cadwell. Nous pourrons donc vérifier que les glandes mammaires se séparent en deux catégories : denses $(h = 0.65)$ et adipeuses $(h = 0.28)$.
Intéressons nous à présent à l'analyse d'images et de signaux réels, et observons si ce code nous permet de détecter aussi la présence de singularités dans les seins.

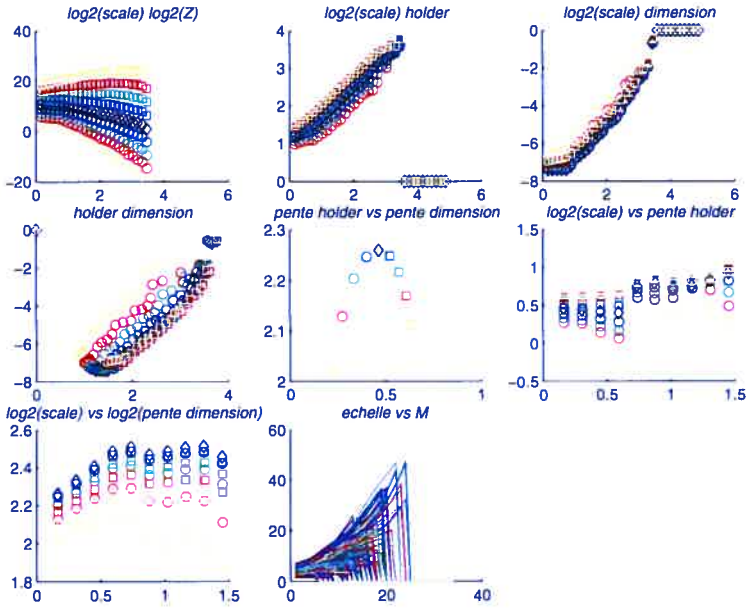


FIG. 11. étude d'un signal deux dimensions d'un brownien fractionnaire avec $h = 0.2$

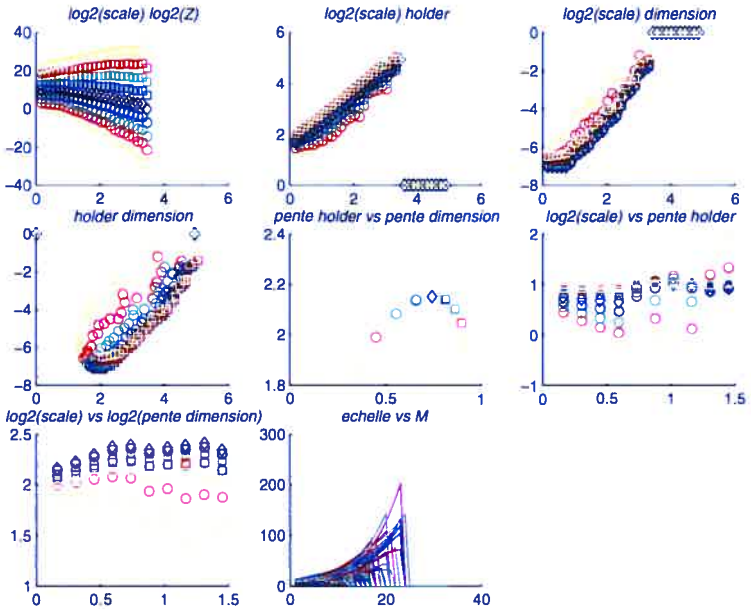


FIG. 12. étude d'un signal deux dimensions d'un brownien fractionnaire avec $h = 0.5$

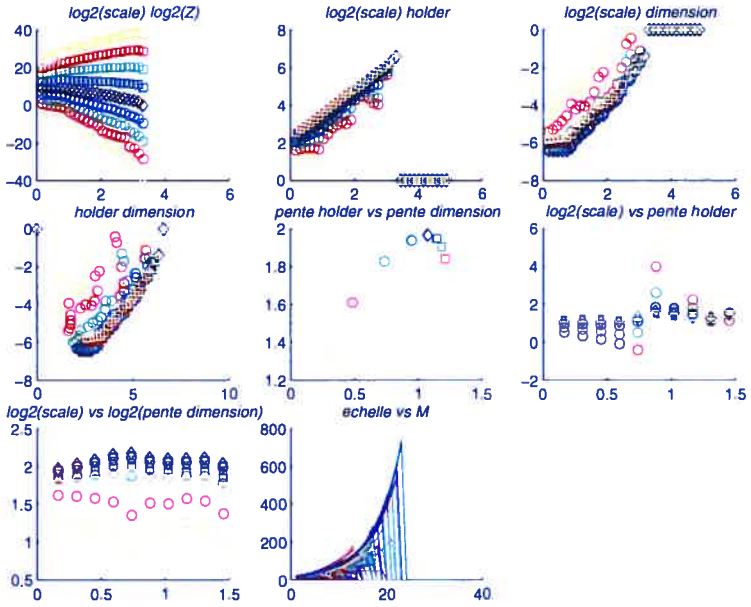


FIG. 13. étude d'un signal deux dimensions d'un brownien fractionnaire avec $h = 0.9$

4.2. APPLICATIONS SUR DONNÉES RÉELLES

4.2.1. Préparation des données réelles

La base de données de mammographie que nous allons utiliser pour nos tests malgré sa limitation par le nombre de patientes montre une grande diversité par

leurs caractéristiques et leurs historiques.

Cette banque de données se compose de quatre patientes. Nous possédons pour chaque année les mammographies céphalo-codales droite (CCD) et gauche (CCG) ainsi que les mammographies latérales droite (MLD) et gauche (MLG) sur diverses périodes de temps (2, 3, 7 et 10 ans).

Ces images radiographiques ont été obtenues par numérisation de films mammographiques. Cette méthode de numérisation indirecte a la caractéristique d'avoir un mauvais rapport signal sur bruit en comparaison à la numérisation directe (voir section 2.2). Cela nous permettra donc de vérifier la robustesse du code pour des signaux très bruités.

Initialement, les mammographies sont archivées en format de compression tiff avec une profondeur de 16 bits. Les programmes calculant les transformées en ondelettes et effectuant le chaînage des lignes de maxima ne traitant que le format numérique double, il nous a fallu reformater les images pour les adapter au programme au prix d'une perte de qualité.

Le passage des données de 16 bits à 8 bits de profondeur et leur stockage en format ASCII peuvent être effectués à l'aide d'un simple script matlab. Voici un exemple pour le traitement et le reformatage d'une mammographie nommée 400f4.tif.

```
x='400f4.tif';
a = imread(x);
a = double(a)/65535;
save('400f4.m','a','-ASCII')
```

En plus du reformatage, nous avons aussi normalisé les données pour que les valeurs soient comprises entre zéro et un. Avant ce traitement, les mammographies ont été réorientées pour avoir la partie la plus proche des poumons du côté droit afin d'optimiser le balayage effectué par waveplz. Chaque image a été renommée pour un classement plus efficace en fonction du numéro de la patiente, de l'année de la mammographie et du type de mammographie dont il s'agit comme par exemple l'image 400f4.tif.

Cette syntaxe vous permet de déterminer les caractéristiques de la mammographie. Le premier chiffre 4 représente le numéro identifiant la patiente. Les deux chiffres suivants permettent de retracer temporellement la mammographie, ici 00 signifie qu'il s'agit de la première année. f4 représente la CCG (f1 pour MLD, f2 pour MLG et f3 pour CCD). Ainsi à partir du nom 400f4.tif, vous pouvez déterminer qu'il s'agit d'une mammographie céphalo-codale gauche de la première

année d'examen de la patiente numéro 4.

La banque de mammographies ainsi disponible représente un total de 54 images pour un volume total de 4.5 Go de données séparées en 4 patientes dont les historiques vont de trois à onze années.

4.2.2. Evaluations temporelles et de stockage des résultats

Avant de lancer l'analyse de ces mammographies, nous devons évaluer le temps de calcul et l'espace disque nécessaire pour le traitement ainsi que le stockage des données et des résultats.

Pour effectuer cette évaluation, commençons par traiter une image simple de 512 par 512 pixels. Pour effectuer nos calculs nous utiliserons un ordinateur personnel du type Pentium 3 1.7 GHz avec 1.5 G de mémoire vive RAM et fonctionnant dans l'environnement Redhat 9.0. Pour une analyse sur 32 échelles d'une image composée de 262 144 pixels, il nous faut environ 15 minutes et une surface disque de 120 Mo de libre pour le traitement et 8Mo de surface pour le stockage des données et des résultats.

On peut estimer qu'une mammographie est décomposée, au maximum, en 50 images de 512 par 512 pixels. En utilisant donc le même type d'ordinateur, on peut estimer qu'une mammographie prendra au maximum un temps de 12 heures 30 minutes dont 11 heures 40 minutes ne serviront qu'au calcul des transformées en ondelettes et au chaînage. Le stockage de toutes les données inhérentes à ce calcul représente un volume total de 420 Mo pour une image de 80 Mo.

Il est donc important ici de considérer qu'à chaque mammographie 0.5 Go de données seront générées en environ 12 heures. Ce temps peut être réduit par l'utilisation de machines plus puissantes, et diminuer la surface de stockage nécessaire en effaçant successivement les fichiers générés par les programmes de transformation en ondelettes.

Notre banque de données complète utiliserait donc moins de 26 Go de surface disque au final pour 26 jours de calcul machine. Suite à un manque de surface disque disponible, seul une dizaine de mammographies complètes a pu être traitées.

4.2.3. Evaluation de la texture mammaire

Au début des années 90, Caldwell et al développèrent un système de caractérisation des tissus du parenchyme mammaire par l'usage d'un système autonome et automatisé basé sur le calcul de la dimension fractale [1].

Dans cette optique, Waveplz a été développé afin de distinguer différents types de textures fractales par l'application des concepts thermodynamiques à la méthode des maxima des transformées en ondelettes.

Le but de cette partie est d'évaluer la stabilité de waveplz pour la caractérisation des tissus sur des données réelles en commençant par les cas classiques de textures denses et adipeuses.

Nous avons donc étudié le cas d'un sein sain en utilisant la CCD de la patiente 1 à l'année 7. Les résultats corrigés selon les observations faites à partir des images synthétiques (c'est-à-dire en enlevant 0.25 à chaque coefficient de texture. Les aboutissements de ces calculs et approximations se trouvent résumés sous forme graphique dans la figure 14 et sous forme numérique dans le tableau 1.

Nous pouvons constater à la vue de ces résultats que le tissu adipeux caractéristique représenté dans le bloc L7-C5 (Ligne 7 - Colonne 5) possède effectivement, selon les constatations de Kestener et al [**13**], une texture de 0.6688, correspondant ainsi à la fourchette de $h = 0.65 \pm 0.1$ fixée.

Les tissus denses sont plus rares dans cette mammographie et vu leur proximité des bords du sein, nous pourrons prendre le bloc L4-C2 comme représentatif d'une texture dense. Nous pouvons aussi constater que ce bloc de coefficient de texture $h = 0.3224$ est inclus dans la fourchette de $h = 0.3 \pm 0.1$ fixée par Kestener et al [**13**].

Les premiers résultats obtenus sur cette mammographie permettent donc de confirmer le bon fonctionnement de Waveplz sur des données réelles et la confirmation des travaux de Caldwell. Nous pouvons cependant dès à présent constater l'hétérogénéité de la texture mammaire grâce à la variation du coefficient de texture à travers la mammographie. Nous pouvons aussi percevoir la forte influence des bords du sein sur les calculs.

Etudions l'évolution de cette texture à travers le temps afin de pouvoir commencer à établir un système automatique de détection des foyers cancéreux et des micro-calcifications.

4.2.4. Evolution temporelle de la texture mammaire

Notre base de données nous permet l'étude plus précise du comportement de la texture du sein au cours du temps sur 4 patientes. Nous avons donc effectué le calcul de texture sur la partie centrale des mammographies, correspondant à la partie la moins sujette à des changements et des variations texturales. Les images extraites font 512 sur 512 pixels normalisées entre zéro et un. L'étude comme précédemment s'est faite sur 32 échelles.

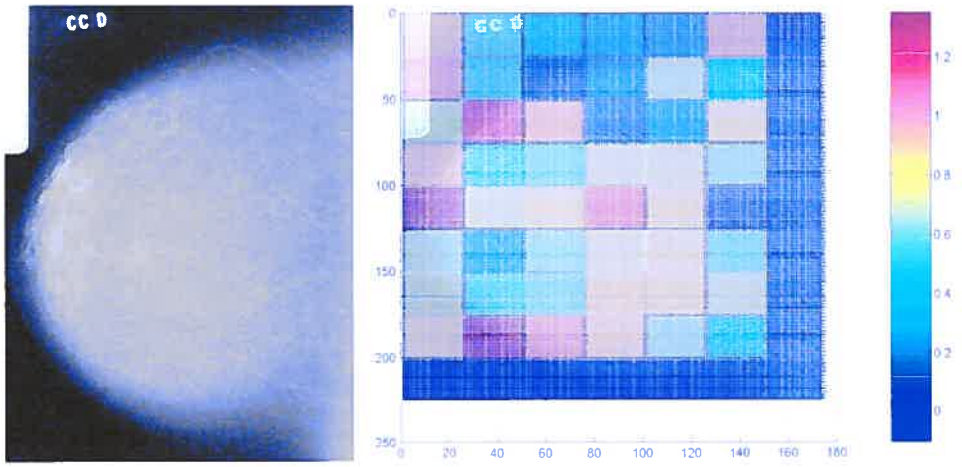


FIG. 14. CCD de la patiente 1 à l'année 7.

0.7488	0.3984	0.2395	0.1608	0.0862	0.7888	0
0.8367	0.3693	0	0.0674	0.5254	0.2537	0
0.5114	1.0753	0.8094	0	0.0957	0.6594	0
0.7510	0.3224	0.4128	0.5867	0.5945	0.4808	0
1.1971	0.6072	0.6463	0.9453	0.6683	0	0
0.5148	0.1034	0.4572	0.6229	0.6333	0.4227	0
0.5320	0.3663	0.4265	0.6795	0.6688	0.5436	0
0.6969	1.1709	0.8888	0.7233	0.4842	0.3281	0
0	0	0	0	0	0	0

TAB. 1. Coefficients de texture du CCD de la patiente 1 pour l'année 07

Les résultats de cette recherche sont recensés par patiente dans les tableaux 2, 3, 4 et 5. La première ligne de chaque tableau correspond à l'année de la prise de la mammographie, chaque colonne de chiffres correspond au coefficient de texture correspondant successivement aux mammographies CCD, CCG, MLD et MLG. La mention AMD signifie qu'aucune mammographie n'était disponible.

Il est intéressant de constater, suite à l'analyse de ces mammographies, la variation de l'indice de la texture mammaire au cours des années. L'évolution structurelle de la glande mammaire étant directement liée à la période d'activité génitale de la femme, il était à prévoir que le comportement temporel de la texture du sein ne serait pas constant voire même linéaire. Nous comprendrons ici par comportement linéaire un changement de structure passant, par exemple, d'une texture rugueuse à une surface lisse comme un rocher poli par le courant d'une rivière. On peut cependant percevoir pour certains cas une constance de la texture

00	02	06	07	10
0.1825	0.1915	0.2533	0.3386	0.3612
0.1804	0.2130	0.3004	0.3226	0.3231
0.1868	0.2575	0.4409	0.3488	0.3650
0.2847	0.2372	0.3176	0.3450	0.4508

TAB. 2. Coefficients de texture de CCD CCG MLD MLG de la patiente 1 pour les années 00, 02, 06, 07 et 10

00	02	04	07
0.2898	0.3338	0.3423	0.2850
0.2981	0.3087	0.3476	AMD
0.3326	0.2852	0.4281	0.4579
0.2869	0.3057	0.3083	0.2928

TAB. 3. Coefficients de texture de CCD, CCG, MLD, MLG de la patiente 2 pour les années 00, 02, 04 et 07

00	02
0.4283	AMD
0.3675	0.2956
0.3609	0.3724
0.3539	0.3899

TAB. 4. Coefficients de texture de CCD CCG MLD MLG de la patiente 3 pour les années 00 et 02

00	03
0.3002	0.3638
0.2890	0.4302
0.2095	0.3614
0.3190	0.3960

TAB. 5. Coefficients de texture de CCD CCG MLD MLG de la patiente 4 pour les années 00 et 03

pour une même année à travers les angles de prises de vue et de manière bilatérale.

Certain de ces résultats pourraient cependant être corrompus par la présence de masses cancéreuses (caractérisées par une texture plus dense par rapport au tissu environnant) ou de micro-calcifications.

4.2.5. Détection de textures cancéreuses et de micro-calcifications

Afin de vérifier le bon fonctionnement de waveplz pour la détection de singularité dans la structure mammaire, utilisons la CCG de la patiente 1 à l'année 10 (voir figure 15).

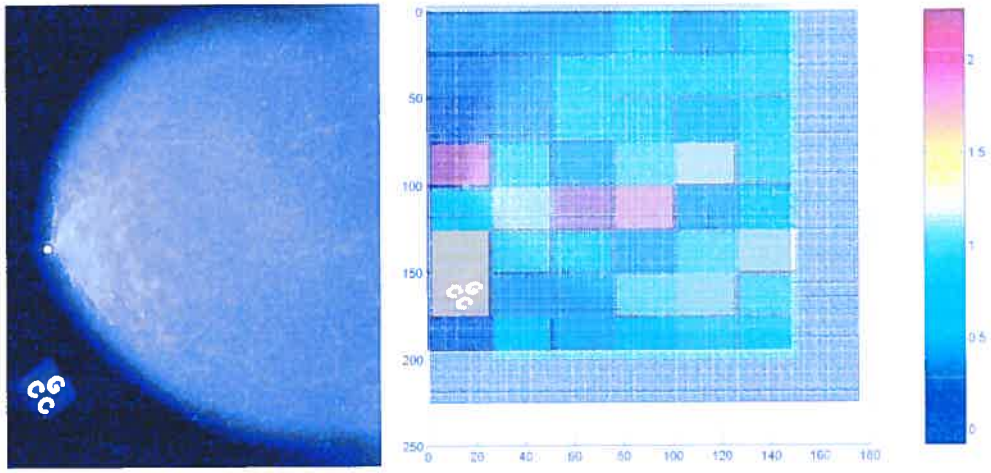


FIG. 15. CCG de la patiente 1 à l'année 10.

A la vue des résultats de cette mammographie, on porte plus précisément notre attention sur les blocs L4-C1, L4-C5, L5-C2, L5-C3 et L5-C4 qui ont des coefficients plus élevés que la moyenne globale. Les blocs sont représentés dans la figure 16.

On constate ainsi la présence dans les blocs de plus forts coefficients de texture la présence de micro-calcifications de différents types y compris le marqueur radiographique du bloc L4-C1.

Observons maintenant cette mammographie pour les années précédentes (voir figures 17, 18, 19 et 20).

On constate déjà la présence des micro-calcifications des blocs L5-C2 et L5-C3 de l'année 10 dans la mammographie 17 de l'année 7 dans le bloc L5-C2 ainsi que des variations de la densité de la texture mammaire au cours des années (voir figures 18, 19 et 20).

Sur le cas précis de cette patiente, nous pouvons constater l'efficacité du logiciel Waveplz. En effet, il permet de localiser la création et la formation de micro-calcifications avant leur apparition sur la mammographie. Des tests sur une base de données plus large semblent se justifier afin de vérifier la stabilité de l'outil de caractérisation de texture.

4.3. Discussion

Nous avons montré dans ce chapitre une des applications possibles de la MMTO et des principes thermodynamiques pour l'évaluation de coefficients de texture ainsi que le calcul de la dimension multifractale sur des signaux de une et deux dimensions. Il nous a aussi été possible de montrer que ces concepts peuvent être utilisés pour la caractérisation de structures étrangères à un corps telles que des tissus de forte densité ainsi que des micro-calcifications.

Le programme waveplz a présenté sa capacité à distinguer des signaux multifractals synthétiques pour des valeurs de Hölder allant de $h = 0.1$ à $h = 0.9$ et ceci pour une et deux dimensions.

Il a par la suite été possible de confirmer les travaux de Caldwell [1] pour des images réelles telles que des mammographies en montrant, dans certaines limites, les caractéristiques multifractales de certains tissus humains. Nous avons réussi à distinguer numériquement les tissus denses avec $h = 0.65 \pm 0.1$ des tissus adipeux $h = 0.3 \pm 0.1$. Ce genre de résultats devrait permettre de mieux comprendre l'évolution de la structure mammaire au cours du temps.

Le logiciel Waveplz a prouvé sa capacité à signaler la présence de singularités telles que des micro-calcifications dans une structure complexe. Cette détection de corps étrangers semble aussi être possible au tout début de la formation de cet objet. Il nous faudra cependant effectuer de nouveaux tests sur des cas cliniques divers afin de vérifier la robustesse du programme et sa capacité a distinguer ces mêmes structures malines dans des seins plus denses.

Toutefois, nous avons constaté les limitations de waveplz à distinguer des tissus cancéreux. Ceci étant essentiellement dû à la structure hétérogène de la glande mammaire. La détection automatique et unilatérale de tissus malins est donc fortement limitée.

Cette recherche nous aura donc permis de montrer les capacités et les limitations de la détection de texture par la méthode thermodynamique appliquée aux ondelettes continues. Cela nous aura aussi permis une meilleure compréhension de la répartition des différents tissus dans la glande mammaire ainsi que son évolution au cours du temps.

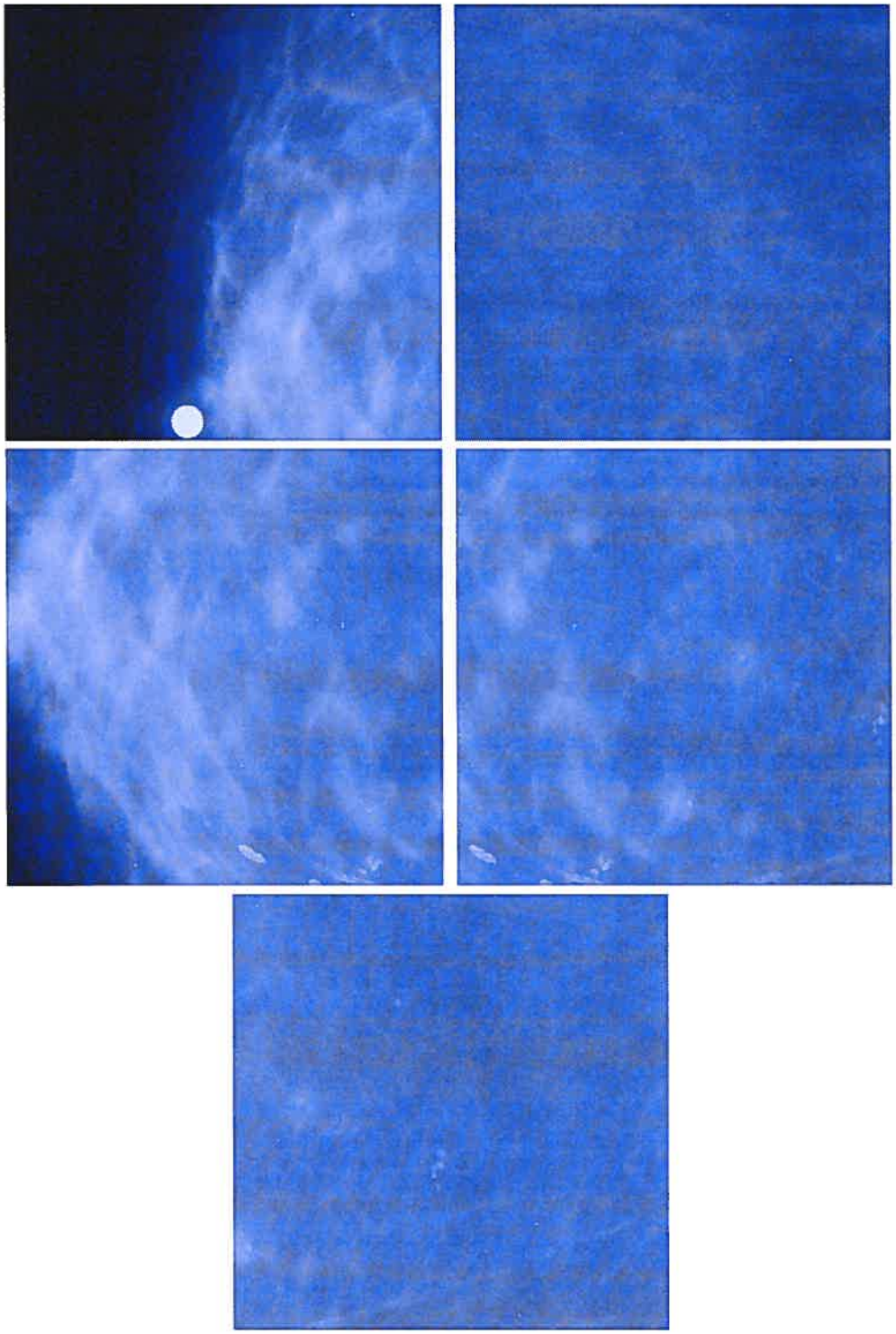


FIG. 16. Gros plans des singularités de la CCG de la patiente 1 à l'année 10, respectivement en partant du haut à gauche L4-C1, L4-C5, L5-C2, L5-C3 et L5-C4

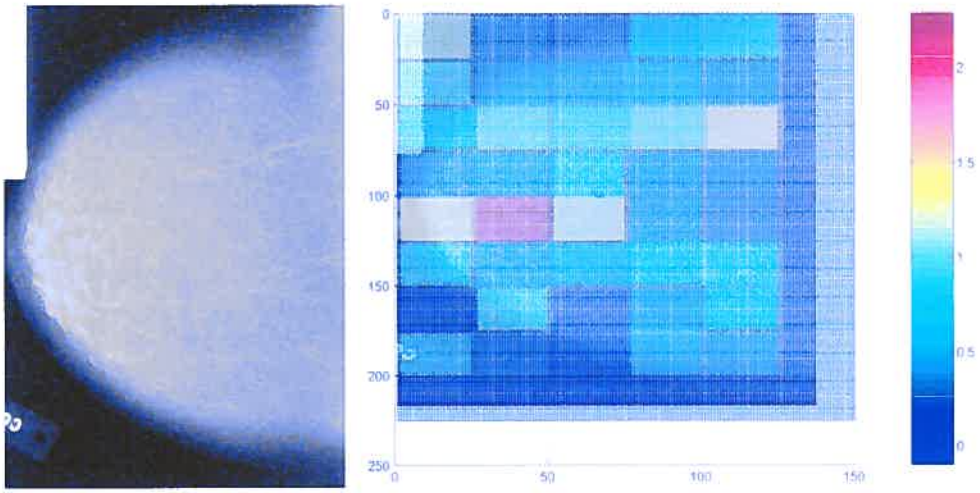


FIG. 17. CCG de la patiente 1 à l'année 7.

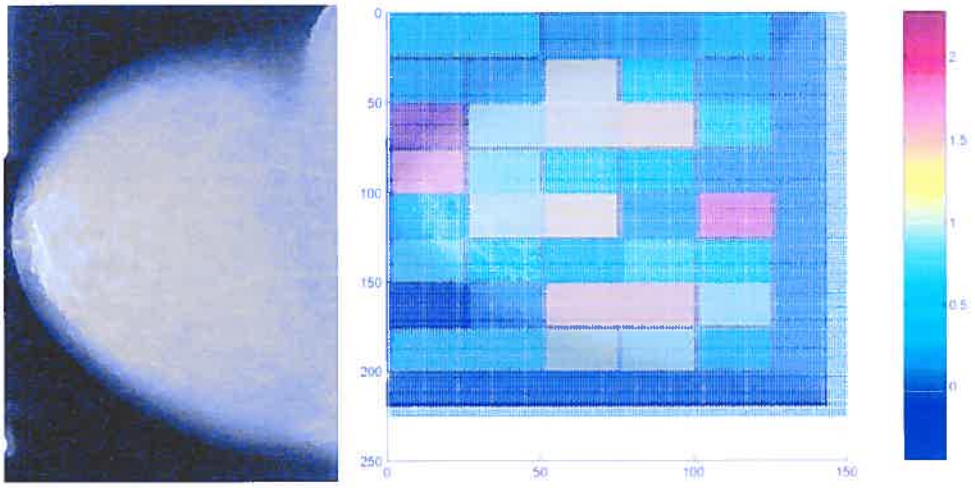


FIG. 18. CCG de la patiente 1 à l'année 6.

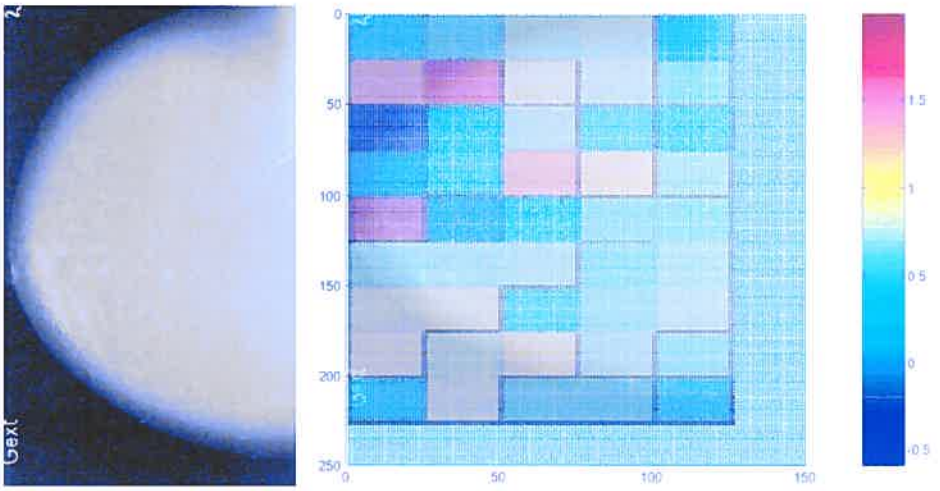


FIG. 19. CCG de la patiente 1 à l'année 2.

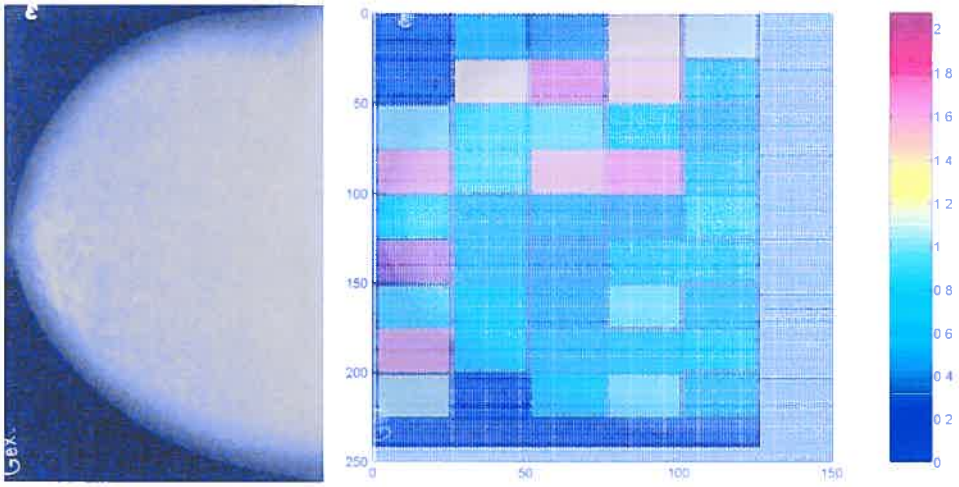


FIG. 20. CCG de la patiente 1 à l'année 0.

CONCLUSION

Le cancer du sein est la première cause de décès chez les femmes dans les pays industrialisés. Le but de cette recherche a été la programmation d'un logiciel permettant la caractérisation de textures cancéreuses dans les mammographies basée sur l'analyse thermodynamique de mesures multifractales.

Dans le premier chapitre nous avons expliqué la lente progression qui a mené la notion d'entropie (symbolisant la notion de réversibilité) à la notion de manque d'information par Shannon. De ces considérations thermodynamiques, nous avons établi les bases de fractales, d'analyses multifractales et de transformées en ondelettes qui ont permis l'invention de la Méthode des Maxima des Transformées en Ondelettes (MMTO). Cette méthode permet ainsi la caractérisation de textures dans toutes sortes de signaux.

Dans le second chapitre, nous avons apprécié la nature propre de la texture de la mammographie. Dans un premier temps, nous avons étudié l'origine de cette texture par l'analyse de la structure de la glande mammaire ainsi que les pathologies pouvant perturber la caractérisation des tissus denses et adipeux. Nous avons ensuite analysé la méthode de production d'une mammographie digitalisée et les rapports signal sur bruit qu'ils impliquent.

Dans le troisième chapitre, nous avons décrit l'implantation numérique de la méthode d'analyse de textures. Nous avons décrit l'algorithme de fonctionnement du logiciel Waveplz, les formats des fichiers analysés et sortant ainsi que les différentes méthodes de visualisation des résultats.

Dans le dernier chapitre, nous avons vérifié le bon fonctionnement du logiciel par l'étude de signaux mono-fractals artificiels et ainsi faire la calibration de l'outil. Nous avons terminé cette section par l'analyse de mammographie. A cette occasion nous avons pu vérifier les travaux de Cadwell par la caractérisation des tissus denses et adipeux que possède le sein. Nous avons aussi pu constater la variation de la densité du sein des patientes au fil des années. Nous avons constaté, à cette occasion, la constance du coefficient de texture du sein à un même endroit de manière bilatérale. Le logiciel Waveplz s'est montré particulièrement performant dans la détection de micro-calcifications dans les divers cas que compose notre base de données. Il a aussi montré sa capacité dans la localisation de micro-calcifications environ 6 ans avant leur visibilité sur la mammographie.

De plus en plus le regroupement de différentes sciences telles que la physique, les mathématiques et l'informatique permet de résoudre des problèmes nécessitant différents niveaux d'expertise dans des domaines variés. C'est dans cette voie de pluridisciplinarité que les recherches de cette maîtrise se sont orientées en appliquant des principes thermodynamiques divers aux ondelettes continues afin de produire un outil informatique permettant l'analyse de textures en mammographie.

La révision de la thermodynamique des fractals avec les ondelettes par Arnéodo et al [**6**] a permis de nouvelles utilisations de l'analyse temps-échelle par les ondelettes continues.

La Méthode de Maxima des Transformées en ondelettes (MMTO) a vu très rapidement son spectre d'application s'étendre à des domaines variés allant des marchés boursiers aux séquences d'ADN [**8**].

Le développement d'un outil informatique basé sur ces théories semblait donc une étape logique. Grâce à la flexibilité des langages C, C++ et python, des librairies et des programmes ont pu être développés selon le type de signaux que l'on cherche à étudier. Waveplz peut ainsi faire une étude précise aussi bien d'une onde unidimensionnelle que de la dissection d'une image pour en étudier sa composition texturale.

L'application de cet outil informatique à la mammographie nous a permis la caractérisation de textures saines comme les tissus adipeux ou les structures mammaires denses. Il nous a ainsi permis de vérifier la nature multifractale de certains de ces objets en leur attribuant une valeur de rugosité numérique, $h = 0.3 \pm 0.1$ pour les tissus adipeux et de $h = 0.65 \pm 0.1$ pour les tissus denses. Cela nous a permis de modéliser sur la valeur de structures plus denses et ainsi de vérifier ces résultats. Ainsi nous avons pu effectuer la détection de micro-calcifications et de textures cancéreuses dans les tissus hétérogènes mammaires.

On peut s'attendre à de nettes améliorations de ce processus de détection de textures malines par comparaison bilatérale. Par extrapolation des mammographies Céphalo-Caudales et latérales, on pourrait peut être effectuer une localisation en trois dimensions des masses suspectes. Les connaissances et l'expertise de médecins pourraient être mises en valeur afin de créer une base de données permettant à l'ordinateur de développer sa propre expertise et ainsi établir une relation entre texture (rugosité) locale et la pathologie déjà détectée.

La mise en avant d'images radiologiques digitales de meilleures précisions favoriserait la précocité du diagnostic de petits cancers. Elle ne remplacera toute

fois pas l'expertise du médecin dont la compétence et l'expérience restent primordiales dans l'analyse des données et l'établissement du diagnostic.

Il reste toute fois à signaler que la contrepartie de cette amélioration de la qualité des images mammographiques reste qu'elle multiplie le risque d'opérer à tort des patientes. En effet, vu l'augmentation de la précision des images obtenues, Les médecins devront repenser leur diagnostique afin d'éviter des erreurs telles que l'amalgame entre tumeurs et simples remaniements architecturaux de la glande mammaire.

Annexe A

MODÈLE DE WAVE.I

```
Wavelet2D
Border Effect : CV1D_MIRROR
Convolution Type : DIRECT_CONV
Derivation order : 2
Norm exponent : 0.5000
Form :  REAL
Omega_max : 7
t_max : 7
Number of octave : 4
Number of Vox : 5
First scale : 1.12065
Last scale : 36.3571
Signal size : 262144
Signal.nbcols : 512
Signal.nblines : 512
It : -1
```

Annexe B

CODE DE WAVEPLZ.PY

```
import sys, os, string, fpformat, os.path, dircache, shutil, time
import math, Tkinter
from string import split
from Tkinter import *
sys.path.append('C:\Program Files\python\Lib\site-packages\Numeric')
import Numeric
from Numeric import *
from FileDialog  import *

#
# waveplz 2.2
# Written by: Gael Sitzia <sitzia@crm.umontreal.ca>
# Last revision: amelioration de la gestion de la memoire
#


class bLoad:

    def __init__(self, name):

        X = self.words(name)
        Y = self.lines(name)
        Data = self.loadData(name,X,Y)
        Min = self.min(Data,X,Y)
        Max = self.max(Data,X,Y)
        self.info = Data,X,Y,Min,Max

    def Info(self):

        return self.info

    def min(self,data,X,Y):
```

```
    min=1
    levels = X
    nbLines = Y
    Rootemp = data
    min = Rootemp[0,0]

    for i in range(nbLines):
        for j in range(levels):
            temp = Rootemp[j,i] - min
            if (temp < 0) :
                min = Rootemp[j,i]
    return min

def max(self,data,X,Y):

    max=1
    levels = X
    nbLines = Y
    Rootemp = data
    max = Rootemp[0,0]

    for i in range(nbLines):
        for j in range(levels):
            temp = Rootemp[j,i] - max
            if (temp > 0) :
                max = Rootemp[j,i]
    return max

def loadData(self,name,X,Y):
    Rootemp = zeros([X,Y])
    #print 'X',X,' Y',Y
    Rootemp = Rootemp.astype(Float64)
    lines = 0
    words = 0
    for line in TextFile(name):
        lines = lines +1
        words = len(split(line))
        temp = split(line)
        for x in range(words):
            #print 'lines-1=', (lines-1)
            #print 'x=', x
            Rootemp[x,(lines-1)] = float(temp[x])
    return Rootemp
```

```
    def lines(self,name):
        lines = 0
        #calcul de la hauteur du tableau de donnees
        for line in TextFile(name):
            lines = lines + 1
        return lines

    def words(self,name):
        lines = 0
        words = 0
        max = 0
        #calcul de la largeur max du tableau de donnees
        for line in TextFile(name):
            words = len(split(line))
            if (max - words)<0:
                max = words
        return max


class TextFile:

    def __init__(self, filename, mode = 'r'):
        if string.find(filename, ':/') > 1: # URL
            if mode != 'r':
                raise IOError, "can't write to a URL"
            import urllib
            self.file = urllib.urlopen(filename)
        else:
            filename = os.path.expanduser(filename)
            if mode == 'r':
                if not os.path.exists(filename):
                    raise IOError, (2, 'No such file or directory: '
                                    + filename)
                if filename[-2:] == '.Z':
                    self.file = os.popen("uncompress -c " + filename
, mode)
                elif filename[-3:] == '.gz':
                    if gzip is None:
                        self.file = os.popen("gunzip -c " + filename
, mode)
                    else:
                        self.file = gzip.GzipFile(filename, 'rb')
```

```
                elif filename[-4:] == '.bz2':
                    self.file = os.popen("bzip2 -dc " + filename, mode)
                else:
                    try:
                        self.file = open(filename, mode)
                    except IOError, details:
                        if type(details) == type(()):
                                details = details + (filename,)
                        raise IOError, details
            elif mode == 'w':
                if filename[-2:] == '.Z':
                    self.file = os.popen("compress > " + filename, mode)
                elif filename[-3:] == '.gz':
                    if gzip is None:
                        self.file = os.popen("gzip > " + filename, mode)
                    else:
                        self.file = gzip.GzipFile(filename, 'wb')
                elif filename[-4:] == '.bz2':
                    self.file = os.popen("bzip2 > " + filename, mode)
                else:
                    try:
                        self.file = open(filename, mode)
                    except IOError, details:
                        if type(details) == type(()):
                                details = details + (filename,)
                        raise IOError, details
            elif mode == 'a':
                if filename[-2:] == '.Z':
                    raise IOError, (0, "Can't append to .Z files")
                elif filename[-3:] == '.gz':
                    if gzip is None:
                        self.file = os.popen("gzip >> " + filename, "w")
                    else:
                        self.file = gzip.GzipFile(filename, 'ab')
                else:
                    self.file = open(filename, mode)
            else:
                raise IOError, (0, 'Illegal mode: ' + repr(mode))

        def __del__(self):
    self.close()

        def __getitem__(self, item):
```

```
line = self.file.readline()
if not line:
    raise IndexError
return line

    def readline(self):
return self.file.readline()

    def readlines(self):
return self.file.readlines()

    def write(self, data):
self.file.write(data)

    def writelines(self, list):
for line in list:
    self.file.write(line)

    def close(self):
self.file.close()

    def flush(self):
        self.file.flush()


class SauveInfo:

    def readMatrix(filename):
        rows = []
        for line in TextFile(self,filename):
            columns = []
            for number in string.split(line):
                columns.append(string.atof(number))
            rows.append(columns)
        return Numeric.array(rows)

    def writeMatrix(self,a, filename):
        file = TextFile(filename, 'w')
        for line in a:
            for number in line:
                file.write(`number` + ' ')
            file.write('\n')
        file.close()
```

```
class FileChoose:
    def __init__(self):
        #choix du fichier de donnee qui est stocke dans filename
        self.parent = Tk()
        Button(self.parent, text="Choix du fichier de data a traiter")
.pack()
        self.filename = LoadFileDialog(self.parent).go('../../',"*.m")
        print 'les datas vont etre trait/ du fichier ', self.filename


    def getFilename(self):
        return self.filename

    def cancel(self, event=None):

        # put focus back to the parent window
        self.parent.focus_set()
        self.destroy()


class PrepaWaveplz:
    def __init__(self):

        self.sauve=SauveInfo()

        test = os.name
        self.platform = os.name    #type de platforme


        if test == 'posix':
            self.dirpath = os.getcwd() #donne le path courant

        if ((test == 'nt') | (test == 'dos')| (test == 'mac')):
            self.dirpath = 'waveplz22' #donne le path courant


        self.dirwork = os.path.join(self.dirpath,'work')
        self.direxec = os.path.join(self.dirpath,'exec')

        self.fichsort1D = ('data.m','wave.i','bM.m','bZ.m','bD.m','bH.m',
'bpented.m','bpenteh.m','bScale.m','matlabroot.m','Is.m','Js.m','roots.dat',
```

```
'extremas.m','scale.m','exact_zone.m','data_scl.m','filtersView.m')

        self.fichsort2D = ('data.m','wave.i','bM.m','bZ.m','bD.m','bH.m',
'bpented.m','bpenteh.m','bScale.m','matlabroot.m','Is.m','Js.m','Ks.m',
'roots.dat','scale.m','MatriceRoots.dat','Filteredroots.dat')

        #trsfert sous dirwork
        os.chdir(self.dirwork)

        #choix du fichier de donnee a traiter
        ask1 = FileChoose()
        self.data = ask1.getFilename()
        #ask1.cancel()
        self.datapathsplit = os.path.split(self.data)
        self.dataname = self.datapathsplit[1]
        self.dataname = string.replace(self.dataname,'.','_')

        ##creer une dir en fonction de l horloge interne
        timefl = time.time()
        timeint = int(timefl)
        temp =  self.dataname + `timeint`
        self.dirwrktmp = os.path.join(self.dirwork,temp)
        os.mkdir(self.dirwrktmp)
        os.chdir(self.dirwrktmp)

        #creer la dir temp
        self.dirtampon = os.path.join(self.dirwork,'temp')
        os.mkdir(self.dirtampon)
        #os.chdir(self.dirwork)

        #sauvegarde des donnees dans un fichier data.m dans dirwrktmp/
        src = os.path.join(self.data)
        dest = os.path.join(self.dirwrktmp,'data.m')
        shutil.copyfile(src,dest)

        #calcul de la taille de l image a gerer
        self.nbLigne = int(self.nblin(self.data))
        self.nbCol = int(self.nbcol(self.data))

        #copie des exec du step 0 de /exec/step0  dans la dir
#tampon(xxx+1)
        src = os.path.join(self.direxec,'step0')
        dest = os.path.join(self.dirwrktmp)
```

```
        liste =os.listdir(src)
        #print 'step0', liste
        for i in range(len(liste)):
            srci = os.path.join(src,liste[i])
            desti = os.path.join(dest)#,liste[i])
            shutil.copy(srci,desti)

        #copie des exec du step 0 de /exec/step1  dans la dir
#tampon(xxx+1)/temp
        src = os.path.join(self.direxec,'step1')
        dest = os.path.join(self.dirtampon)
        liste =os.listdir(src)
        #print 'step1', liste
        for i in range(len(liste)):
            srci = os.path.join(src,liste[i])
            desti = os.path.join(dest)#,liste[i])
            shutil.copy(srci,desti)

        #stockage des donnees
        self.MasterData = self.remplissage(self.data,self.nbLigne,
self.nbCol)

        #preparation de wave.i
        if test == 'posix':
            os.popen('python guiWvplz.py')
        else:
            os.startfile ('guiWvplz.py')

        #copie du wave.i cree dans le repertoire tampon(xxx+1)/temp
        src = os.path.join(self.dirwrktmp,'wave.i')
        dest = os.path.join(self.dirtampon)#,'wave.i')
        shutil.copy(src,dest)

        #initialisation de la liste des dir a traiter
        listeDir = []

        #test signal 2D:
        if ((self.nbLigne - 1) > 0) & ((self.nbCol - 1) > 0) :

            #test signal > 512*512
            if ((self.nbLigne - 512) > 0) | ((self.nbCol - 512) > 0) :
                Nx = int(self.nbLigne/256)
                Ny = int(self.nbCol/256)
```

```
                    if Nx<1:Nx=1
                    if Ny<1:Ny=1

                    for i in range(Nx-1):
                        for j in range(Ny-1):

                            #creation  dir ij format 00 00
                            tempname=self.repNxNy(i,j)
                            dest = os.path.join(self.dirwrktmp,tempname)
                            os.mkdir(dest)

                            #transfert dans dir tempxxxx/temp
                            os.chdir(dest)

                            #stockage des dir de travail
                            listeDir = listeDir + [dest]

                            #creation du data.m situe en ij dans le masterdata
                            datatemp=self.extractData(self.MasterData,
self.nbLigne,self.nbCol,i,Nx,j,Ny)
                            self.sauve.writeMatrix(datatemp,'data.m')

                            #transfert sous la directorie tempxxxxx/
                            os.chdir(self.dirwrktmp)


                else : #image de 512*512 ou moins

                    #creation  dir ij format 000 000
                    tempname='000000'
                    dest = os.path.join(self.dirwrktmp,tempname)
                    os.mkdir(dest)

                    #transfert dans dir tempxxxx/temp
                    os.chdir(dest)

                    # stockage des dir de travail
                    listeDir = listeDir + [dest]

                    #creation du data.m situe en ij dans le masterdata
                    datatemp=self.regulData2D(self.MasterData,self.nbLigne,
self.nbCol)
```

```
                    self.sauve.writeMatrix(datatemp,'data.m')

                    #remonte d 1 a tempxxxxx/
                    #transfert sous la directorie tempxxxxx/
                    os.chdir(self.dirwrktmp)



            #le signal 1D
            else:
                #test signal >  1024*1 ou 1*1024
                if ((self.nbLigne - 1024) > 0) | ((self.nbCol - 1024) > 0) :
                    N=1
                    Nx = int(self.nbLigne/512)
                    Ny = int(self.nbCol/512)
                    if Nx<1:N=Ny
                    if Ny<1:N=Nx
                    else:Nx=Nx

                    for i in range(N-1):

                        #creation  dir ij format 00 00
                        tempname=self.repN(i)
                        dest = os.path.join(self.dirwrktmp,tempname)
                        os.mkdir(dest)
                        #transfert dans dir tempxxxx/xxxyyy
                        os.chdir(dest)
                        # stockage des dir de travail
                        listeDir = listeDir + [dest]

                        #creation du data.m situe en ij dans le masterdata
                        datatemp=self.extractData1D(self.MasterData,
self.nbLigne,self.nbCol,i,Nx,1,Ny)
                        self.sauve.writeMatrix(datatemp,'data.m')

                        #transfert sous la directorie tempxxxxx/
                        os.chdir(self.dirwrktmp)

                else:
                    #signal 1D > 1024

                    #creation  dir ij format 000 000
```

```
                    tempname='000000'
                    dest = os.path.join(self.dirwrktmp,tempname)
                    os.mkdir(dest)
                    os.chdir(dest)

                    # stockage des dir de travail
                    listeDir = listeDir + [dest]

                    #creation du data.m situe en ij dans le masterdata
                    datatemp=self.MasterData
                    self.sauve.writeMatrix(datatemp,'data.m')

                    os.chdir(self.dirwrktmp)

## fin de preparation des donnees

        #liberation de la memoire
        self.MasterData=0


## demarrage des calculs

        #test signal 2D:
        if ((self.nbLigne - 1) > 0) & ((self.nbCol - 1) > 0) :

            #test signal > 512*512
            if ((self.nbLigne - 512) > 0) | ((self.nbCol - 512) > 0) :
                Nx = int(self.nbLigne/256)
                Ny = int(self.nbCol/256)

                if Nx<1:Nx=1
                if Ny<1:Ny=1

                resbask = zeros([int(Nx),2*int(Ny)])
                resbask = resbask.astype(Float64)

                for i in range(Nx-1):
                    for j in range(Ny-1):
                        #transfert dans dir tempxxxx/temp
                        os.chdir(self.dirtampon)

                        #recherche dir ij format 00 00
                        tempname=self.repNxNy(i,j)
```

```
dirtmpname = os.path.join(self.dirwrktmp,tempname)

#copie data.m et wave.i dans tampon pour calcul
liste =os.listdir(dirtmpname)
for k in range(len(liste)):
    srci = os.path.join(dirtmpname,liste[k])
    shutil.copy(srci,self.dirtampon)

#activ wavelet & activ chain & activ basilik40.py
if test == 'posix':
    os.popen('wavelet2d wave.i')
    os.popen('chain2d wave.i')
    os.popen('python basilik40.py')

else:
    os.startfile ('wavelet2d.exe')
    os.startfile ('chain2d.exe')
    os.startfile ('basilik40.py')

srcb = os.path.join(self.dirtampon,'bpenteh.m')
h = bLoad(srcb)
srcb = os.path.join(self.dirtampon,'bpented.m')
d = bLoad(srcb)

bph = h.Info()
bpd = d.Info()

bpenteh = bph[0]
bpented = bpd[0]

resbask[i,2*j]   = bpenteh[0,4]
resbask[i,2*j+1] = bpented[0,4]

#copie des fichiers de sortie dans tempxxxxxx/iijj/
src = os.path.join(self.dirtampon)
for k in range(len(self.fichsort2D)):
    srck = os.path.join(src,self.fichsort2D[k])
    destk = os.path.join(dirtmpname)
    shutil.copy(srck,destk)

#remonte d 1 a tempxxxxx/
#transfert sous la directorie tempxxxxx/
os.chdir(self.dirwrktmp)
```

```
else : #image de 512*512 ou moins

    #une seul image donc resbask
    resbask = zeros([int(1),2*int(1)])
    resbask = resbask.astype(Float64)

    #transfert dans dir tempxxxx/temp
    os.chdir(self.dirtampon)

    #recherche dir ij format 00 00
    tempname='000000'
    dirtmpname = os.path.join(self.dirwrktmp,tempname)

    #copie data.m et wave.i dans tampon pour calcul
    liste =os.listdir(dirtmpname)
    for k in range(len(liste)):
        srci = os.path.join(dirtmpname,liste[k])
        shutil.copy(srci,self.dirtampon)

    #activ wavelet & activ chain & activ basilik40.py
    if test == 'posix':
        os.popen('wavelet2d wave.i')
        os.popen('chain2d wave.i')
        os.popen('python basilik40.py')

    else:
        os.startfile ('wavelet2d.exe')
        os.startfile ('chain2d.exe')
        os.startfile ('basilik40.py')

    srcb = os.path.join(self.dirtampon,'bpenteh.m')
    h = bLoad(srcb)
    srcb = os.path.join(self.dirtampon,'bpented.m')
    d = bLoad(srcb)

    bph = h.Info()
    bpd = d.Info()

    bpenteh = bph[0]
    bpented = bpd[0]

    resbask[0,0] = bpenteh[0,4]
```

```
                resbask[0,1] = bpented[0,4]

                #copie des fichiers de sortie dans tempxxxxxx/iijj/
                src = os.path.join(self.dirtampon)
                for k in range(len(self.fichsort2D)):
                    srck = os.path.join(src,self.fichsort2D[k])
                    destk = os.path.join(dirtmpname)
                    shutil.copy(srck,destk)

                #remonte d 1 a tempxxxxx/
                #transfert sous la directorie tempxxxxx/
                os.chdir(self.dirwrktmp)


        #le signal 1D
        else:
            #test signal >  1024*1 ou 1*1024
            if ((self.nbLigne - 1024) > 0) | ((self.nbCol - 1024) > 0) :
                N=1
                Nx = int(self.nbLigne/512)
                Ny = int(self.nbCol/512)
                if Nx<1:N=Ny
                if Ny<1:N=Nx
                else:Nx=Nx

                resbask = zeros([1,2*int(N)])
                resbask = resbask.astype(Float64)

                for i in range(N-1):
                    #print 'i',i
                    #transfert dans dir tempxxxx/temp
                    os.chdir(self.dirtampon)

                    #recherche dir ij format 00 00
                    tempname=self.repN(i)
                    dirtmpname = os.path.join(self.dirwrktmp,tempname)

                    #copie data.m et wave.i dans tampon pour calcul
                    liste =os.listdir(dirtmpname)
                    for k in range(len(liste)):
                        srci = os.path.join(dirtmpname,liste[k])
                        shutil.copy(srci,self.dirtampon)
```

```
                #activ wavelet & activ chain & activ basilik40.py
                if test == 'posix':
                    os.popen('wavelet2d wave.i')
                    os.popen('chain2d wave.i')
                    os.popen('python basilik40.py')

                else:
                    os.startfile ('wavelet2d.exe')
                    os.startfile ('chain2d.exe')
                    os.startfile ('basilik40.py')

                srcb = os.path.join(self.dirtampon,'bpenteh.m')
                h = bLoad(srcb)
                srcb = os.path.join(self.dirtampon,'bpented.m')
                d = bLoad(srcb)

                bph = h.Info()
                bpd = d.Info()

                bpenteh = bph[0]
                bpented = bpd[0]

                resbask[0,2*i]   = bpenteh[0,4]
                resbask[0,2*i+1] = bpented[0,4]


                src = os.path.join(self.dirtampon)
                #copie des fichiers de sortie dans tempxxxxxx/iijj/
                for k in range(len(self.fichsort1D)):
                    srck = os.path.join(src,self.fichsort1D[k])
                    destk = os.path.join(dirtmpname)
                    shutil.copy(srck,destk)

                #remonte d 1 a tempxxxxx/
                #transfert sous la directorie tempxxxxx/
                os.chdir(self.dirwrktmp)


            else:
                #signal 1D > 1024

                #transfert dans dir tempxxxx/temp
                os.chdir(self.dirtampon)
```

```
#creation du data.m situe en ij dans le masterdata
datatemp=self.MasterData
self.sauve.writeMatrix(datatemp,'data.m')

resbask = zeros([1,2])
resbask = resbask.astype(Float64)

#transfert dans dir tempxxxx/temp
os.chdir(self.dirtampon)

#recherche dir ij format 00 00
tempname='000000'
dirtmpname = os.path.join(self.dirwrktmp,tempname)

#copie data.m et wave.i dans tampon pour calcul
liste =os.listdir(dirtmpname)
for k in range(len(liste)):
    srci = os.path.join(dirtmpname,liste[k])
    shutil.copy(srci,self.dirtampon)

#activ wavelet & activ chain & activ basilik40.py
if test == 'posix':
    os.popen('wavelet2d wave.i')
    os.popen('chain2d wave.i')
    os.popen('python basilik40.py')

else:
    os.startfile ('wavelet2d.exe')
    os.startfile ('chain2d.exe')
    os.startfile ('basilik40.py')

srcb = os.path.join(self.dirtampon,'bpenteh.m')
h = bLoad(srcb)
srcb = os.path.join(self.dirtampon,'bpented.m')
d = bLoad(srcb)

bph = h.Info()
bpd = d.Info()

bpenteh = bph[0]
bpented = bpd[0]
```

```
                    resbask[0,0] = bpenteh[0,4]
                    resbask[0,1] = bpented[0,4]

                    #copie des fichiers de sortie dans tempxxxxxx/iijj/
                    src = os.path.join(self.dirtampon)

                    for k in range(len(self.fichsort1D)):
                        srck = os.path.join(src,self.fichsort1D[k])
                        destk = os.path.join(dirtmpname)
                        shutil.copy(srck,destk)

                    #remonte d 1 a tempxxxxx/
                    #transfert sous la directorie tempxxxxx/
                    os.chdir(self.dirwrktmp)


## fin des calculs


        #enlever la dir tampon avec tous les calculs#
        #os.chdir(self.dirtampon)
        src = self.dirtampon
        liste =os.listdir(src)

        for i in range(len(liste)):
            srci = os.path.join(src,liste[i])
            os.remove(srci)

        self.sauve.writeMatrix(resbask,'resbask.m')
        os.chdir(self.dirpath)
        os.rmdir(self.dirtampon)


    def regulData2D(self,MasterData,nbLigne,nbCol):
        L=512
        if nbCol>=nbLigne:
            L=nbLigne
        else:
            L=nbCol
        Rootemp = zeros([L,L])
        Rootemp = Rootemp.astype(Float64)
```

```
        #on considere que traitement se fait toujours dans une image avec
#un cadre de 256 non traite
        for x in range(L):
            for y in range(L):
                Rootemp[x,y]=MasterData[x,y]
        return Rootemp


    def extractData(self,MasterData,nbLigne,nbCol,i,Nx,j,Ny):
        l = 256
        L = 512
        Rootemp = zeros([L,L])
        Rootemp = Rootemp.astype(Float64)
        #on considere que traitement se fait toujours dans une image avec
#un cadre de 256 non traite
        for x in range(L):
            for y in range(L):
                xi=i*l+x
                yi=j*l+y
                Rootemp[x,y]=MasterData[xi,yi]
        return Rootemp

    def extractData1D(self,MasterData,nbLigne,nbCol,i,Nx,j,Ny):
        l = 512
        L = 1024
        Rootemp = zeros([L,1])
        Rootemp = Rootemp.astype(Float64)
        #on considere que traitement se fait toujours dans une image avec
#un cadre de 256 non traite
        if nbLigne>nbCol:
            for x in range(L):
                xi=i*l+x
                Rootemp[x,0]=MasterData[xi,0]
        else :
            yi=j*l+y
            Rootemp[0,y]=MasterData[0,yi]
        return Rootemp


    def repNxNy(self,i,j):
        a='000000'
        i=int(i)
        j=int(j)
```

```
    if ((i-10)<0):
        if ((j-10)<0):
            a='00'+'i'+'00'+'j'
        if ((j-100)<0) & ((j-10)>=0):
            a='00'+'i'+'0'+'j'
        if ((j-100)>=0):
            a='00'+'i'+'j'

    if ((i-100)<0) & ((i-10)>=0):
        if ((j-10)<0):
            a='0'+'i'+'00'+'j'
        if ((j-100)<0) & ((j-10)>=0):
            a='0'+'i'+'0'+'j'
        if ((j-100)>=0):
            a='0'+'i'+'j'

    if ((i-100)>=0):
        if ((j-10)<0):
            a='i'+'00'+'j'
        if ((j-100)<0) & ((j-10)>=0):
            a='i'+'0'+'j'
        if ((j-100)>=0):
            a='i'+'j'
    return a


def repN(self,i):
    a='000000'
    i=int(i)

    if ((i-10)<0):
            a='00'+'i'
    if ((i-100)<0) & ((i-10)>=0):
            a='0'+'i'
    if ((i-100)>=0):
            a='i'
    return a

def remplissage(self,name,nbligne,nbcol):
    Rootemp = zeros([nbligne,nbcol])
    Rootemp = Rootemp.astype(Float64)
    lines = 0
```

```
        words = 0
        for line in TextFile(name):
            lines = lines +1
            words = len(split(line))
            temp = split(line)
            for x in range(words):
                Rootemp[(lines-1),x] = float(temp[x])
        return Rootemp


    def nblin(self,name):
        lines = 0
        #calcul de la hauteur du tableau de donnees
        for line in TextFile(name):
            lines = lines + 1
        return lines


    def nbcol(self,name):
        lines = 0
        words = 0
        max = 0
        #calcul de la largeur max du tableau de donnees
        for line in TextFile(name):
            words = len(split(line))
            if (max - words)<0:
                max = words
        return max


    def finProg(self):
        #sortie de la directorie tampon(xxx+1) (cad remonte de un cd..
#ou cd ..)
        os.chdir(self.dirpath)


class Boucle:
    def __init__(self):
        win = Frame()
        win.pack()
        Label(win, text='Recommencer WavePlaza ?').pack(side=TOP)
        Button(win, text='Oui', command=self.encore).pack(side=LEFT)
        Button(win, text='Quitter', command=win.quit).pack(side=RIGHT)
```

```
            win.mainloop()


        def encore(self):
            self.dirpath = os.getcwd() #donne le path courant
            self.dir = os.path.join(self.dirpath,'..','..')
            os.chdir(self.dir)
            print 'nouvel essaie'
            start()
            sys.exit()

class start:
    def __init__(self):
        #test de la classe PrepaWaveplz
        PrepaWaveplz()
        Boucle()

        print 'check over'


start()
```

Annexe C

CODE DE GUIWAVEPLZ.PY

```
import sys, os, string, fpformat, os.path, dircache, shutil, time
import math
from string import split
sys.path.append('C:\Program Files\python\Lib\site-packages\Numeric')
import Numeric
from Numeric import *
from Tkinter import *
from FileDialog  import *

#
# guiWaveplz.py
# Written by: Gael Sitzia <sitzia@crm.umontreal.ca>
# Last revision: filtres 1D & 2D
#



class TextFile:

    def __init__(self, filename, mode = 'r'):
        if string.find(filename, ':/') > 1: # URL
            if mode != 'r':
                raise IOError, "can't write to a URL"
            import urllib
            self.file = urllib.urlopen(filename)
        else:
            filename = os.path.expanduser(filename)
            if mode == 'r':
                if not os.path.exists(filename):
                    raise IOError, (2, 'No such file or directory: '
                                    + filename)
                if filename[-2:] == '.Z':
                    self.file = os.popen("uncompress -c " + filename, mode)
```

```
    elif filename[-3:] == '.gz':
        if gzip is None:
            self.file = os.popen("gunzip -c " + filename, mode)
        else:
            self.file = gzip.GzipFile(filename, 'rb')
    elif filename[-4:] == '.bz2':
        self.file = os.popen("bzip2 -dc " + filename, mode)
    else:
        try:
            self.file = open(filename, mode)
        except IOError, details:
            if type(details) == type(()):
                    details = details + (filename,)
            raise IOError, details
elif mode == 'w':
    if filename[-2:] == '.Z':
        self.file = os.popen("compress > " + filename, mode)
    elif filename[-3:] == '.gz':
        if gzip is None:
            self.file = os.popen("gzip > " + filename, mode)
        else:
            self.file = gzip.GzipFile(filename, 'wb')
    elif filename[-4:] == '.bz2':
        self.file = os.popen("bzip2 > " + filename, mode)
    else:
        try:
            self.file = open(filename, mode)
        except IOError, details:
            if type(details) == type(()):
                    details = details + (filename,)
            raise IOError, details
elif mode == 'a':
    if filename[-2:] == '.Z':
        raise IOError, (0, "Can't append to .Z files")
    elif filename[-3:] == '.gz':
        if gzip is None:
            self.file = os.popen("gzip >> " + filename, "w")
        else:
            self.file = gzip.GzipFile(filename, 'ab')
    else:
        self.file = open(filename, mode)
else:
    raise IOError, (0, 'Illegal mode: ' + repr(mode))
```

```
    def __del__(self):
self.close()

    def __getitem__(self, item):
line = self.file.readline()
if not line:
    raise IndexError
return line

    def readline(self):
return self.file.readline()

    def readlines(self):
return self.file.readlines()

    def write(self, data):
self.file.write(data)

    def writelines(self, list):
for line in list:
    self.file.write(line)

    def close(self):
self.file.close()

    def flush(self):
        self.file.flush()


class Dialog(Toplevel):

    def __init__(self, parent, title = None):


        Toplevel.__init__(self, parent)
        self.transient(parent)

        if title:
            self.title(title)

        self.parent = parent
```

```
        self.result = None

        body = Frame(self)
        self.initial_focus = self.body(body)
        body.pack(padx=7, pady=7)

        self.buttonbox()

        self.grab_set()

        if not self.initial_focus:
            self.initial_focus = self

        self.protocol("WM_DELETE_WINDOW", self.cancel)

        self.geometry("+%d+%d" % (parent.winfo_rootx()+50,
                                  parent.winfo_rooty()+50))

        self.initial_focus.focus_set()

        self.wait_window(self)

    #
    # construction hooks

    def body(self, master):
        #-----Initialisation du fichier data.m-----
        dirpath = os.getcwd() #donne le path courant
        dirdata = os.path.join(dirpath,'data.m')

        self.nbLigne = int(self.nblin(dirdata))
        self.nbCol = int(self.nbcol(dirdata))
        self.dimension = 1

        if ((self.nbLigne - 1) > 0) & ((self.nbCol - 1) > 0) :
self.dimension = 2

        self.nb_vox = 5
        self.nb_octave = 5
        self.border_effect = 1
        self.convolution_type = 0
        self.derivation_order = 1
        self.norm_exponent = 0.5
```

```
self.form = 0
self.omega_max = 0.67787
self.t_max = 0.667
self.first_scale = 0.666
self.last_scale = 0.8
self.nb_sample_sig = 512
self.nb_col = 512
self.nb_line = 1
self.It = -1

Label(master, text="Dimension:" ).grid(row=0, sticky=W)
Label(master, text="Derivation order:" ).grid(row=1, sticky=W)
Label(master, text="Convolution type:").grid(row=2, sticky=W)
Label(master, text="Border effet:").grid(row=3, sticky=W)
Label(master, text="Number of octave:").grid(row=4,sticky=W)
Label(master, text="Number of voice:").grid(row=5, sticky=W)
Label(master, text="Normalisation exponent:").grid(row=6, sticky=W)
Label(master, text="It:").grid(row=7, sticky=W)

self.e0 = Entry(master)
self.e1 = Entry(master)
self.e2 = Entry(master)
self.e3 = Entry(master)
self.e4 = Entry(master)
self.e5 = Entry(master)
self.e6 = Entry(master)
self.e7 = Entry(master)

self.e0.grid(row=0, column=1)
self.e1.grid(row=1, column=1)
self.e2.grid(row=2, column=1)
self.e3.grid(row=3, column=1)
self.e4.grid(row=4, column=1)
self.e5.grid(row=5, column=1)
self.e6.grid(row=6, column=1)
self.e7.grid(row=7, column=1)

self.e0.insert(INSERT,self.dimension)
self.e1.insert(INSERT,self.derivation_order)
self.e2.insert(INSERT,self.convolution_type)
self.e3.insert(INSERT,self.border_effect)
self.e4.insert(INSERT,self.nb_octave)
self.e5.insert(INSERT,self.nb_vox)
```

```
        self.e6.insert(INSERT,self.norm_exponent)
        self.e7.insert(INSERT,self.It)

        self.v = IntVar()

        self.cb = Checkbutton(master, text="Chainage", variable=self.v)
        self.cb.grid(row=8, columnspan=2, sticky=W)

        self.v2 = IntVar()

        self.cb2 = Checkbutton(master, text="Basilik", variable=self.v2)
        self.cb2.grid(row=8, columnspan=2, sticky=E)


    def apply(self):
        try:

            self.dimension = int(self.e0.get())
            self.nb_vox = int(self.e1.get())
            self.nb_octave = int(self.e2.get())
            self.border_effect = int(self.e3.get())
            self.convolution_type = int(self.e4.get())
            self.derivation_order = int(self.e5.get())
            self.norm_exponent = float(self.e6.get())
            self.form = int(0)
            self.omega_max = int(0.67787)
            self.t_max = float(0.667)
            self.first_scale = int(0.666)
            self.last_scale = int(0.8)
            self.nb_sample_sig = int(512)
            self.nb_col = int(512)
            self.nb_line = int(1)
            self.It = int(self.e7.get())

            print 'cb', self.cb,' cb2', self.cb2

            self.cb = self.v.get()
            self.cb2 =self.v2.get()

            print 'cb', self.cb,' cb2', self.cb2

            self.result = self.dimension ,self.nb_vox ,self.nb_octave ,
self.border_effect ,self.convolution_type ,self.derivation_order ,
```

```
self.norm_exponent ,self.form ,self.omega_max ,self.t_max ,self.first_scale
,self.last_scale ,self.nb_sample_sig ,self.nb_col ,self.nb_line ,self.cb
,self.cb2

        except ValueError:
            self.dimension = int(1)
            self.nb_vox = int(5)
            self.nb_octave = int(5)
            self.border_effect =  int(1)
            self.convolution_type =  int(0)
            self.derivation_order =  int(1)
            self.norm_exponent =  float(0.5)
            self.form =  int(0)
            self.omega_max =  int(0.67787)
            self.t_max =  float(0.667)
            self.first_scale =  int(0.666)
            self.last_scale =  int(0.8)
            self.nb_sample_sig =  int(512)
            self.nb_col =  int(512)
            self.nb_line =  int(-999)
            self.It = -1

            print '2cb', self.cb,' cb2', self.cb2

            self.cb = 1
            self.cb2 = 1

            print '2cb', self.cb,' cb2', self.cb2

            self.result = self.dimension ,self.nb_vox ,self.nb_octave
,self.border_effect ,self.convolution_type ,self.derivation_order
,self.norm_exponent ,self.form ,self.omega_max ,self.t_max
,self.first_scale ,self.last_scale ,self.nb_sample_sig ,self.nb_col
,self.nb_line, self.It ,self.cb ,self.cb2

            tkMessageBox.showwarning("Aucune donnee","les valeurs
par defaut seront asignees")

    def buttonbox(self):
        # add standard button box. override if you don't want the
        # standard buttons

        box = Frame(self)
```

```
        w = Button(box, text="OK", width=10, command=self.ok, default=ACTIVE
        w.pack(side=LEFT, padx=5, pady=5)
        w = Button(box, text="Cancel", width=10, command=self.cancel)
        w.pack(side=LEFT, padx=5, pady=5)

        self.bind("&lt;Return>", self.ok)
        self.bind("&lt;Escape>", self.cancel)

        box.pack()

    #
    # standard button semantics

    def ok(self, event=None):

        self.withdraw()
        self.update_idletasks()
        self.apply()

        test = os.name
        #Extrait les entrees de l'utilisateur
        dimension = string.atoi(self.e0.get())
        derivation_order = string.atoi(self.e1.get())
        convolution_type = string.atoi(self.e2.get())
        border_effect = string.atoi (self.e3.get())
        nbOctave = string.atoi (self.e4.get())
        nbVox = string.atoi (self.e5.get())
        exponent = string.atof (self.e6.get())
        nbLigne = int(self.nbLigne)
        nbCol = int(self.nbCol)

        if (nbCol-nbLigne)< 0 :
            tmp = nbCol
            nbCol = nbLigne
            nbLigne = tmp


        size = int(nbCol*nbLigne)

        #faire une creation d un fichier wave.i (anciennement
#Wavelet.info) avec 20 lignes d initialise
        dirpath = os.getcwd() #donne le path courant
```

```
dirwave = os.path.join(dirpath,'wave.i')


f=open(dirwave, 'w')

#print 'check2 size', size, ' col ', nbCol, ' lin ', nbLigne

if dimension == 1:
   f.write('1 Wavelet1D')
else:
   f.write('2 Wavelet2D')
#f.write('Wavelet1D')                              #ligne 1



f.write('\nBorderEffect : ')                       #ligne 2
if border_effect == 0:
    f.write('CV1D_PERIODIC')
if border_effect == 1:
     f.write('CV1D_MIRROR')
if border_effect == 2:
    f.write('CV1D_PADDING')
if border_effect == 3:
     f.write('CV1D_0_PADDING')

f.write('\nConvolution Type : ')                   #ligne 3
if convolution_type == 0:
    f.write('DIRECT_CONV')
if convolution_type == 1:
    f.write('FFTW_CONV')

#print 'check3'

f.write('\nDerivation order : '+ self.e3.get() )  #ligne 4
f.write('\nNorm exponent : '+ self.e6.get() )     #ligne 5
f.write('\nForm : REAL ')#+listline[5])            #ligne 6
f.write('\nOmega_max : 4.4000 ')#+listline[6])     #ligne 7
f.write('\nt_max : 55.5000 ')#+listline[7])        #ligne 8
f.write('\nNumber of octave : '+self.e4.get())     #ligne 9
f.write('\nNumber of vox : '+self.e5.get())       #ligne 10
f.write('\nFirst scale : 2.2000 ')#+listline[10])  #ligne 11
f.write('\nLast scale : 43.4000 ')#+listline[11])  #ligne 12
```

```
        if dimension == 1:
            if (size <= 1024) :
                f.write('\nSignal size : ' + `size` )
      #ligne 13
                f.write('\nSignal.nbcols : ' + `nbCol` )
      #ligne 14
                f.write('\nSignal.nblines : ' + `nbLigne` )
  #ligne 15
            if (size > 1024) :
                f.write('\nSignal size : 1024')# + `size` )
      #ligne 13
                f.write('\nSignal.nbcols : 1024')# + `nbCol` )
      #ligne 14
                f.write('\nSignal.nblines : 1')# + `nbLigne` )
)   #ligne 15

        if dimension == 2:
            if (size <=  262144) :
                f.write('\nSignal size : ' + `size` )
#ligne 13
                f.write('\nSignal.nbcols : ' + `nbCol` )
#ligne 14
                f.write('\nSignal.nblines : ' + `nbLigne` )
#ligne 15
                f.write('\nIt : '+ self.e7.get())
#ligne 16
            if (size >  262144) :
                f.write('\nSignal size : 262144')# + `size` )
      #ligne 13
                f.write('\nSignal.nbcols : 512')# + `nbCol` )
      #ligne 14
                f.write('\nSignal.nblines : 512')# + `nbLigne` )
  #ligne 15
                f.write('\nIt : '+ self.e7.get())

        #print 'check 4'
        f.close()

        print 'fin creation wave.i'

        if not self.validate():
            self.initial_focus.focus_set() # put focus back
```

```
            return

        self.cancel()


    def cancel(self, event=None):

        # put focus back to the parent window
        self.parent.focus_set()
        self.destroy()

    #
    # command hooks

    def validate(self):

        return 1 # override


    def nblin(self,name):
        lines = 0
        #calcul de la hauteur du tableau de donnees
        for line in TextFile(name):
            lines = lines + 1
        return lines


    def nbcol(self,name):
        lines = 0
        words = 0
        max = 0
        #calcul de la largeur max du tableau de donnees
        for line in TextFile(name):
            words = len(split(line))
            if (max - words)<0:
                max = words
        return max


class Maxima:
    def __init__(self):
```

```
        #self.sauve = SauveInfo()
        #a = PrepaWaveplz()

        #demarrage interface
        self.root = Tk()
        Button(self.root, text="bienvenue dans WavePlaza! calcul en cours !"
.pack()
        self.root.update()

        d = Dialog(self.root)

        #a.finProg()

print 'check gui in'
Maxima()
print 'check gui out'
```

Annexe D

CODE DE BASILIK40.PY

```
#! /usr/local/lib/python2.2 python

import sys, os, Numeric, string, fpformat
import math
from string import split
from Numeric import *
from Tkinter import *

# basilik40.py
# Written by: Gael Sitzia <sitzia@crm.umontreal.ca>
# Last revision: compatible avec waveplz 2.0, plus de lib Dislin
#



class TextFile:

    """Text files with line iteration and transparent compression

    TextFile instances can be used like normal file objects
    (i.e. by calling readline(), readlines(), and write()), but can
    also be used as sequences of lines in for-loops.

    TextFile objects also handle compression transparently. i.e. it is
    possible to read lines from a compressed text file as if it were not
    compressed.  Compression is deduced from the file name suffixes '.Z'
    (compress/uncompress), '.gz' (gzip/gunzip), and '.bz2' (bzip2).

    Finally, TextFile objects accept file names that start with '~' or
    '~user' to indicate a home directory, as well as URLs (for reading only)

    Constructor: TextFile(|filename|, |mode|='"r"'), where |filename| is
```

```
the name of the file (or a URL) and |mode| is one of '"r"' (read),
'"w"' (write) or '"a"' (append, not supported for .Z files).
"""

def __init__(self, filename, mode = 'r'):
    if string.find(filename, ':/') > 1: # URL
        if mode != 'r':
            raise IOError, "can't write to a URL"
        import urllib
        self.file = urllib.urlopen(filename)
    else:
        filename = os.path.expanduser(filename)
        if mode == 'r':
            if not os.path.exists(filename):
                raise IOError, (2, 'No such file or directory: '
                                + filename)
            if filename[-2:] == '.Z':
                self.file = os.popen("uncompress -c " + filename, mode)
            elif filename[-3:] == '.gz':
                if gzip is None:
                    self.file = os.popen("gunzip -c " + filename, mode)
                else:
                    self.file = gzip.GzipFile(filename, 'rb')
            elif filename[-4:] == '.bz2':
                self.file = os.popen("bzip2 -dc " + filename, mode)
            else:
                try:
                    self.file = open(filename, mode)
                except IOError, details:
                    if type(details) == type(()):
                            details = details + (filename,)
                    raise IOError, details
        elif mode == 'w':
            if filename[-2:] == '.Z':
                self.file = os.popen("compress > " + filename, mode)
            elif filename[-3:] == '.gz':
                if gzip is None:
                    self.file = os.popen("gzip > " + filename, mode)
                else:
                    self.file = gzip.GzipFile(filename, 'wb')
            elif filename[-4:] == '.bz2':
                self.file = os.popen("bzip2 > " + filename, mode)
            else:
```

```
                    try:
                        self.file = open(filename, mode)
                    except IOError, details:
                        if type(details) == type(()):
                                details = details + (filename,)
                        raise IOError, details
            elif mode == 'a':
                if filename[-2:] == '.Z':
                    raise IOError, (0, "Can't append to .Z files")
                elif filename[-3:] == '.gz':
                    if gzip is None:
                        self.file = os.popen("gzip >> " + filename, "w")
                    else:
                        self.file = gzip.GzipFile(filename, 'ab')
                else:
                    self.file = open(filename, mode)
            else:
                raise IOError, (0, 'Illegal mode: ' + repr(mode))

    def __del__(self):
self.close()

    def __getitem__(self, item):
line = self.file.readline()
if not line:
    raise IndexError
return line

    def readline(self):
return self.file.readline()

    def readlines(self):
return self.file.readlines()

    def write(self, data):
self.file.write(data)

    def writelines(self, list):
for line in list:
    self.file.write(line)

    def close(self):
self.file.close()
```

```
    def flush(self):
        self.file.flush()

class Dialog(Toplevel):

    def __init__(self, parent, title = None):

        Toplevel.__init__(self, parent)
        self.transient(parent)

        if title:
            self.title(title)

        self.parent = parent

        self.result = None

        body = Frame(self)
        self.initial_focus = self.body(body)
        body.pack(padx=7, pady=7)

        self.buttonbox()

        self.grab_set()

        if not self.initial_focus:
            self.initial_focus = self

        self.protocol("WM_DELETE_WINDOW", self.cancel)

        self.geometry("+%d+%d" % (parent.winfo_rootx()+50,
                                  parent.winfo_rooty()+50))

        self.initial_focus.focus_set()

        self.wait_window(self)

    #
    # construction hooks

    def body(self, master):
```

```
        bord_dflt = 0.25
        lgMin_dflt = 1
        lgMax_dflt = 45
        q_dflt = 9


        Label(master, text="Bord:" ).grid(row=0, sticky=W)
        Label(master, text="Lg Min:").grid(row=1, sticky=W)
        Label(master, text="Lg Max:").grid(row=2, sticky=W)
        Label(master, text="# q:").grid(row=3,sticky=W)

        self.e1 = Entry(master)
        self.e2 = Entry(master)
        self.e3 = Entry(master)
        self.e4 = Entry(master)

        self.e1.grid(row=0, column=1)
        self.e2.grid(row=1, column=1)
        self.e3.grid(row=2, column=1)
        self.e4.grid(row=3, column=1)

        self.e1.insert(INSERT,bord_dflt)
        self.e2.insert(INSERT,lgMin_dflt)
        self.e3.insert(INSERT,lgMax_dflt)
        self.e4.insert(INSERT,q_dflt)

        self.v = IntVar()

        self.cb = Checkbutton(master, text="Verification",
variable=self.v)
        self.cb.grid(row=4, columnspan=2, sticky=W)

    def apply(self):
        try:
            first = float(self.e1.get())
            second = int(self.e2.get())
            third = int(self.e3.get())
            fourth = int(self.e4.get())
            cb = self.v.get()
            self.result = first , second, third, fourth, cb
            #self.result = first , fourth, cb

        except ValueError:
```

```
            first = 0.25
            second = 1
            third = 45
            fourth = 9
            cb = 0
            self.result = first , second, third, fourth, cb
            #self.result = first , fourth, cb
            tkMessageBox.showwarning("Aucune donnee","les valeurs
par defaut seront asignees")

    def buttonbox(self):
        # add standard button box. override if you don't want the
        # standard buttons

        box = Frame(self)

        w = Button(box, text="OK", width=10, command=self.ok,
default=ACTIVE)
        w.pack(side=LEFT, padx=5, pady=5)
        w = Button(box, text="Cancel", width=10, command=self.cancel)
        w.pack(side=LEFT, padx=5, pady=5)

        self.bind("&lt;Return>", self.ok)
        self.bind("&lt;Escape>", self.cancel)

        box.pack()

    #
    # standard button semantics

    def ok(self, event=None):

        if not self.validate():
            self.initial_focus.focus_set() # put focus back
            return

        self.withdraw()
        self.update_idletasks()

        self.apply()

        self.cancel()
```

```
    def cancel(self, event=None):

        # put focus back to the parent window
        self.parent.focus_set()
        self.destroy()

    #
    # command hooks

    def validate(self):

        return 1 # override


class SauveInfo:


    def readMatrix(filename):
        rows = []
        for line in TextFile(self,filename):
            columns = []
            for number in string.split(line):
                columns.append(string.atof(number))
            rows.append(columns)
        return Numeric.array(rows)

    def writeMatrix(self,a, filename):
        file = TextFile(filename, 'w')
        for line in a:
            for number in line:
                file.write(`number` + ' ')
            file.write('\n')
        file.close()


class Info:
    def __init__(self):

        #extraction des infos de  Wavelet.info sur l'image utilise
        self.wtinfo=self.remplissage()
        #print 'wtinfo', self.wtinfo
        self.derivOrder=self.derivOrder()
        self.normExp=self.normExp()
```

```
        self.omegaMax=self.omegaMax()
        self.tMax=self.tMax()
        self.nbOctave=self.nbOctave()
        self.nbVox=self.nbVox()
        self.firstScale=self.firstScale()
        self.lastScale=self.lastScale()
        self.Size=self.Size()
        self.nbCol=self.nbCol()
        self.nbLign=self.nbLign()
        self.dimension = self.dim()
        #test impression de toutes les donnees extraites
        #a = self.printa()

    def printa(self):
        tab = self.wtinfo
        a=0
        for line in tab:
            print line

        print self.derivOrder
        print self.normExp
        print self.omegaMax
        print self.tMax
        print self.nbOctave
        print self.nbVox
        print self.firstScale
        print self.lastScale
        print self.Size
        print self.nbCol
        print self.nbLign

        return a

    def dim(self):
        ans=1
        DD = '2'
        tab = self.wtinfo
        a=tab[0]
        #print 'dim tab[0]' , a, ' a[0] ', a[0]
        if a[0] == DD:
            ans = 2
        else:
            ans = 1
```

```
        return ans


    def derivOrder(self):
        tab = self.wtinfo
        a=tab[3]
        return a[3]

    def normExp(self):
        tab = self.wtinfo
        a=tab[4]
        return a[3]

    def omegaMax(self):
        tab = self.wtinfo
        a=tab[6]
        return a[2]

    def tMax(self):
        tab = self.wtinfo
        a=tab[7]
        return a[2]

    def nbOctave(self):
        tab = self.wtinfo
        a=tab[8]
        return a[4]

    def nbVox(self):
        tab = self.wtinfo
        a=tab[9]
        return a[4]

    def firstScale(self):
        tab = self.wtinfo
        a=tab[10]
        return a[3]

    def lastScale(self):
        tab = self.wtinfo
        a=tab[11]
        return a[3]
```

```
    def Size(self):
        tab = self.wtinfo
        a=tab[12]
        return a[3]

    def nbCol(self):
        tab = self.wtinfo
        a=tab[13]
        return a[2]

    def nbLign(self):
        tab = self.wtinfo
        a=tab[14]
        return a[2]

    def remplissage(self):
        wtinfotemp = [[0],[0],[0],[0],[0],[0],[0],[0],[0],[0],
[0],[0],[0],[0],[0],[0],[0],[0],[0],[0],[0],[0],[0],[0],[0]]
        lines = 0
        for line in TextFile('wave.i'):
            lines = lines +1
            temp = split(line)
            wtinfotemp[(lines-1)] = temp
        a = wtinfotemp[4]
        return wtinfotemp

class Maxima:
    def __init__(self):
        #compilation des donnees contenuent dans wavelet.info
        self.image = Info()
        self.sauve = SauveInfo()

        #demarrage interface
        self.root = Tk()
        Button(self.root, text="bienvenue dans basilik! calcul
en cours !").pack()
        self.root.update()

        #extraction de coef d ondelette
        self.filename = 'matlabroot.m'
        self.filescale = 'scale.m'
        #extraction des valeurs scale (a) du fichier scale.m
        self.xFile = self.words(self.filescale)
```

```
        self.yFile = self.lines(self.filescale)
        self.scale = self.remplissage(self.filescale)
        #extraction des coef d ondelette du fichier matlabroot.m
        self.xFile = self.words(self.filename)
        self.xRoot = self.bNorm(self.xFile)
        self.yFile = self.lines(self.filename)
        self.bord = 0.25
        #self.bord = self.boxFloatBord()
        print "nombre de ligne de maxima", self.yFile
        print "nombre de niveau", self.xRoot
        self.lgnMin = 1
        self.lgnMax = self.xRoot
        self.q = self.setQ(9) #q varie de -4a +4 par pas de 1 par defaut
        self.cb = 0

        self.dimension = 1
        self.dimension = self.image.dimension

        #separation du fichier matlabroot en deux: racines
#et localisations
        self.yRoot = self.yFile
        self.rootfile = self.remplissage(self.filename)
        self.roots = self.remplissageRoot()
        self.locaRoots = self.remplissageLoca()
        self.locaFin = self.loca()


        #activation de la boite de dialogue
        self.joshua = self.joshua()
        self.h=self.holder()
        self.D=self.d()
        self.h_pente=self.pente_holder()
        self.d_pente=self.pente_d()


        #test de verification sur les donnees du fichier
#des lignes de maxima
        #a=self.printa()

        #Boucle pour refaire les calculs sur le meme signal
        ##boucle enleve ench/nement de traitement automatique
        #Boucle()
```

```
def joshua(self):
    #d = Dialog(self.root)
    self.lgnMin = 1                    #d.result[1]
    self.lgnMax = self.dome(50)        #self.dome(d.result[2])
    self.bord = 0.25                   #d.result[0]
    self.q = self.setQ(9)              #self.setQ(d.result[3])
    self.cb = 0                        #d.result[4]
    dimension = self.dimension
    print 'dimension ', dimension
    format1D = 0
    format2D = 0


    self.lgnMaxFin = self.fin()
    self.lgnMaxLong = self.longueur()
    self.lgnMaxDebut = self.debut()


    if self.dimension == 2:
        print 'dim 2D'
        format2D = self.reformat2D()
        self.lignes = self.calcNbLign()
        self.M = self.regulRoot2D()
    else :
        print 'dim 1D'
        format1D = self.reformat1D()
        self.lignes = self.calcNbLign()
        self.M = self.regulRoot1D()

    print 'M'
    #print self.M
    print "nombre de lignes apres trie", len(self.lignes)
    self.Z = self.partition()


def dome(self, lgmax):
    a = self.xRoot - lgmax
    if a<0 :
        return self.xRoot
    return lgmax
```

```
    def printa(self):
        a=0
        print "root file "
        print self.rootfile
        print "localisation des fins de lignes de maxima"
        print self.locaFin
        print " debuts des lignes de maxima"
        print self.lgnMaxDebut
        print "fins des lignes de maxima"
        print self.lgnMaxFin
        print "longueurs des lignes"
        print self.lgnMaxLong
        print "roots "
        print self.roots
        print "localisations "
        print self.locaRoots
        print "maxima regularise"
        print self.maxima
        print "pente holder"
        print self.h_pente
        print "pente dimension de hausendorf"
        print self.d_pente
        return a


    def reformat1D(self):
        c=1
        fin = self.lgnMaxFin
        debut = self.lgnMaxDebut
        nblevel = self.xRoot
        tabNbLign=self.lgnMaxLong
        levels = self.xRoot
        nbLines = self.yRoot
        Rootemp = zeros([nbLines,self.xRoot])
        Rootemp = Rootemp.astype(Float64)
        regulRoots = self.roots

        #print 'avant', regulRoots
        #passage en valeur absolue des maxima et inversion
#de leur ordre pour la 1D
        for i in range(nbLines):
            for j in range(levels):
```

```
#        print 'j',j,'j-levels',j-levels
                Rootemp[i,levels-j-1]=abs(regulRoots[i,j])
#        Rootemp[i,j]=abs(regulRoots[i,j])
        self.roots = Rootemp

        #print 'apres', Rootemp
        return c

    def reformat2D(self):
        c=0
        fin = self.lgnMaxFin
        debut = self.lgnMaxDebut
        nblevel = self.xRoot
        tabNbLign=self.lgnMaxLong

        for i in range(self.yRoot):

            j=int(tabNbLign[i])
            deb = nblevel - fin[i]
            debut[i]=deb
            fin[i]=deb+tabNbLign[i]
            #print 'nblevel ' ,nblevel ,' fin[i]', fin[i],
' debut[i]',debut[i],' deb', deb ,' j', j, ' lgi', tabNbLign[i]

        levels = self.xRoot
        nbLines = self.yRoot
        Rootemp = zeros([nbLines,self.xRoot])
        Rootemp = Rootemp.astype(Float64)
        regulRoots = self.roots

        #passage en valeur absolue des maxima et inversion de
#leur ordre pour la 2D
        for i in range(nbLines):
            for j in range(levels):
                Rootemp[i,j-levels]=abs(regulRoots[i,j])

        self.roots = Rootemp
        self.lgnMaxFin = fin
        self.lgnMaxDebut = debut

        return c
```

```
    def calcNbLign(self):
        bord  = self.bord

        lgMin = self.lgnMin
        lgMax = self.lgnMax

        fin = self.lgnMaxFin
        debut = self.lgnMaxDebut

        nblevel = self.xRoot

        x = int(self.image.nbCol)
        xmin = 0
        xmax = x
        y = int(self.image.nbLign)
        ymin = 0
        ymax = y
        if y != 1:
            ymin = y*bord
            ymax = y*(1-bord)
        if x != 1:
            xmin = x*bord
            xmax = x*(1-bord)
        loca = self.locaFin

        tabNbLign=self.lgnMaxLong
        b = float(self.image.Size)
        b=int(b/2)
        a=[b]
        #douane = 0
        for i in range(self.yRoot):
            #douane +=1

            #print douane
            min=int(tabNbLign[i]-lgMin)
            max=int(tabNbLign[i]-lgMax)
            deb = nblevel - fin[i]
            #print 'nblevel ' ,nblevel ,' fin[i]', fin[i],
#' debut[i]',debut[i],' deb', deb ,' j', j, ' lgi', tabNbLign[i]
#, ' lgmin' ,lgMin
            xi=self.xi(loca[i],x)
```

```
            if ymax != 1:
                yi=self.yi(loca[i],x)

            else:
                yi=1

            if  (xi >= xmin) & (xi <= xmax) & (yi >= ymin) & (yi <= ymax)
#& (debut[i] <= 3) & (max <= 0) & (min >= 0):
                a=a+[i]
                #print "oui"
            else:
                a=a
                #print "non"
        if len(a)>1:
            c = zeros([(len(a)-1)])
            for k in range(len(a)-1):
                c[k]=a[k+1]
        else :
            c = a
        return c

    def yi(self,i,largeur):
        yi = int(i/largeur)
        return yi

    def xi(self,i,largeur):
        xi = int(i%largeur)
        return xi


    def setQ(self,a):
        b=range(a)
        for i in b:
            b[i]=b[i]-int(a/2)
        return b

    def bNorm(self,a):
        b=(a-4)/2
        return b

    def loca (self):
        Root = self.rootfile
        r_loc = Root[:,0]
```

```
        return r_loc

    def debut(self):
        Rootfin = self.lgnMaxFin
        lgnMaxLong = self.lgnMaxLong
        lines = self.yRoot
        r_debut = zeros([lines])
        for i in range(lines):
            r_debut[i] = Rootfin[i] - lgnMaxLong[i] + 1
        return r_debut

    def fin (self):
        Root = self.rootfile
        r_fin = Root[:,1]
        return r_fin

    def longueur (self):
        Root = self.rootfile
        r_longueur = Root[:,2]
        return r_longueur

    def lines(self,name):
        lines = 0
        #calcul de la hauteur du tableau de donnees
        for line in TextFile(name):
            lines = lines + 1
        return lines

    def words(self,name):
        lines = 0
        words = 0
        max = 0
        #calcul de la largeur max du tableau de donnees
        for line in TextFile(name):
            words = len(split(line))
            if (max - words)<0:
                max = words
        return max

    def remplissage(self,name):
        Rootemp = zeros([self.yFile,self.xFile])
        Rootemp = Rootemp.astype(Float64)
        lines = 0
```

```
            words = 0
            for line in TextFile(name):
                lines = lines +1
                words = len(split(line))
                temp = split(line)
                for x in range(words):
                    Rootemp[(lines-1),x] = float(temp[x])
            return Rootemp

        def remplissageRoot(self):
            Rootemp = zeros([self.yRoot,self.xRoot])
            Rootemp = Rootemp.astype(Float64)
            for i in range(self.xRoot):
                j= i*2 + 4
                Rootemp[:,i] = self.rootfile[:,j]
            return Rootemp

        def remplissageLoca(self):
            Rootemp = zeros([self.yRoot,self.xRoot])
            Rootemp = Rootemp.astype(Float64)
            for i in range(self.xRoot):
                j= i*2 + 5
                Rootemp[:,i] = self.rootfile[:,j]
            return Rootemp

        def regulRoot2D(self):
            levels = self.xRoot
            nbLines = len(self.lignes)
            lines = self.lignes
            Rootemp = zeros([nbLines,self.xRoot])
            Rootemp = Rootemp.astype(Float64)
            regulRoots = self.roots

            print 'regul 2D'

            #passage en valeur absolue des maxima
            for i in range(nbLines):
                for j in range(levels):
                    Rootemp[i,j]=abs(regulRoots[i,j])

            ##regularisation enlever a fin de test des fois qu en 2D ca ne
#serait pas des lignes de maxima
            #regularisation des lignes de maxima pour avoir la plus
```

```
#forte valeur en haut
        for i in range(nbLines):
            for j in range((levels-1)):
                temp = Rootemp[i,(j+1)]-Rootemp[i,j]
                if (temp < 0) & (Rootemp[i,(j+1)] != 0) | (temp>10000):
                    Rootemp[i,(j+1)]=Rootemp[i,j]

        Rootemp1 = zeros([levels,nbLines])
        Rootemp1 = Rootemp1.astype(Float64)
        for i in range(nbLines):
            for j in range(levels):
                Rootemp1[j,i]=Rootemp[i,j]
        self.sauve.writeMatrix(Rootemp1,'bM.m')
        print "sauvegarde de M dans bM.m"

        return Rootemp

    def regulRoot1D(self):
        levels = self.xRoot
        nbLines = len(self.lignes)
        lines = self.lignes
        Rootemp = zeros([nbLines,self.xRoot])
        Rootemp = Rootemp.astype(Float64)
        regulRoots = self.roots

        print 'regul 1D'

        #passage en valeur absolue des maxima
        for i in range(nbLines):
            for j in range(levels):
                Rootemp[i,j]=abs(regulRoots[i,j])

        ##regularisation enlever a fin de test des fois qu en 2D ca
#ne serait pas des lignes de maxima
        #regularisation des lignes de maxima pour avoir la plus forte
#valeur en haut
        for i in range(nbLines):
            for j in range((levels-1)):
                temp = Rootemp[i,(j+1)]-Rootemp[i,j]
                if (temp < 0) & (Rootemp[i,(j+1)] != 0) | (temp>10000):
                    Rootemp[i,(j+1)]=Rootemp[i,j]

        Rootemp1 = zeros([levels,nbLines])
```

```
        Rootemp1 = Rootemp1.astype(Float64)
        for i in range(nbLines):
            for j in range(levels):
                Rootemp1[j,i]=Rootemp[i,j]
        self.sauve.writeMatrix(Rootemp1,'bM.m')
        print "sauvegarde de M dans bM.m"

        return Rootemp


    def puiss(self,M,i):
        a = M**i
        #print "M=", M," i=",i," M**i=",a
        return a

    def partition(self):
        q = self.q
        cst = len(q)/2
        levels = self.lgnMax
        nbLines = len(self.lignes)
        M = self.M
        Mtemp = zeros([nbLines,levels])
        Mtemp = Mtemp.astype(Float64)
        Z = zeros([len(q),levels])
        Z = Z.astype(Float64)
        #print "levels", levels, " lignes", nbLines, " q:", q

        for i in q:
            for j in range(nbLines):
                for k in range(levels):
                    #print "i(q)",i," j(lines)",j," k(levels)",k
                    #print "M",M[j,k], " i", i
                    if (M[j,k]- 1e-100) <= 0 :
                        #print "bah si"
                        Mtemp[j,k] = 0
                    else:
                        #print "bah non"
                        Mtemp[j,k] = self.puiss(M[j,k],i)
                    #print "presque"
            temp = sum(Mtemp,axis=0)
            #print " temp som", temp
            #print "range m", len(temp)
            for m in range(len(temp)):
```

```
                Z[(i+cst),m] = temp[m]
                #print "i+cst",(i+cst)," m",m ," temp[m]", temp[m],
 " Z", Z[(i+cst),m]
        #print "init Z2"
        Z2 = zeros([levels,len(q)])
        Z2 = Z2.astype(Float64)

        for i in range(len(q)):
            for j in range(levels):
                #print "fin entree Z2"
                Z2[j,i] = Z[i,j]
                #print "fin sorti Z2"
        self.sauve.writeMatrix(Z2,'bZ.m')
        print "sauvegarde de Z dans bZ.m"
        return Z2

    def W(self,M,Z,q):
        # calcul du poids de Boltzmann
        if ((Z - 1e-100) <= 0 ) | ((M - 1e-100) <= 0 ):
            w = 0
        else :
            w = float((abs(M))**q)/float(Z)
        #print "M=",M," Z=", Z," q=",q," w=",w
        return w

    def holder(self):
        q = self.q
        cst = len(q)/2
        levels = self.lgnMax
        nbLines = len(self.lignes)
        Z=self.Z
        M=self.M
        h = zeros([levels,len(q)])
        h = h.astype(Float64)
        for i in range(levels):
            for j in q:
                for k in range(nbLines):
                    if M[k,i]!=0:
                        h[i,(j+cst)]=h[i,(j+cst)]+(self.W(M[k,i],
#Z[i,(j+cst)],j))*log((abs(M[k,i])))#/log(2)
                    else :
                        h[i,(j+cst)]=h[i,(j+cst)]
        self.sauve.writeMatrix(h,'bH.m')
```

```
        print "sauvegarde de h dans bH.m"
        return h

    def pente_holder(self):
        level = self.lgnMax
        levels = self.lgnMax/3
        scale = self.scale
        scale_inv = zeros([level])
        scale_inv = scale_inv.astype(Float64)
        for i in range(level):
            scale_inv[level-1-i] = scale[0,i]
        #print 'q', self.q
        #print 'scale_inv', scale_inv
        #print 'level', level

        max_scale = scale[0,level]

        h = self.h
        q = self.q
        cst = len(q)/2

        pente_h = zeros([(levels-1),len(q)])
        pente_h = pente_h.astype(Float64)

        for i in range(levels-1):
            #print "ph i:", i
            for j in q:
                #print "ph j", j

                log_a =( log(scale[0,i])-log(scale[0,i+1]))/log(2) #scale no



                if (abs(log_a)-1e-100) >=  0:
                    pente_h[i,(j+cst)] = (h[(i),(j+cst)]-h[(i+1),(j+cst)])/(
                else:
                    pente_h[i,(j+cst)] = 0
        self.sauve.writeMatrix(pente_h,'bpenteh.m')
        print "sauvegarde de pente_h dans bpenteh.m"
        Scale = zeros([level,1])
        Scale = Scale.astype(Float64)
        #print "levels", levels
        for i in range(level):
```

```
            #print " i=",i
            Scale[i,0]=scale[0,i]
            #print "Scale[0,i]", Scale[0,i]
        self.sauve.writeMatrix(Scale,'bScale.m')
        print "sauvegarde de scale dans bScale.m"
        return pente_h


    def d(self):
        level = self.lgnMax
        levels = self.lgnMax
        nbLines = len(self.lignes)
        Z=self.Z
        M=self.M
        q=self.q
        cst = len(q)/2
        d = zeros([levels,len(q)])
        d = d.astype(Float64)
        for i in range(levels):
            for j in q:
                for k in range(nbLines):
                    w=self.W(M[k,i],Z[i,(j+cst)],j)
                    if w!=0:
                        d[i,(j+cst)]=d[i,(j+cst)]+w*log(w)#/log(2)
                    else :
                        d[i,(j+cst)]=d[i,(j+cst)]
        self.sauve.writeMatrix(d,'bD.m')
        print "sauvegarde de d dans bD.m"
        return d

    def pente_d(self):
        level = self.lgnMax
        levels = self.lgnMax/3
        scale = self.scale

        max_scale = scale[0,level-1]

        d = self.D
        q = self.q
        cst = len(q)/2
        pente_d = zeros([(levels-1),len(q)])
        pente_d = pente_d.astype(Float64)
        for i in range(levels-1):
```

```
            #print "pd i", i
            for j in q:
                #print "pd j", j

                #log_a = log(scale[0,i])#/log(2) scale non normalise

                log_a = log(scale[0,i]/max_scale)
# normalisation de scale tel que a_norm = a/a_max
                if (abs(log_a)-1e-100) >=  0:
                    pente_d[i,(j+cst)] = d[(i+1),(j+cst)]/(log_a)
                else:
                    pente_d[i,(j+cst)] = 0
        print "levels", levels
        self.sauve.writeMatrix(pente_d,'bpented.m')
        print "sauvegarde de pente_d dans bpented.m"
        return pente_d


class Boucle:
    def __init__(self):
        win = Frame()
        win.pack()
        Label(win, text='Recommencer Basilik ?').pack(side=TOP)
        Button(win, text='Oui', command=self.encore).pack(side=LEFT)
        Button(win, text='Quitter', command=win.quit).pack(side=RIGHT)
        win.mainloop()


    def encore(self):
        print 'nouvel essaie'
        Maxima()
        sys.exit()



# Use the gzip module for Python version 1.5.2 or higher
gzip = None
try:
    _version = map(string.atoi,
                   string.split(string.split(sys.version)[0], '.'))
    if _version >= [1, 5, 2]:
        try:
            import gzip
```

```
        except ImportError:
            gzip = None
except:
    pass

max = Maxima()
print "fichiers sauvegardes"
```

Annexe E

CODE DE BVISU.PY

```
#! /usr/local/lib/python2.2 python

import sys, os, Numeric, string, fpformat, tkSimpleDialog, tkMessageBox
import math, dislin
from dislin import *
from string import split
from Numeric import *
from Tkinter import *

# bVisu.py
# Written by: Gael Sitzia <sitzia@crm.umontreal.ca>
# Last revision: compatible avec waveplz2_0.py
# permet la creation de fichiers de visualisation de resultat
#

class bLoad:

    def __init__(self, name):

        X = self.words(name)
        Y = self.lines(name)
        Data = self.loadData(name,X,Y)
        Min = self.min(Data,X,Y)
        Max = self.max(Data,X,Y)
        info = Data,X,Y,Min,Max
        return info

    def min(self,data,X,Y):

        min=1
        levels = X
        nbLines = Y
        Rootemp = data
```

```
        min = Rootemp[0,0]

        #regularisation des lignes de maxima pour avoir
#la plus forte valeur en haut
        for i in range(nbLines):
            for j in range(levels):
                temp = Rootemp[j,i] - min
                if (temp < 0) :
                    min = Rootemp[j,i]
        return min

    def max(self,data,X,Y):

        max=1
        levels = X
        nbLines = Y
        Rootemp = data
        max = Rootemp[0,0]

        #regularisation des lignes de maxima pour avoir
#la plus forte valeur en haut
        for i in range(nbLines):
            for j in range(levels):
                temp = Rootemp[j,i] - max
                if (temp > 0) :
                    max = Rootemp[j,i]
        return max

    def loadData(self,name,X,Y):
        Rootemp = zeros([X,Y])
        #print 'X',X,' Y',Y
        Rootemp = Rootemp.astype(Float64)
        lines = 0
        words = 0
        for line in TextFile(name):
            lines = lines +1
            words = len(split(line))
            temp = split(line)
            for x in range(words):
                #print 'lines-1=', (lines-1)
                #print 'x=', x
                Rootemp[x,(lines-1)] = float(temp[x])
        return Rootemp
```

```
    def lines(self,name):
        lines = 0
        #calcul de la hauteur du tableau de donnees
        for line in TextFile(name):
            lines = lines + 1
        return lines

    def words(self,name):
        lines = 0
        words = 0
        max = 0
        #calcul de la largeur max du tableau de donnees
        for line in TextFile(name):
            words = len(split(line))
            if (max - words)<0:
                max = words
        return max



class TextFile:

    def __init__(self, filename, mode = 'r'):
        if string.find(filename, ':/') > 1: # URL
            if mode != 'r':
                raise IOError, "can't write to a URL"
            import urllib
            self.file = urllib.urlopen(filename)
        else:
            filename = os.path.expanduser(filename)
            if mode == 'r':
                if not os.path.exists(filename):
                    raise IOError, (2, 'No such file or directory: '
                                    + filename)
                if filename[-2:] == '.Z':
                    self.file = os.popen("uncompress -c "
+ filename, mode)
                elif filename[-3:] == '.gz':
                    if gzip is None:
                        self.file = os.popen("gunzip -c "
+ filename, mode)
```

```
                    else:
                        self.file = gzip.GzipFile(filename, 'rb')
                elif filename[-4:] == '.bz2':
                    self.file = os.popen("bzip2 -dc " + filename, mode)
                else:
                    try:
                        self.file = open(filename, mode)
                    except IOError, details:
                        if type(details) == type(()):
                                details = details + (filename,)
                        raise IOError, details
            elif mode == 'w':
                if filename[-2:] == '.Z':
                    self.file = os.popen("compress > " + filename, mode)
                elif filename[-3:] == '.gz':
                    if gzip is None:
                        self.file = os.popen("gzip > " + filename, mode)
                    else:
                        self.file = gzip.GzipFile(filename, 'wb')
                elif filename[-4:] == '.bz2':
                    self.file = os.popen("bzip2 > " + filename, mode)
                else:
                    try:
                        self.file = open(filename, mode)
                    except IOError, details:
                        if type(details) == type(()):
                                details = details + (filename,)
                        raise IOError, details
            elif mode == 'a':
                if filename[-2:] == '.Z':
                    raise IOError, (0, "Can't append to .Z files")
                elif filename[-3:] == '.gz':
                    if gzip is None:
                        self.file = os.popen("gzip >> " + filename, "w")
                    else:
                        self.file = gzip.GzipFile(filename, 'ab')
                else:
                    self.file = open(filename, mode)
            else:
                raise IOError, (0, 'Illegal mode: ' + repr(mode))

        def __del__(self):
    self.close()
```

```
    def __getitem__(self, item):
line = self.file.readline()
if not line:
    raise IndexError
return line

    def readline(self):
return self.file.readline()

    def readlines(self):
return self.file.readlines()

    def write(self, data):
self.file.write(data)

    def writelines(self, list):
for line in list:
    self.file.write(line)

    def close(self):
self.file.close()

    def flush(self):
        self.file.flush()


class SauveInfo:

    def readMatrix(filename):
        rows = []
        for line in TextFile(self,filename):
            columns = []
            for number in string.split(line):
                columns.append(string.atof(number))
            rows.append(columns)
        return Numeric.array(rows)

    def writeMatrix(self,a, filename):
        file = TextFile(filename, 'w')
        for line in a:
            for number in line:
                file.write(`number` + ' ')
```

```
            file.write('\n')
        file.close()


class PrepabVisu:

    def __init__(self):

        self.fich = ('data.m','bpented.m','bpenteh.m')
        self.sauve = SauveInfo()
        self.dirpath = os.getcwd() #donne le path courant
        src = self.dirpath

        #calcul de la taille de l image a gerer
        self.nbLigne = int(self.nblin(self.data))
        self.nbCol = int(self.nbcol(self.data))



        #test signal 2D:
        if ((self.nbLigne - 1) > 0) & ((self.nbCol - 1) > 0) :

            #test signal > 512*512
            if ((self.nbLigne - 512) > 0) | ((self.nbCol - 512) > 0) :
                Nx = int(self.nbLigne/256)
                Ny = int(self.nbCol/256)

                if Nx<1:Nx=1
                if Ny<1:Ny=1

                resbask = zeros([int(Nx),2*int(Ny)])
                resbask = Rootemp.astype(Float64)

                for i in range(Nx-1):
                    for j in range(Ny-1):
                        #transfert dans dir tempxxxx/temp
                        os.chdir(self.dirtampon)

                        #creation du data.m situe en ij dans le masterdata
                        datatemp=self.extractData(self.MasterData,
self.nbLigne,self.nbCol,i,Nx,j,Ny)
                        self.sauve.writeMatrix(datatemp,'data.m')
```

```
                #activ wavelet & activ chain & activ basilik40.py
                if test == 'posix':
                    os.popen('wavelet2d wave.i')
                    os.popen('chain2d wave.i')
                    os.popen('python basilik40.py')


                else:
                    os.startfile ('wavelet2d.exe')
                    os.startfile ('chain2d.exe')
                    os.startfile ('basilik40.py')

                bph = bLoad('bpenteh.m')
                bpd = bLoad('bpented.m')

                bpenteh = bph[0]
                bpented = bpd[0]

                resbask[i,2*j]   = bpenteh[0,4]
                resbask[i,2*j+1] = bpented[0,4]

                #creation  dir ij format 00 00
                tempname=self.repNxNy(i,j)
                dest = os.path.join(self.dirwrktmp,tempname)
                os.mkdir(dest)

                #copie des fichiers de sortie dans tempxxxxxx/iijj/
                src = os.path.join(self.dirtampon)

                for k in range(len(self.fichsort2D)):
                    srck = os.path.join(src,self.fichsort2D[k])
                    destk = os.path.join(dest)#,fichsort2D[k])
                    shutil.copy(srck,destk)

                #remonte d 1 a tempxxxxx/
                #transfert sous la directorie tempxxxxx/
                os.chdir(self.dirwrktmp)

        else : #image de 512*512 ou moins

            #transfert dans dir tempxxxx/temp
            os.chdir(self.dirtampon)
```

```
                    #creation du data.m situe en ij dans le masterdata
                    datatemp=self.regulData2D(self.MasterData,self.nbLigne,
self.nbCol)
                    self.sauve.writeMatrix(datatemp,'data.m')

                    resbask = zeros([1,2])
                    resbask = Rootemp.astype(Float64)

                    #activ wavelet & activ chain & activ basilik40.py
                    if test == 'posix':
                        os.popen('wavelet2d wave.i')
                        os.popen('chain2d wave.i')
                        os.popen('python basilik40.py')

                    else:
                        os.startfile ('wavelet2d.exe')
                        os.startfile ('chain2d.exe')
                        os.startfile ('basilik40.py')

                    bph = bLoad('bpenteh.m')
                    bpd = bLoad('bpented.m')

                    bpenteh = bph[0]
                    bpented = bpd[0]

                    resbask[0,0] = bpenteh[0,4]
                    resbask[0,1] = bpented[0,4]


                    #creation  dir ij format 000 000
                    tempname='000000'
                    dest = os.path.join(self.dirwrktmp,tempname)
                    os.mkdir(dest)

                    #copie des fichiers de sortie dans tempxxxxxx/iijj/
                    src = os.path.join(self.dirtampon)

                    for k in range(len(self.fichsort2D)):
                        srck = os.path.join(src,self.fichsort2D[k])
                        destk = os.path.join(dest)#,fichsort2D[k])
                        shutil.copy(srck,destk)

                    #remonte d 1 a tempxxxxx/
```

```
                    #transfert sous la directorie tempxxxxx/
                    os.chdir(self.dirwrktmp)



                #le signal 1D
                else:
                    #test signal >  1024*1 ou 1*1024
                    if ((self.nbLigne - 1024) > 0) | ((self.nbCol - 1024) > 0) :
                        print '10'
                        N=1
                        Nx = int(self.nbLigne/512)
                        Ny = int(self.nbCol/512)
                        print 'Nx',Nx, ' Ny',Ny
                        if Nx<1:N=Ny
                        if Ny<1:N=Nx
                        else:Nx=Nx

                        resbask = zeros([1,2*int(N)])
                        resbask = Rootemp.astype(Float64)

                        for i in range(N-1):
                            print 'i',i

                            #transfert dans dir tempxxxx/temp
                            os.chdir(self.dirtampon)
                            print '100'
                            #creation du data.m situe en ij dans le masterdata
                            datatemp=self.extractData1D(self.MasterData,
self.nbLigne,self.nbCol,i,Nx,1,Ny)
                            self.sauve.writeMatrix(datatemp,'data.m')
                            print '1000'
                            #activ wavelet & activ chain & activ basilik40.py
                            if test == 'posix':
                                print '11'
                                os.popen('wavelet1d wave.i')
                                print '12'
                                os.popen('chain1d wave.i')
                                print '13'
                                os.popen('python basilik40.py')
                                print '14'
```

```
                #si la platforme est autre, ne marche que sous windo
            else:
                os.startfile ('wavelet1d.exe')
                os.startfile ('chain1d.exe')
                os.startfile ('basilik40.py')

            bph = bLoad('bpenteh.m')
            bpd = bLoad('bpented.m')

            bpenteh = bph[0]
            bpented = bpd[0]

            resbask[0,2*i]   = bpenteh[0,4]
            resbask[0,2*i+1] = bpented[0,4]

            #creation  dir ij format 00 00
            tempname=self.repN(i)
            dest = os.path.join(self.dirwrktmp,tempname)
            os.mkdir(dest)

            #copie des fichiers de sortie dans tempxxxxxx/iijj/
            src = os.path.join(self.dirtampon)

            for k in range(len(self.fichsort1D)):
                srck = os.path.join(src,self.fichsort1D[k])
                destk = os.path.join(dest)#,fichsort1D[k])
                shutil.copy(srck,destk)

            #remonte d 1 a tempxxxxx/
            #transfert sous la directorie tempxxxxx/
            os.chdir(self.dirwrktmp)

    else:
        #signal 1D > 1024

        #transfert dans dir tempxxxx/temp
        os.chdir(self.dirtampon)

        #creation du data.m situe en ij dans le masterdata
        datatemp=self.MasterData
        self.sauve.writeMatrix(datatemp,'data.m')

        resbask = zeros([1,2])
```

```
                resbask = Rootemp.astype(Float64)

                bph = bLoad('bpenteh.m')
                bpd = bLoad('bpented.m')

                bpenteh = bph[0]
                bpented = bpd[0]

                resbask[0,0] = bpenteh[0,4]
                resbask[0,1] = bpented[0,4]

                #creation  dir ij format 000 000
                tempname='000000'
                dest = os.path.join(self.dirwrktmp,tempname)
                os.mkdir(src)

                #copie des fichiers de sortie dans tempxxxxxx/iijj/
                src = os.path.join(self.dirtampon)

                for k in range(len(self.fichsort1D)):
                    srck = os.path.join(src,self.fichsort1D[k])
                    destk = os.path.join(dest)#,fichsort1D[k])
                    shutil.copy(srck,destk)

                #remonte d 1 a tempxxxxx/
                #transfert sous la directorie tempxxxxx/
                os.chdir(self.dirwrktmp)

        #enlever la dir tampon avec tous les calculs#
        #os.chdir(self.dirtampon)
        src = self.dirtampon
        liste =os.listdir(src)

        for i in range(len(liste)):
            srci = os.path.join(src,liste[i])
            os.remove(srci)

        self.sauve.writeMatrix(resbask,'resbask.m')
        os.chdir(self.dirpath)
        os.rmdir(self.dirtampon)


max = Maxima()
```

Annexe F

CODE DE BVISU42.M

```
clear all;
load bD.m;
load bH.m;
load bM.m;
load bpented.m;
load bpenteh.m;
load bZ.m;
load bScale.m;

sSc = size(bScale);
sZ = size(bZ);
sd = size(bpented);

q = sZ(2);
a = sZ(1);
b = sd(1);


figure;

subplot(3,3,1)
hold on
    plot(log2(bScale(:,1)),log2(bZ(:,1)),'yo')
    plot(log2(bScale(:,1)),log2(bZ(:,2)),'ro')
    plot(log2(bScale(:,1)),log2(bZ(:,3)),'go')
    plot(log2(bScale(:,1)),log2(bZ(:,4)),'bo')
    plot(log2(bScale(:,1)),log2(bZ(:,5)),'kd')
    plot(log2(bScale(:,1)),log2(bZ(:,6)),'bs')
    plot(log2(bScale(:,1)),log2(bZ(:,7)),'gs')
    plot(log2(bScale(:,1)),log2(bZ(:,8)),'rs')
    plot(log2(bScale(:,1)),log2(bZ(:,9)),'ys')
    title('\it{log2(scale) log2(Z)}','FontSize', 10)
hold off
```

```
subplot(3,3,2)
hold on
    plot(log2(bScale(:,1)),bH(:,1),'yo')
    plot(log2(bScale(:,1)),bH(:,2),'ro')
    plot(log2(bScale(:,1)),bH(:,3),'go')
    plot(log2(bScale(:,1)),bH(:,4),'bo')
    plot(log2(bScale(:,1)),bH(:,5),'kd')
    plot(log2(bScale(:,1)),bH(:,6),'bs')
    plot(log2(bScale(:,1)),bH(:,7),'gs')
    plot(log2(bScale(:,1)),bH(:,8),'rs')
    plot(log2(bScale(:,1)),bH(:,9),'ys')
    title('\it{log2(scale) holder}','FontSize', 10)
hold off

subplot(3,3,3)
hold on
    plot(log2(bScale(:,1)),bD(:,1),'yo')
    plot(log2(bScale(:,1)),bD(:,2),'ro')
    plot(log2(bScale(:,1)),bD(:,3),'go')
    plot(log2(bScale(:,1)),bD(:,4),'bo')
    plot(log2(bScale(:,1)),bD(:,5),'kd')
    plot(log2(bScale(:,1)),bD(:,6),'bs')
    plot(log2(bScale(:,1)),bD(:,7),'gs')
    plot(log2(bScale(:,1)),bD(:,8),'rs')
    plot(log2(bScale(:,1)),bD(:,9),'ys')
    title('\it{log2(scale) dimension}','FontSize', 10)
hold off

subplot(3,3,4)
hold on
    plot(bH(:,1),bD(:,1),'yo')
    plot(bH(:,2),bD(:,2),'ro')
    plot(bH(:,3),bD(:,3),'go')
    plot(bH(:,4),bD(:,4),'bo')
    plot(bH(:,5),bD(:,5),'kd')
    plot(bH(:,6),bD(:,6),'bs')
    plot(bH(:,7),bD(:,7),'gs')
    plot(bH(:,8),bD(:,8),'rs')
    plot(bH(:,9),bD(:,9),'ys')
    title('\it{holder dimension}','FontSize', 10)
hold off
```

```
subplot(3,3,5)
hold on
    plot(bpenteh(:,1),bpented(:,1),'yo')
    plot(bpenteh(:,2),bpented(:,2),'ro')
    plot(bpenteh(:,3),bpented(:,3),'go')
    plot(bpenteh(:,4),bpented(:,4),'bo')
    plot(bpenteh(:,5),bpented(:,5),'kd')
    plot(bpenteh(:,6),bpented(:,6),'bs')
    plot(bpenteh(:,7),bpented(:,7),'gs')
    plot(bpenteh(:,8),bpented(:,8),'rs')
    plot(bpenteh(:,9),bpented(:,9),'ys')
    title('\it{pente holder vs pente dimension}','FontSize', 10)
hold off

subplot(3,3,6)
hold on
    plot(log2(bScale(1:b,1)),bpenteh(:,1),'yo')
    plot(log2(bScale(1:b,1)),bpenteh(:,2),'ro')
    plot(log2(bScale(1:b,1)),bpenteh(:,3),'go')
    plot(log2(bScale(1:b,1)),bpenteh(:,4),'bo')
    plot(log2(bScale(1:b,1)),bpenteh(:,5),'kd')
    plot(log2(bScale(1:b,1)),bpenteh(:,6),'bs')
    plot(log2(bScale(1:b,1)),bpenteh(:,7),'gs')
    plot(log2(bScale(1:b,1)),bpenteh(:,8),'rs')
    plot(log2(bScale(1:b,1)),bpenteh(:,9),'ys')
    title('\it{log2(scale) vs pente holder}','FontSize', 10)
hold off

subplot(3,3,7)
hold on
    plot(log2(bScale(1:b,1)),bpented(:,1),'yo')
    plot(log2(bScale(1:b,1)),bpented(:,2),'ro')
    plot(log2(bScale(1:b,1)),bpented(:,3),'go')
    plot(log2(bScale(1:b,1)),bpented(:,4),'bo')
    plot(log2(bScale(1:b,1)),bpented(:,5),'kd')
    plot(log2(bScale(1:b,1)),bpented(:,6),'bs')
    plot(log2(bScale(1:b,1)),bpented(:,7),'gs')
    plot(log2(bScale(1:b,1)),bpented(:,8),'rs')
    plot(log2(bScale(1:b,1)),bpented(:,9),'ys')
    title('\it{log2(scale) vs log2(pente dimension)}','FontSize', 10)
hold off

subplot(3,3,8)
```

```
hold on
    plot(bM)
    title('\it{echelle vs M }','FontSize', 10)
hold off
```

Annexe G

CODE DE BVISU43.M

```
clear all;
load bD.m;
load bH.m;
load bM.m;
load bpented.m;
load bpenteh.m;
load bZ.m;
load bScale.m;

bpenteh = bpenteh - 0.25;

sSc = size(bScale);
sZ = size(bZ);
sd = size(bpented);

q = sZ(2);
a = sZ(1);
b = sd(1);
c = floor(b/4)


figure;

subplot(3,3,1)
hold on
    plot(log2(bScale(:,1)),log2(bZ(:,1)),'yo')
    plot(log2(bScale(:,1)),log2(bZ(:,2)),'ro')
    plot(log2(bScale(:,1)),log2(bZ(:,3)),'go')
    plot(log2(bScale(:,1)),log2(bZ(:,4)),'bo')
    plot(log2(bScale(:,1)),log2(bZ(:,5)),'kd')
    plot(log2(bScale(:,1)),log2(bZ(:,6)),'bs')
    plot(log2(bScale(:,1)),log2(bZ(:,7)),'gs')
    plot(log2(bScale(:,1)),log2(bZ(:,8)),'rs')
```

```
    plot(log2(bScale(:,1)),log2(bZ(:,9)),'ys')
    title('\it{log2(scale) log2(Z)}','FontSize', 10)
hold off

subplot(3,3,2)
hold on
    plot(log2(bScale(:,1)),bH(:,1),'yo')
    plot(log2(bScale(:,1)),bH(:,2),'ro')
    plot(log2(bScale(:,1)),bH(:,3),'go')
    plot(log2(bScale(:,1)),bH(:,4),'bo')
    plot(log2(bScale(:,1)),bH(:,5),'kd')
    plot(log2(bScale(:,1)),bH(:,6),'bs')
    plot(log2(bScale(:,1)),bH(:,7),'gs')
    plot(log2(bScale(:,1)),bH(:,8),'rs')
    plot(log2(bScale(:,1)),bH(:,9),'ys')
    title('\it{log2(scale) holder}','FontSize', 10)
hold off

subplot(3,3,3)
hold on
    plot(log2(bScale(:,1)),bD(:,1),'yo')
    plot(log2(bScale(:,1)),bD(:,2),'ro')
    plot(log2(bScale(:,1)),bD(:,3),'go')
    plot(log2(bScale(:,1)),bD(:,4),'bo')
    plot(log2(bScale(:,1)),bD(:,5),'kd')
    plot(log2(bScale(:,1)),bD(:,6),'bs')
    plot(log2(bScale(:,1)),bD(:,7),'gs')
    plot(log2(bScale(:,1)),bD(:,8),'rs')
    plot(log2(bScale(:,1)),bD(:,9),'ys')
    title('\it{log2(scale) dimension}','FontSize', 10)
hold off

subplot(3,3,4)
hold on
    plot(bH(:,1),bD(:,1),'yo')
    plot(bH(:,2),bD(:,2),'ro')
    plot(bH(:,3),bD(:,3),'go')
    plot(bH(:,4),bD(:,4),'bo')
    plot(bH(:,5),bD(:,5),'kd')
    plot(bH(:,6),bD(:,6),'bs')
    plot(bH(:,7),bD(:,7),'gs')
    plot(bH(:,8),bD(:,8),'rs')
    plot(bH(:,9),bD(:,9),'ys')
```

```
    title('\it{holder dimension}','FontSize', 10)
hold off

subplot(3,3,5)
hold on
    plot(bpenteh(1,1),bpented(1,1),'yo')
    plot(bpenteh(1,2),bpented(1,2),'ro')
    plot(bpenteh(1,3),bpented(1,3),'go')
    plot(bpenteh(1,4),bpented(1,4),'bo')
    plot(bpenteh(1,5),bpented(1,5),'kd')
    plot(bpenteh(1,6),bpented(1,6),'bs')
    plot(bpenteh(1,7),bpented(1,7),'gs')
    plot(bpenteh(1,8),bpented(1,8),'rs')
    plot(bpenteh(1,9),bpented(1,9),'ys')
    title('\it{pente holder vs pente dimension}','FontSize', 10)
hold off

subplot(3,3,6)
hold on
    plot(log2(bScale(1:b,1)),bpenteh(:,1),'yo')
    plot(log2(bScale(1:b,1)),bpenteh(:,2),'ro')
    plot(log2(bScale(1:b,1)),bpenteh(:,3),'go')
    plot(log2(bScale(1:b,1)),bpenteh(:,4),'bo')
    plot(log2(bScale(1:b,1)),bpenteh(:,5),'kd')
    plot(log2(bScale(1:b,1)),bpenteh(:,6),'bs')
    plot(log2(bScale(1:b,1)),bpenteh(:,7),'gs')
    plot(log2(bScale(1:b,1)),bpenteh(:,8),'rs')
    plot(log2(bScale(1:b,1)),bpenteh(:,9),'ys')
    title('\it{log2(scale) vs pente holder}','FontSize', 10)
hold off

subplot(3,3,7)
hold on
    plot(log2(bScale(1:b,1)),bpented(:,1),'yo')
    plot(log2(bScale(1:b,1)),bpented(:,2),'ro')
    plot(log2(bScale(1:b,1)),bpented(:,3),'go')
    plot(log2(bScale(1:b,1)),bpented(:,4),'bo')
    plot(log2(bScale(1:b,1)),bpented(:,5),'kd')
    plot(log2(bScale(1:b,1)),bpented(:,6),'bs')
    plot(log2(bScale(1:b,1)),bpented(:,7),'gs')
    plot(log2(bScale(1:b,1)),bpented(:,8),'rs')
    plot(log2(bScale(1:b,1)),bpented(:,9),'ys')
    title('\it{log2(scale) vs log2(pente dimension)}','FontSize', 10)
```

```
hold off

subplot(3,3,8)
hold on
    plot(bM)
    title('\it{echelle vs M }','FontSize', 10)
hold off
```

Annexe H

CODE DE BVISU6.M

```
clear all;

load data.m
load resbask.m

sdata = size(data);
sresbask = size(resbask);

cst = 25;
rap = 1/cst; %1/256;

lgndata = sdata(1);
coldata = sdata(2);
C = coldata;
L = lgndata;

lgnresbask = sresbask(1);
colresbask = sresbask(2);
Nx = colresbask/2;
Ny = lgnresbask;

[x,y]=meshgrid(0:rap:Nx,0:rap:Ny);
z=0.*x;
if (Nx<=1)
    i=1;
    if (Ny>1)
        for j = 1:Ny
            for x1 = ((i-1)*cst+1):1:(i*cst)
                for y1 =  ((j-1)*cst+1):1:(j*cst)
                    z(x1,y1) = resbask(j,(2*i-1));
                end
            end
        end
```

```
    end
    if (Ny<=1)
        j=1;
        for x1 = ((i-1)*cst+1):1:(i*cst)
            for y1 =  ((j-1)*cst+1):1:(j*cst)
                z(x1,y1) = resbask(j,(2*i-1));
            end
        end
    end
end
if (Nx>1)
    for i = 1:Nx
        if (Ny<=1)
            j=1;
            for x1 = ((i-1)*cst+1):1:(i*cst)
                for y1 =  ((j-1)*cst+1):1:(j*cst)
                    z(x1,y1) = resbask(j,(2*i-1));
                end
            end
        end
        if (Ny>1)
            for j = 1:Ny
                for x1 = ((i-1)*cst+1):1:(i*cst)
                    for y1 =  ((j-1)*cst+1):1:(j*cst)
                        z(x1,y1) = resbask(j,(2*i-1));
                    end
                end
            end
        end
    end
end

figure;

%subplot(1,2,1)
hold on
    imshow(data)
hold off

figure
%subplot(1,2,2)
hold on
    mesh(z);
```

```
%     surf(x,y,z,'FaceColor','interp','EdgeColor',
%'none','FaceLighting','phong')
%     daspect([100 100 1])
%     axis tight
    view(90,90)
%     camlight left
    grid on
hold off
```

Annexe I

CODE COMPLÉMENTAIRE POUR LA VISUALISATION À L'AIDE DE MATLAB

```
function showIm (im)

pcolor (im)
colormap default;
shading interp


function showLine (b)

load Is.m
load Js.m
load Ks.m
load scale.m

szk = size( Ks )

scales = zeros (szk(1),szk(2));
maxLevel = max (max (Ks))
maxKs = (maxLevel)*ones (szk(1),szk(2))
Ks
Ks = maxKs - Ks

plot3 ( transpose(Js), transpose(Is), transpose (Ks) )


function showSurf (m)

surf (m)
shading interp
colormap default
```

BIBLIOGRAPHIE

[1] Characterization of mammographic parenchymal pattern by fractal dimension, C. B. Cadwell, S.J. Stapleton, D.W. Holdsworth, R.A. Jong, W.J. Weiser, G. Cooke, and M.J. Yaffe, (Phys. Med. Biol. 35,235-247 1990)

[2] Fractal analysis of clustered microcalcifications in mammograms, F. Lefebvre, H. Benali et E. Kahn (Acta Sterco. , Vol. 11 pp.611-616 1992)

[3] Les ondelettes : algorithmes et applications, Y. Meyer (Armand Colin Editeur, Paris, 1992)

[4] Fractal modeling and segmentation for the enhancement of microcalcifications in digital mammograms, H. Li, R. Liu, S. Lo (IEEE Trans. Medical imaging, Vol.16 No6 p.785-798 Dec. 1997)

[5] Image feature extractation for mass detection in digital mammography : influence of wavelet analysis, W. Qian, L. Li, L. Clarke (Medical Physics Vol. 26 No3 p.402-408 1999)

[6] The thermodynamics of fractals revisited with wavelets, A. Arneodo, E. Bacry, J.F. Muzy (Physica A 213 (1995) 232-275)

[7] Wavelet based multifractal formalism : Applications to DNA sequences, satellite images of the cloud structure and stock market data, A. Arneodo, B. Audit, N. Decoster, J-F. Muzy, and C. Vaillant (Physica A 254 (15 mai 1998) p.24-45).

[8] Thermodynamics of fractal signals based on wavelet analysis : application to fully developed turbulence data and DNA sequences, A. Arneodo, B. Audit, E. Bacry, S. Manneville, J.-F. Muzy and S. G. Roux. (Physica A 254, 24-45 (1998)).

[9] Wavelets, Theory and Applications , A. Arneodo (Oxford University Press, 1996)

[10] A wavelet tour of signal processing, S. Mallat (Academic Press, 1998)

[11] Singularity detection and processing with wavelets, S. Mallat, W.L. Hwang (IEEE Transactions on Information Theory, vol. 32, no. 2, March 1992)

[12] Characterization of signals from multiscale edges, S. Mallat, S. Zhong (IEEE Transaction on Pattern Analysis and Machine Intelligence, vol. 14, No. 7, p. 710-732, July 1992)

[13] Wavelet-based multifractal formalism to assist in diagnosis in digitized mamograms, P. Kestener, J.M. Lina, P. St-Jean and A. Arnéodo (Image Anal Steriol 2001 ;20 :169-174, 2001)

[14] Ondelettes, multifractales et turbulences, de l'ADN aux croissances cristallines, A. Arneodo, F. Argoul, E. Bacry, J. Elezgaray et J-F.Muzy (Sciences en actes, Diderot Editeur, arts et sciences, 1995)

[15] Entropie mutli-échelle : définition et applications, Jean-Luc Starck (université Paris-Sud, mémoire , novembre 1999)

[16] Physique statistique, Nicolas Vandewalle (GRASP, institut de Physique de l'université de Liège, 2003)

[17] Fractals et lois d'échelle - Volume I, P. Abry, P. Gonçalvès, J. Lévy Véhel (2001)

[18] Fractals et lois d'échelle : Méthodes d'ondelettes pour analyse multifractale de fonctions, Volume I, Chapitre 2, S. Jaffard (2001)

[19] Scintillator detectors, Claude Leroy (LPS, département de Physique de l'Université de Montréal, 2002)

[20] Introduction to digital radiography : the role of digital radiography in medical imaging, Kodak (Kodak Canada inc., 2000)

[21] The Encyclopaedia of Medical Imaging Volume I : Physics, Techniques and Procedures Gustav K. von Schulthess MD, PhD, Professor of Nuclear Medicine, University of Zürich, Switzerland, Hans-Jørgen Smith MD, PhD, Professor of Radiology, University of Oslo, Norway.

[22] Analyse multifractale d'images de surfaces rugueuses à l'aide de la transformation en ondelettes, Nicolas Decoster (Thèse de l'Université de Bordeaux I, pour l'obtention du grade de docteur en informatique)

[23] Medical Imaging Systems, Albert Marcovski (Information and systems science series, Thomas Kailath, 2002)

[24] The physics of radiology, Harold Elford Johns and John Robert Cunningham (Charles C. Thomas Publisher, 1983)

[25] Principles of Computorized Tomographic Imaging, A.C. Kak and Malcolm Slaney (IEEE Press, 1988)

[26] Uber die bewegende Kraft der Wärme, Rudolf Clausius (Berlin academy, 18 février 1850)

[27] Thermal Physics, Charles Kittel and Herbert Kroemer (W.H. Freeman and company, New York, 1998)

[28] Thermodynamics, kinetic theory, and statistical thermodynamics, Francis W. Sears and Gerhard L. Salinger (Addison-Wesley Publishing Company, 1975)

[29] Thermodynamique : fondements et applications, Josée-Philippe Pérez (Dunod, 3e édition, 2001)

[30] Entropie, information : un concept proteiforme, Roger Balian (Texte de la 239ième conférence de tous les savoirs, 26 août 2000)

[31] Cours de thermodynamique, Jean Louis Deiss (Université Louis Pasteur - Strasbourg 1 - France, 2003)

[32] Introduction à la thermodynamique, Claire Lhuillier et Jean Rousse (Dunod, 2nd édition, 1988)

[33] Tables of X-Ray Mass Attenuation Coefficients and Mass Energy-Absorption Coefficients, J. H. Hubbell+ and S. M. Seltzer Ionizing Radiation Division, Physics Laboratory National Institute of Standards and Technology Gaithersburg, MD 20899 (http ://physics.nist.gov/PhysRefData/XrayMassCoef/cover.html)

[34] L'anatomie humaine, Pr. Gérard Outrequin et Bertrand Bouteillier (CHUM Limoges, 2003)

[35] Anatomo pathologie mammaire, Pr. J.P. Bellocq, Pr. M.P. Chenard (U.L.P. Strasbourg, 2003)

[36] Cours d'histologie, Pr. Jacques Poirier et Dr. Jean-Michel André (Université Paris VI Pierre et Marie Curie, Faculté de Médecine Pitié-Salpêtrière, 2003)

[37] Au coeur de l'extra-ordinaire, Henri Broch (l'horizon chimérique, 1992)

[38] Documentation des librairies CRM_wt1d, CRM_wt2d et CRM_chaining, Yan Basile-Bellavance et Jean-Marc Lina(Groupe PHYSNUM, Centre de Recherches Mathématiques de l'Université de Montréal, 2002)