Institut Superieur
d'Informatique
de Modelisation
et de leurs Applications

complexe des cezeaux
BP 125 - 63173 aubiere cedex

Rapport de Projet 2ème année

Implantation de la methode EMD en C avec interface Matlab

Présenté par : Tarik Boustane
et Gwénolé Quellec
Responsable : Pierre Chainais

Projet de 100 heures
de Novembre 2003 à Mars 2004

Remerciements

On tient à remercier particulièrement notre maître de projet, Pierre CHAINAIS, enseignant-chercheur à l'ISIMA de Clermont-Ferrand, pour ses conseils avisés et ses encouragements, sa sympathie, qui nous ont beaucoup aidés dans la réalisation et la réussite de ce projet.

Résumé

En 1998, l'**Empirical Mode Decomposition** (EMD), une nouvelle méthode d'analyse du signal a été proposée par N.E. Huang. Cette méthode, qui est purement algorithmique, a déjà été implémentée sous Matlab par l'équipe de P. Flandrin (ENS Lyon). Mais le nombre d'opérations pour décomposer un signal par EMD croît rapidement avec sa taille, ce qui implique de longs **temps de calcul** sous Matlab.

Le travail effectué dans ce projet a consisté à implanter l'algorithme en langage C pour résoudre ce problème. Il a fallu pour cela coder une méthode d'**interpolation**, la plus adaptée étant l'interpolation cubique par des splines, et définir des paramètres et critères d'arrêt. On a dû également résoudre des problèmes causés par d'importants effets de bords. Pour rendre l'utilisation du programme plus pratique, on a réalisé une **interface avec Matlab**. Finalement, on a analysé quelques exemples susceptibles d'ouvrir des perspectives pour la méthode.

Mots clés : Empirical Mode Decomposition, temps de calcul, interpolation, interface avec Matlab

Abstract

In 1998, the **Empirical Mode Decomposition** (EMD), a new signal analysis technique has been pioneered by N.E. Huang. This method, only defined by an algorithm, has already been implemented into Matlab by P. Flandrin's team (ENS Lyon). Anyway, the number of calculations to decompose a signal by the EMD quickly increases with its size, which involves a considerable **computation time** using Matlab.

The work we carried out in this project consisted in coding the algorithm in C language to solve this problem. In that purpose, we had to code an **interpolation** method, the most suited being the cubic spline interpolation, and define parameters and stopping criteria. We also had to solve problems caused by considerable side effects. To make the program more user-friendly, we built an **interface with Matlab**. Finally, we analysed examples likely to lead to new prospects for the technique.

Keywords : Empirical Mode Decomposition, computation time, interpolation, interface with Matlab

Table des matières

Remerciements

Résumé

Table des matières

Table des figures

Introduction **6**

1 Présentation du problème **7**
1.1 Intérêts de la méthode EMD 7
1.2 Travail demandé 7
1.3 Algorithme de principe 7

2 Méthodes utilisées **9**
2.1 Interpolation cubique par des splines 9
2.2 Choix des valeurs à tabuler pour l'interpolation 9
2.3 Précision des calculs 10
2.4 Interface C/Matlab 13

3 Résultats et discussion **14**
3.1 Analyse d'un exemple 14
3.2 Temps de calcul du programme 15
3.3 Problème non résolu 15

Conclusion **17**

Références bibliographiques

Table des figures

1 Enveloppes supérieures et inférieures d'un signal 11
2 Choix des valeurs à tabuler pour l'interpolation 11
3 Influence des valeurs tabulées pour l'interpolation 12
4 Somme d'un signal triangulaire et d'une sinusoïde 14
5 Temps de calcul en fonction de la taille de l'instance 15
6 Problème des termes parasites 16

Introduction

Dans le cadre de notre deuxième d'études en spécialité calcul scientifique et modélisation à l'ISIMA, nous avons réalisé un projet de cent heures réparties sur cinq mois, entre novembre 2003 et mars 2004.

Il y a six ans, une nouvelle méthode d'analyse du signal a été proposée par N.E. Huang. Cette méthode a pour particularité de n'être définie que par un algorithme et, bien que sans fondements théoriques, elle s'avère très efficace. Des recherches sont effectuées pour tenter de justifier la méthode. Afin de comprendre pourquoi elle fonctionne aussi bien, on a besoin d'outils pour l'étudier, d'où l'intérêt de la coder.

Cet algorithme a déjà été implémenté sous Matlab. Il est néanmoins intéressant de le programmer en C, afin de pouvoir étudier des signaux de grande taille en minimisant le temps de calcul. Ainsi le but de notre projet a été d'implémenter la méthode en C avec interface Matlab.

Dans un premier temps, nous allons présenter notre étude en décrivant le principe de l'EMD. Puis nous détaillerons les méthodes utilisées dans notre programme. Enfin, nous analyserons le résultat de l'exécution de notre programme sur un exemple et discuterons de ses performances.

1 Présentation du problème

1.1 Intérêts de la méthode EMD

La méthode EMD a pour objectif de décomposer des signaux en différents modes ayant une bonne signification physique. Cela veut dire notamment que la connaissance de ces modes permet de comprendre de manière intuitive le contenu fréquentiel du signal.

Les méthodes classiques décomposent un signal sur une base de fonctions propre à la méthode, qui sont donc indépendantes du signal étudié. Ainsi, par exemple, la transformée de Fourier décompose un signal en une somme de sinusoïdes. La méthode des ondelettes, quant à elle, utilise pour fonctions de base les *"ondelettes"*, qui sont des fonctions localisées : la signification physique des différents modes est donc déjà meilleure.

La méthode EMD, au contraire, a une approche adaptative : pour chaque signal étudié, une nouvelle base de fonctions est construite. Ainsi ces modes décriront mieux le signal. Ces modes, appelés IMFs (*Intrinsic Mode Functions*), ont pour seule caractéristique d'être de moyenne nulle.

1.2 Travail demandé

L'algorithme a déjà été implémenté sous Matlab par l'équipe de P. Flandrin (ENS Lyon). Cependant, le nombre d'opérations à réaliser pour décomposer un signal par la méthode EMD croît rapidement avec la taille des signaux. Or l'inconvénient de la programmation sous Matlab, qui utilise un langage de haut niveau, est que les temps de calculs sont relativement importants.

Pour résoudre ce problème, il était donc intéressant de programmer la méthode en C. Cependant, il est commode de pouvoir appeler le programme depuis l'invite de commande Matlab. En effet, ceci permet notamment d'utiliser l'interface graphique de Matlab pour visualiser les sorties du programme.

Il nous a donc été demandé d'écrire un programme en C, réalisant la décomposition d'un signal par la méthode EMD, qui puisse être compilé et exécuté depuis l'invite de commande Matlab, en réalisant une interface.

1.3 Algorithme de principe

L'EMD, tel que l'explique l'article *On empirical mode decomposition and its algorithms* ([1]), considère les oscillations dans un signal au niveau local. Sur un intervalle séparant deux extrema consécutifs, on sépare le signal en un terme de haute fréquence et un terme de basse fréquence. Dans un premier temps, on élimine celui de basse fréquence (dans tout le signal) pour isoler celui de haute fréquence. On a alors extrait un premier mode. Ensuite, on itère sur le résidu.

On assimile le terme de basse fréquence à la moyenne entre les enveloppes inférieure et supérieure du signal. Ces enveloppes sont calculées par interpolation entre les minima (pour l'enveloppe inférieure) ou les maxima (pour l'enveloppe supérieure) du signal. En pratique, ce n'est pas suffisant : pour isoler le terme de haute fréquence, il faut éliminer la moyenne des enveloppes du signal courant jusqu'à ce que celle-ci devienne nulle. On répète finalement le procédé sur le signal, tel qu'il était avant la phase d'élimination, privé du terme de haute fréquence.

On considère que l'on a extrait tous les modes du signal lorsqu'il compte moins de

trois extrema (≡ moins d'une période).

Soit y=f(x) le signal discrétisé à analyser, x et y sont deux vecteurs de taille n paramétrés par t.

lecture des entrées (x et y) depuis Matlab
a=amplitude de y
tant que y contient au moins une période (ie. au moins trois extrema) **faire**
  ∀ t, z(t)=y(t)
  ∀ t, m(t)=y(t)
  tant que amplitude de m $\geq \varepsilon \times$a **faire**
    recherche des extrema de z
      → $(x_{max},y_{max},x_{min},y_{min})$=*min_max(x,z)*
    interpolation des maximums de z
      → e_{max}=*interpolation*(x_{max},y_{max},x)
    interpolation des minimums de z
      → e_{min}=*interpolation*(x_{min},y_{min},x)
    calcul de la moyenne des enveloppes
      → ∀ *t*, *m(t)*=(e_{max}(t)+e_{min}(t))/2
    mise à jour de z
      → ∀ *t*, *z(t)=z(t)-m(t)*
  fait
  sauvegarde de z dans la liste des modes
  ∀ t, y(t)=y(t)-z(t)
fait
sauvegarde de y dans la liste des modes (en tant que résidu)
écriture des sorties (la matrice contenant les modes) sous Matlab

- *min_max(x,z)* doit retourner les abscisses et les ordonnées des extrema du signal *z=f(x)*
- *interpolation*($x_{tabulés},y_{tabulés},x$) doit retourner les valeurs de la fonction interpolant les points tabulés ($x_{tabulés},y_{tabulés}$) pour les abscisses x

On a vu que pour extraire le terme de haute fréquence d'un signal, il fallait rechercher les extrema d'un signal et interpoler entre ces points. Nous préciserons par la suite comment on a implémenté les fonctions *min_max* et *interpolation*. Nous discuterons également de l'influence et du choix du paramètre ε qui contrôle l'élimination. Nous verrons aussi comment gérer les entrées-sorties avec Matlab.

2 Méthodes utilisées

2.1 Interpolation cubique par des splines

On utilise la méthode d'interpolation par des splines car on a besoin d'avoir des courbes lisses afin de ne pas ajouter des irrégularités dans le signal.

Soit $x = [x_1, x_2, ..., x_n]$ et $y = [y_1, y_2, ..., y_n]$ les abscisses et les ordonnées des points à interpoler. Le principe de cette méthode (expliqué dans *Cubic spline interpolation* [2]) est de calculer une fonction interpolante S :

$$S(x) = \begin{cases} s_1(x) \ si \ x_1 \leq x \leq x_2 \\ s_2(x) \ si \ x_2 \leq x \leq x_3 \\ ... \\ s_{n-1}(x) \ si \ x_{n-1} \leq x \leq x_n \end{cases}$$

où les s_i sont des polynômes de degré 3. Pour que S soit une spline, il faut que S, S' et S" soient continues sur l'intervalle $[x_1,x_n]$. De plus, on a S(x_i)=y_i $\forall$ i.

On obtient que pour x ϵ $[x_i, x_{i+1}[$: $S(x) = Ay_i + By_{i+1} + Cy_i'' + Dy_{i+1}''$ avec :

$$A = \frac{x_{i+1} - x}{x_{i+1} - x_i} \qquad B = 1 - A = \frac{x - x_i}{x_{i+1} - x_i}$$

$$C = \frac{1}{6}(A^3 - A)(x_{i+1} - x_i)^2 \qquad D = \frac{1}{6}(B^3 - B)(x_{i+1} - x_i)^2$$

les y_i'' étant calculés par la résolution du système :

$$\frac{x_j - x_{j-1}}{6}y_{j-1}'' + \frac{x_{j+1} - x_{j-1}}{3}y_j'' + \frac{x_{j+1} - x_j}{6}y_{j+1}'' = \frac{y_{j+1} - y_j}{x_{j+1} - x_j} - \frac{y_j - y_{j-1}}{x_j - x_{j-1}}, \ j \ \epsilon \ \{2, ..., n-1\}$$

(d'après *Numerical recipes in C* [3]). Ce système est tridiagonal, ce qui est pratique car il existe des algorithmes pour résoudre de tels systèmes en O(n).

Cependant, on a n-2 équations pour n inconnues, il faut donc en fixer deux. On a fixé y_1'' et y_n'' à 0 (on a alors ce qu'on appelle des splines *naturelles*).

On a représenté le résultat de l'interpolation des minima, et celle des maxima, d'un signal par cette méthode (voir Fig. 1).

2.2 Choix des valeurs à tabuler pour l'interpolation

La fonction *interpolation* prend en entrée la liste des minima (ou des maxima) d'un signal ainsi que celle des abscisses correspondantes. Comme on l'a vu précedemment, la fonction détermine, entre deux points consécutifs, un polynôme afin de construire la fonction interpolante.

Le problème qui se pose est que le premier et le dernier échantillon du signal ne sont pas, en général, des minima et des maxima. Il faut néanmoins fournir à *interpolation* une valeur en ces points pour déterminer le premier et le dernier polynôme de la fonction interpolante.

Soit y le signal de longueur n dont on doit déterminer les enveloppes. Soit y_max (resp. y_min) le vecteur de longueur n_max (resp. n_min) qui contient l'ordonnée des points à interpoler pour trouver l'enveloppe supérieure (resp. inférieure). y_max contient donc les maxima de y (de y_max_2 à $y\_max_{n\_max-1}$) (idem pour y_min). Il reste à déterminer

y_max_1, $y\_max_{n\_max}$, y_min_1 et $y\_min_{n\_min}$:

Première méthode : Choisir simplement (voir FIG. 2 - première méthode) :
y_max_1=y_max_2, $y\_max_{n\_max}$=$y\_max_{n\_max-1}$,
y_min_1=y_min_2 et $y\_min_{n\_min}$=$y\_min_{n\_min-1}$.

Cette technique provoque une erreur importante aux extrémités du signal, mais cette erreur ne se propage pas sur le signal tout entier (voir FIG. 3 - première méthode).

Deuxième méthode : Considérer la pente du signal y en *1* et en n (voir FIG. 2 - deuxième méthode) . Ainsi en *1*, par exemple :
si le signal est croissant, on choisit y_min_1=y_1 et y_max_1=y_max_2.
si le signal est décroissant, on choisit y_max_1=y_1 et y_min_1=y_min_2.

Cette technique donne un résultat plus précis localement, mais provoque une erreur qui se propage dans tout le signal. Ceci implique l'apparition de termes parasites dans la décomposition (voir FIG. 3 - deuxième méthode).
Remarque : on aurait eu le même problème (en pire) si on avait pris y_min_1=y_1, y_max_1=y_1, $y\_min_{n\_min}$=y_n et $y\_max_{n\_max}$=y_n.

Compromis : Au début du signal y, par exemple, si $y\_min_2 \leq y_1 \leq y\_max_2$, on choisit la première méthode, sinon, on choisit la deuxième. En effet, dans ce cas, on considère que la tendance générale du signal est constante au début du signal. On évite ainsi de perturber l'interpolation. Par contre, dans le cas contraire, on ne peut pas toujours le considérer. On minimise ainsi les deux types de défauts (voir FIG. 3 - compromis).

2.3 Précision des calculs

Lorsque l'on a commencé à implémenter l'algorithme, on a codé nos données en réels simple précision. L'objectif était de rendre les calculs plus rapides et on pensait que la précision serait suffisante.

Cependant, on s'est rendu compte que le programme ne convergeait pas toujours. En effet, dans la boucle interne, on doit rendre l'amplitude crête à crête du vecteur m (moyenne des enveloppes inférieures et supérieures du signal courant) inférieure à une constante ($\varepsilon\times$amplitude(signal d'entrée)). Or, pour que le programme converge, il fallait augmenter cette constante.

Du coup, on a reconsidéré l'hypothèse selon laquelle le codage des données était suffisamment précis. On s'est rendu compte qu'en travaillant avec des réels double précision, on n'avait plus de problèmes de convergence.

Auparavant, la décomposition de certains signaux simples (i.e. composés de quelques milliers de points avec relativement peu d'oscillations) divergeait pour $\varepsilon = 10^{-3}$. Après cette modification, on a refait les mêmes tests avec un $\varepsilon = 10^{-8}$: tous les tests ont été positifs.

Ce problème est probablement dû au fait que, lors de l'interpolation, on inverse un système linéaire. En effet, si la matrice du système est mal conditionnée, une légère imprécision sur ses valeurs peut conduire à des résultats tout à fait différents de ceux attendus.

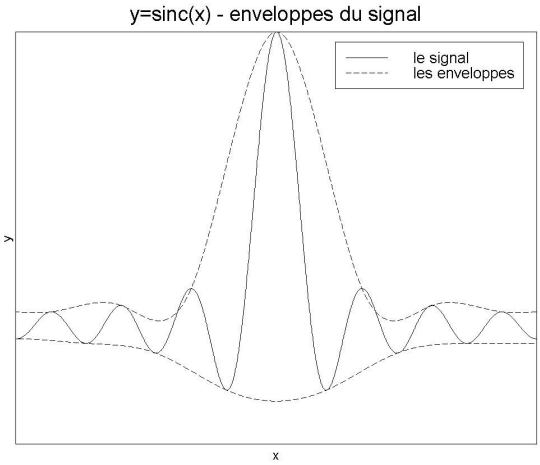

FIG. 1 – Enveloppes supérieures et inférieures d'un signal

On a interpolé entre les minima (resp. maxima) du signal y=sinc(x), ce qui constitue l'enveloppe inférieure (resp. supérieure) de ce signal. L'enveloppe supérieure présente plus d'oscillations que ce qu'on aurait pu espérer.

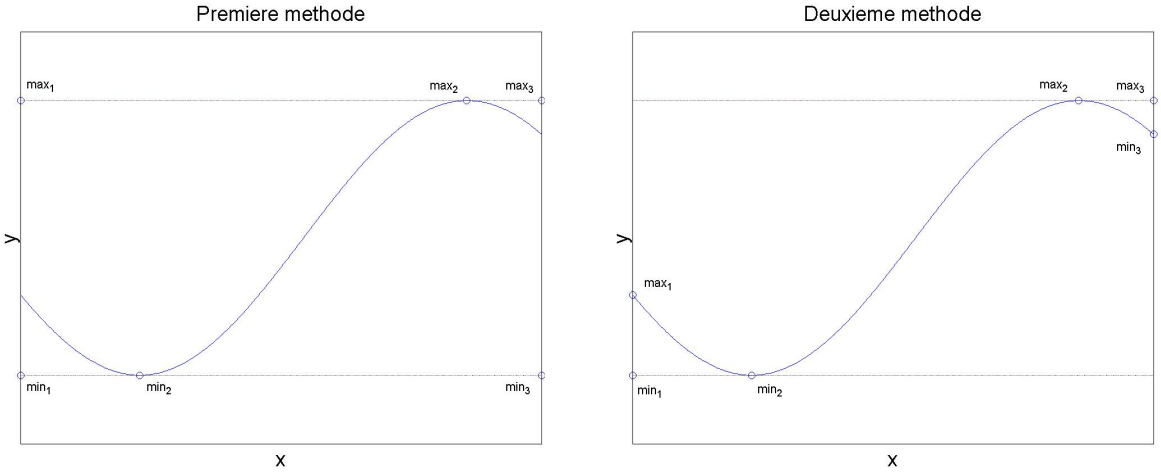

FIG. 2 – Choix des valeurs à tabuler pour l'interpolation

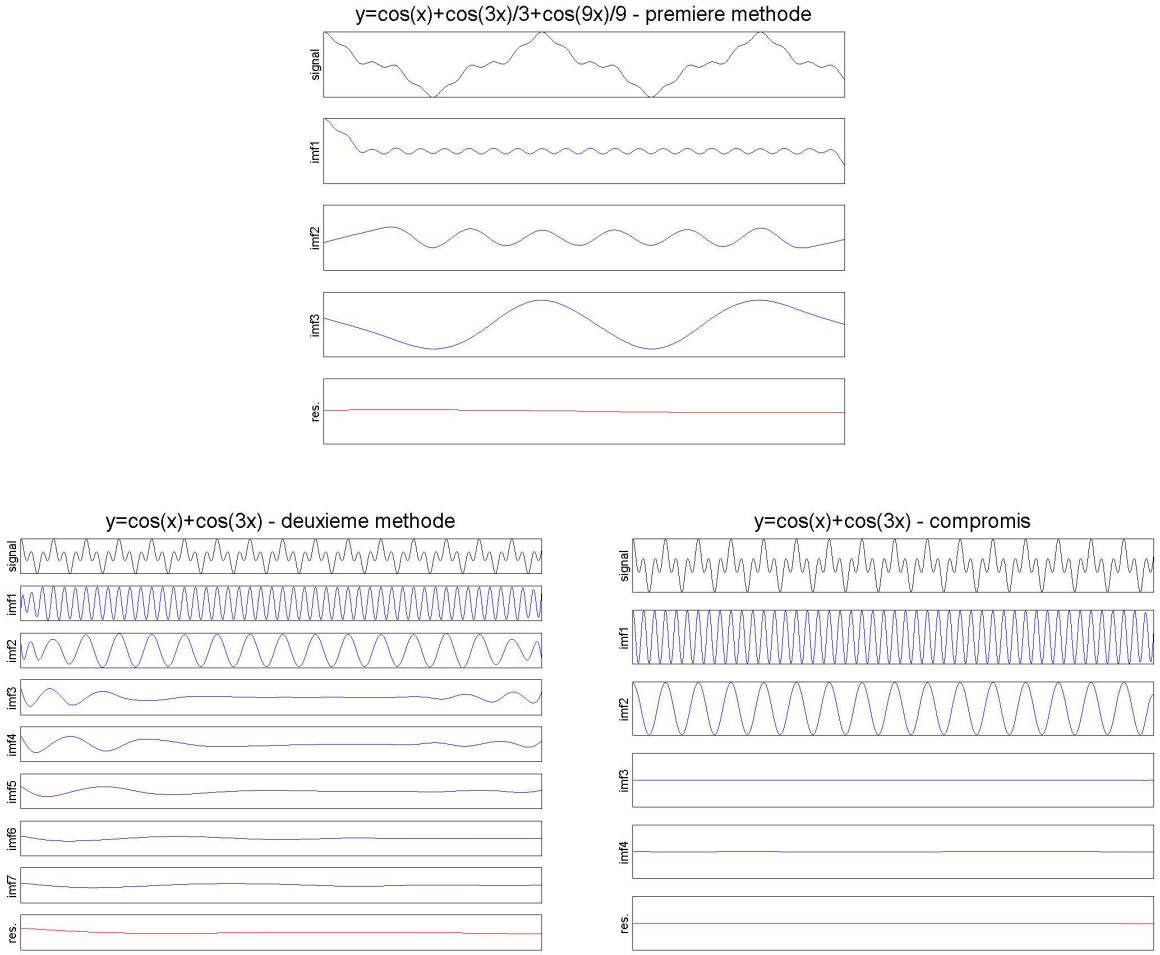

FIG. 3 – Influence des valeurs tabulées pour l'interpolation

Concernant la première façon de choisir les valeurs à tabuler, si le premier point du signal est supérieur au premier maximum ou inférieur au premier minimum, un gros défaut apparaît. Concernant la deuxième, c'est le contraire. Le problème est le même à la fin du signal. En choisissant quelle méthode appliquer en fonction de ce critère, les défauts sont moins importants.

2.4 Interface C/Matlab

Le programme doit lire deux vecteurs (x et y) au début du programme et écrire une matrice (*imfs*, qui contient les modes extraits du signal) à la fin.

Pour cela, la fonction principale doit avoir le prototype suivant :

*void mexFunction(int nlhs,mxArray *plhs[ ],int nrhs,const mxArray *prhs[ ])*

- *nlhs* : nombre d'arguments en sortie
- *plhs* : liste de pointeurs vers les matrices Matlab en sortie
- *nrhs* : nombre d'arguments en entrée
- *prhs* : liste de pointeurs vers les matrices Matlab en entrée

Pour lire les données des matrices en entrée, on utilise les fonctions suivantes :

- *mxGetM(prhs[i])* : lecture du nombre de lignes de la i$^{\text{ème}}$ matrice
- *mxGetN(prhs[i])* : lecture du nombre de colonnes de la i$^{\text{ème}}$ matrice
- *mxGetPr(prhs[i])* : lecture du pointeur vers les données

Pour écrire les données des matrices en sortie, on utilise les fonctions suivantes :

- *plhs[i]=mxCreateDoubleMatrix(m,n,mxREAL)* : allocation d'une zone mémoire pour créer la i$^{\text{ème}}$ matrice Matlab (de taille $m \times n$)
- *mxGetPr(plhs[i])* : récupération depuis le programme d'un pointeur vers cette zone mémoire, pour y écrire les données

Le programme peut alors être compilé depuis l'invite de commande Matlab

$\gg$ *mex emd.c*

et y être appelé comme une fonction Matlab.

Grâce à l'utilisation des splines pour l'interpolation, on espère bien approximer le terme de basse fréquence du signal, que l'on veut éliminer. Grâce à une méthode heuristique de choix des valeurs à tabuler pour l'interpolation, on espère diminuer efficacement les effets de bord. Nous verrons dans quelle mesure l'extraction des termes est précise. En outre, on verra quelles sont les performances du programme en terme de temps de calcul, compte tenu de la précision des calculs.

3 Résultats et discussion

3.1 Analyse d'un exemple

La caractéristique de la méthode EMD est de décomposer un signal en des modes bien adaptés au signal. Ainsi la décomposition d'un signal triangulaire donne un seul terme : lui-même. En effet, c'est trivial : dès la première itération de la boucle interne du programme, les enveloppes supérieures et inférieures du signal sont constantes et opposées, ce qui implique que la moyenne est nulle. Donc on extrait le triangle, ainsi le reste est nul et le programme s'arrête. Pour que çà ne soit plus trivial, on crée une perturbation en ajoutant une sinusoïde.

On voit (Fig. 4, $\omega_{cosinus} = 15\omega_0$) que, si la fréquence de la sinusoïde est suffisamment différente de celle du triangle, la décomposition de ce signal donne à peu près la même sinusoïde et le même triangle. Les modes sont donc biens adaptés au signal.

Cet exemple nous montre également (Fig. 4, $\omega_{cosinus} = 3\omega_0$) que l'on peut extraire le fondamental d'un signal. En effet, la décomposition en séries de Fourier d'un signal triangulaire est de la forme :

$$f(t) = C \times \sum_{k=0}^{\infty} \frac{cos((2k+1)\omega_0 t)}{(2k+1)^2}$$

Comme l'amplitude des harmoniques diminue en $1/n^2$ pour un signal triangulaire, l'EMD ne voit pas les fréquences $\omega_i > \omega_0$ séparément.

Or, en ajoutant suffisamment d'énergie à la première harmonique (une sinusoïde de fréquence $3\omega_0$), on force l'EMD à ne pas voir un seul mode dans le signal, mais deux : un premier mode qui contient la sinusoïde et les harmoniques du triangle, un second qui contient le fondamental du triangle. Il serait donc envisageable d'utiliser l'EMD pour décomposer itérativement un signal en série de Fourier, grâce à un procédé similaire.

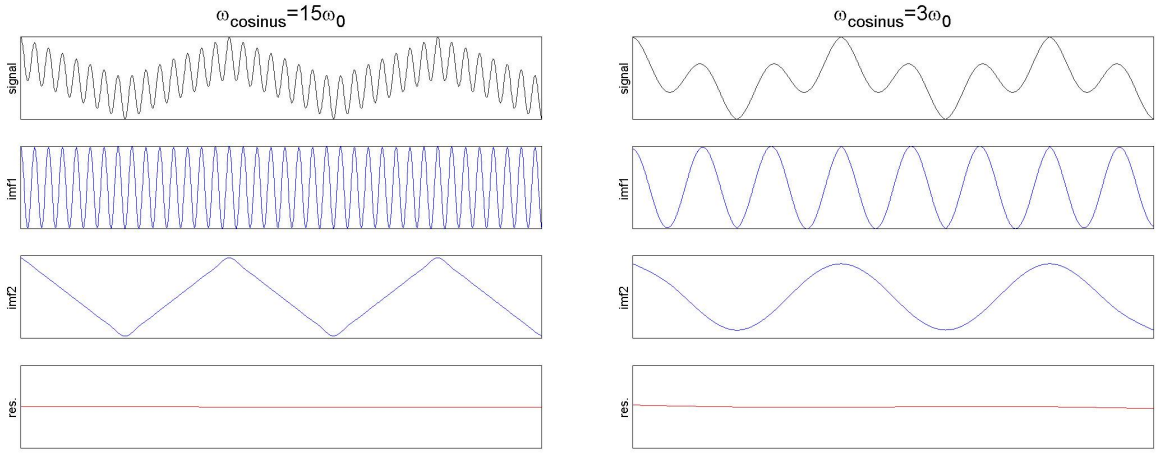

Fig. 4 – Somme d'un signal triangulaire et d'une sinusoïde

La sinusoïde a, dans le premier cas, la fréquence du $7^{ème}$ harmonique du triangle, on récupère alors pratiquement, après EMD, les deux signaux sommés. Dans le deuxième cas, la fréquence de la sinusoïde est celle du premier harmonique du triangle, ce n'est alors plus le cas : le contenu fréquentiel des deux signaux sommés n'est plus séparable.

3.2 Temps de calcul du programme

On a mesuré les temps de calcul du programme, sur plusieurs signaux, en faisant varier la taille de l'instance et le paramètre ε de contrôle de la boucle interne. On a comparé ces temps de calcul avec ceux du programme Matlab de Flandrin. (voir FIG. 5)

De manière générale, on remarque que le rapport entre les temps de calcul du programme Matlab et ceux du programme C augmente avec la taille de l'instance. Par contre, le paramètre ε a apparemment peu d'influence sur cette augmentation.

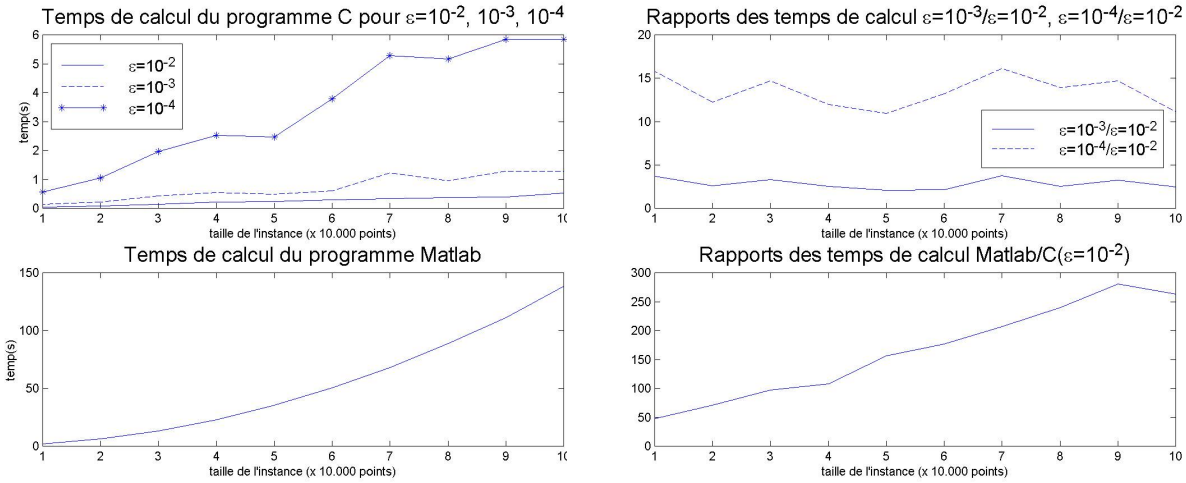

FIG. 5 – Temps de calcul en fonction de la taille de l'instance

Comparaison des durées d'extraction des trois premiers modes du signal "y=cos(x)+cos(3x)+cos(5x)" avec un nombre de points variant de 10.000 à 100.000 points. Mesures effectuées sur un PC avec processeur AMD Athlon 1,47GHz

3.3 Problème non résolu

Lorsque l'on fait la somme de sinusoïdes suffisamment séparées fréquentiellement, on devrait extraire ses différentes sinusoïdes plus un reste. Cependant, quelque soit la précision de l'élimination (i.e. quelque soit la paramètre ε), on peut obtenir des termes supplémentaires (voir FIG. 6). Cependant leur énergie est négligeable devant les autres termes quand ε est suffisamment petit. Ceci met en évidence que lors de la phase d'élimination, on introduit des perturbations. En effet, plus on élimine ($\equiv$ plus ε est petit) plus, en général, le nombre de modes extraits augmente.

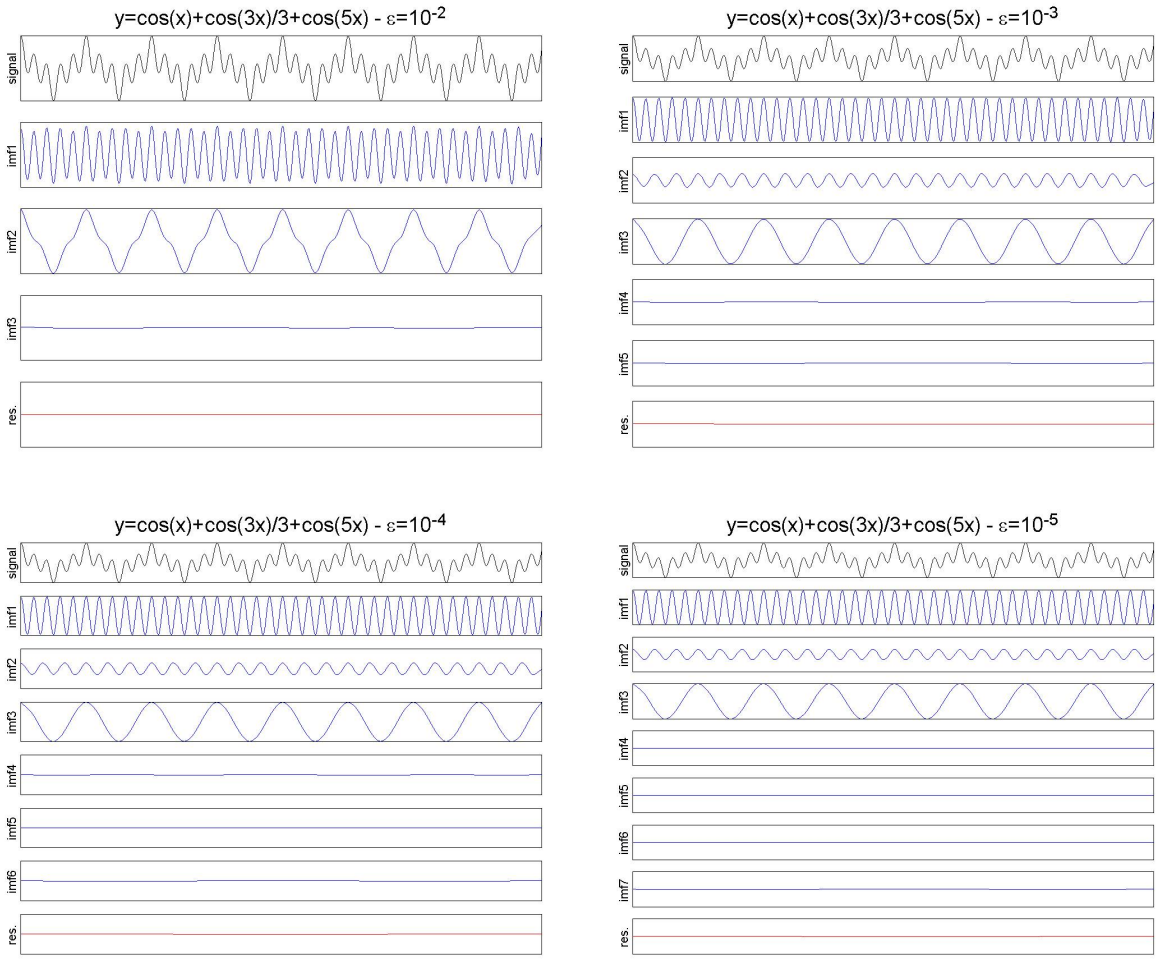

FIG. 6 – Problème des termes parasites

Dans cet exemple (y=cos(x)+cos(3x)/3+cos(5x)), on voit que le nombre de termes supplémentaires augmente quand ε diminue (ce n'est pas toujours le cas). On voit cependant qu'il faut qu'ε soit suffisamment petit pour détecter toutes les oscillations.

Conclusion

Les principaux objectifs fixés ont été atteints : avoir un programme rapide avec une interface Matlab, bien que certaines améliorations soient envisageables. Les problèmes rencontrés ont d'abord été d'ordre communicationnel. Nous avions en effet des difficultés à présenter, à chaque rendez-vous avec notre responsable de projet, le travail effectué. Les consignes et conseils de ce dernier nous ont cependant permis, en partie, de les résoudre. Ils ont ensuite été d'ordre technique : nous avions un programme qui ne convergeait pas. Nous avons résolu le problème en augmentant la précision des calculs. De plus, La méthode EMD est très sensible aux effets de bord (au fait que les signaux étudiés ne sont pas infinis). Grâce à un choix des valeurs à tabuler pour l'interpolation, nous les avons atténués. Les nombreuses explications fournies par notre responsable de projet sur l'analyse des signaux nous ont permis d'augmenter nos connaissances et notre compréhension dans ce domaine.

Le projet laisse des perspectives d'exploitation. En effet, par un procédé qu'il reste à définir, le programme pourrait être utilisé pour décomposer, algorithmiquement, un signal en séries de Fourier.

Références bibliographiques

[1] RILLING Gabriel, FLANDRIN Patrick, GONCALVES Paulo, On empirical mode decomposition and its algorithms, Juin 2003
(http ://perso.ens-lyon.fr/patrick.flandrin/NSIP03.pdf)

[2] MCKINLEY Sky, LEVINE Megan, Cubic spline interpolation
(http ://online.redwoods.cc.ca.us/instruct/darnold/laproj/Fall98/SkyMeg/Proj.PDF)

[3] PRESS William, FLANNERY Brian, TEUKOLSKY Saul, VETTERLING William, Numerical recipes in C, Octobre 1985, pages 113 à 116
(http ://www.library.cornell.edu/nr/bookcpdf/c3-3.pdf)