ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

COMME EXIGENCE PARTIELLE
À L'OBTENTION DE LA
MAÎTRISE EN GÉNIE ÉLECTRIQUE
M.Ing.

PAR
Mathieu PERRON

DÉVELOPPEMENT D'UN OUTIL PORTABLE D'ANALYSE MODALE PAR RÉPONSE TEMPORELLE

MONTRÉAL, LE 5 MAI 2010

PRÉSENTATION DU JURY

CE MÉMOIRE A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE

M. Louis-A. Dessaint, directeur de mémoire
Département de génie électrique à l'École de technologie supérieure

M. Innocent Kamwa, examinateur externe
Institut de recherche d'Hydro-Québec (IREQ)

M. Pierre Jean Lagacé, président du jury
Département de génie électrique à l'École de technologie supérieure

IL A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY ET PUBLIC

LE 30 AVRIL 2010

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

AVANT-PROPOS ET REMERCIEMENTS

Ce document présente mes travaux effectués dans le cadre du programme de maîtrise en génie électrique à l'École de technologie supérieure.

Un travail préliminaire a été réalisé par M. Innocent Kamwa et M. Luc Gérin-Lajoie lors du développement d'un code MATLAB d'analyse modale appliqué spécifiquement à l'étude des modes de résonance électromécanique d'un turbo-alternateur. Les principes fondamentaux utilisés dans ce mémoire proviennent en partie de ce travail important. L'objectif est de développer un code source portable et générique permettant d'effectuer l'analyse modale de n'importe quel signal discrétisé, peu importe sa provenance et selon certaines conditions fixées par l'utilisateur d'un programme hôte intégrant l'outil mis au point.

Ce projet m'a été présenté par le professeur Louis-A. Dessaint durant l'hiver 2007 alors que je terminais mes études de premier cycle et c'est avec enthousiasme que j'ai accepté d'y participer en soulignant mon intérêt marqué pour le domaine du transport de l'énergie.

À priori, ce projet a été commandé par M. Innocent Kamwa de la division Expertise réseaux électriques et mathématiques de l'Institut de Recherche d'Hydro-Québec (IREQ). Le travail en résultant est destiné à l'étude des comportements dynamiques sur des simulations ou des lectures provenant du réseau d'Hydro-Québec et sera utilisé par des ingénieurs de cette entreprise. La dimension pratique du projet et la possibilité de m'impliquer à l'IREQ ont largement contribué à ma motivation tout au long de ce travail et je suis reconnaissant d'avoir obtenu cette chance.

Je reconnais le support financier de la chaire TransÉnergie de l'ÉTS, du professeur Louis-A. Dessaint et de l'organisme MITACS qui m'ont permis de me consacrer entièrement à mes recherches au cours des deux dernières années et de mener ce mémoire à terme.

Je tiens à remercier sincèrement mon directeur de maîtrise, le professeur Louis-A. Dessaint pour sa confiance immuable à mon égard durant toute la durée du projet. D'autre part, je souligne conjointement l'expertise et l'appui de mon codirecteur, M. Innocent Kamwa et de M. Éric Lemieux, sans qui ce projet n'aurait pu être mené à échéance. Leur générosité et leur patience ont été déterminantes à la complétion de ce projet ainsi qu'à ma réussite personnelle. Je remercie également mes collègues de l'IREQ et de l'ÉTS : Simon, Sylvain, Omar, Alain, Gilbert, François, Christian, Jean-François, Josianne et Mélanie pour les judicieux conseils et les bons moments passés ensemble.

De tout cœur, je remercie les membres de ma famille qui m'ont toujours supporté et encouragé au long de mes études et dans les moments plus difficiles. À ma mère et à mon père, Michèle et Martin, un merci spécial pour votre amour, votre soutien inconditionnel à travers mes nombreuses entreprises et pour avoir su me guider vers les bonnes décisions.

Finalement, je garde une pensée particulière pour ma conjointe Anne, envers qui j'exprime toute mon affection et ma gratitude pour sa patience, sa compréhension et ses nombreuses attentions qui m'ont permis de poursuivre mes rêves jusqu'au bout.

DÉVELOPPEMENT D'UN OUTIL PORTABLE D'ANALYSE MODALE PAR RÉPONSE TEMPORELLE

Mathieu PERRON

RÉSUMÉ

L'analyse modale s'impose de plus en plus comme procédé de caractérisation des comportements dynamiques de systèmes. Son application à l'étude des phénomènes basse fréquence sur les réseaux électriques propose un large éventail de solutions, applicables à des contextes particuliers. Toutefois, la taille grandissante des réseaux actuels augmente considérablement la complexité de leur modélisation et du traitement de leurs variables connues. Pour ces raisons, le besoin d'une approche signal à l'analyse modale devient essentiel dans le but de répondre aux contraintes des approches systèmes habituelles.

Ce rapport de recherche traite d'un outil portable d'analyse modale par réponse temporelle. Le produit logiciel développé fournit une solution d'identification par modèle d'état d'ordre réduit, associée à une étude paramétrique des éléments de Prony. La réalisation *Single-Input-Single-Output* (SISO) est basée sur le lien entre le signal temporel du système analysé et la réponse à l'impulsion du modèle d'état équivalent. Un algorithme ERA/Prony est développé et permet d'obtenir les paramètres de chacune des sinusoïdes amorties qui composent le signal, sans nécessiter d'informations sur la provenance de celui-ci.

Le processus d'identification par modèle d'état est entièrement détaillé et propose une réalisation par décomposition singulière d'une matrice de Hankel construite à partir des échantillons du signal. Le modèle est réduit d'après l'étude des valeurs singulières obtenues. Par la suite, les paramètres de Prony sont estimés selon une décomposition par éléments propres et un calcul des éléments résiduels à l'aide des matrices d'état du modèle. Les particularités logicielles sont précisées et trois études de cas démontrent la validité de l'outil. Les cas étudiés incluent la décomposition d'un signal sinusoïdal simple, la caractérisation de phénomènes oscillatoires sur le réseau à quatre machines de Kundur et l'analyse d'une lecture provenant du réseau d'Hydro-Québec, à la suite d'une perte de synchronisation entre les postes LG4 et Boucherville.

Les études s'effectuent grâce à l'importation de l'outil dans le logiciel de traitement de signal ScopeView d'Hydro-Québec. Cette utilisation démontre la portabilité du code source et permet d'étendre la fonctionnalité d'analyse modale à des logiciels tels qu'HYPERSIM et EMTP-RV.

Mots-clés : Analyse modale, stabilité dynamique, *Eigensystem Realization Algorithm* (ERA), analyse de Prony, mode interzone.

DEVELOPMENT OF A PORTABLE SOFTWARE TOOL FOR TIME RESPONSE MODAL ANALYSIS

Mathieu PERRON

ABSTRACT

Modal analysis is increasingly used as a technique to characterize systems' dynamic behaviour. Its application in the study of low frequency phenomena on electrical networks offers a broad range of solutions for specific situations. Nevertheless, the increasing size of present-day networks complicates their modeling and the use of their known variables. For these reasons, the need for a signal-based approach to modal analysis is essential to avoid the constraints of standard system-based approaches.

The present research report deals with a portable software tool developed for time-response modal analysis. This product provides a reduced-order, state-space model identification method combined with Prony parameters estimation. The Single-Input-Single-Output (SISO) realization is based on the link between the time response from the analyzed system and the pulse response of the equivalent state-space model. An ERA/Prony algorithm is developed which estimates the dynamic parameters of superimposed sinusoids from a discrete time signal, without requiring any information from its original system.

The reduced-order, state-space identification method is detailed and provides a singular value decomposition realization from a Hankel block matrix of the signal's samples. The order of the model is determined according to the observed singular value diagonal. Later, the Prony parameters are estimated from eingenvalue decomposition and calculation of the state matrices residues. The software features are specified and three cases of study show the validity of the algorithm. Studied cases include a simple sinusoidal decomposition, the characterization of oscillatory phenomena on Kundur's four machine network, and the analysis of a sampling taken from Hydro-Québec's network following a synchronization loss between LG4 and Boucherville substations.

The studies were carried out by importing the product into Hydro-Québec's signal analysis software ScopeView. This use demonstrates the portability aspect of the product and moreover gives access to a modal analysis tool in software such as HYPERSIM and EMTP-RV.

Keywords: Modal analysis, dynamic stability, *Eigensystem Realization Algorithm* (ERA), Prony analysis, interarea oscillations.

TABLE DES MATIÈRES

Page

INTRODUCTION 16

CHAPITRE 1 REVUE DE LA LITTÉRATURE 18

CHAPITRE 2 IDENTIFICATION DU MODÈLE D'ÉTAT 19
2.1 Introduction 19
2.2 Hypothèses 19
2.2.1 L'approche signal 19
2.2.2 Équivalence du modèle boîte-noire 20
2.3 Méthode de réalisation ERA 21
2.3.1 Équations du système 21
2.3.2 Approche de la réalisation ERA 22
2.3.3 Matrice de Hankel 26
2.3.4 Décomposition SVD 28
2.3.5 Détermination du modèle boîte-noire 30
2.4 Conclusion 31

CHAPITRE 3 ESTIMATION DES PARAMÈTRES 33
3.1 Introduction 33
3.2 Hypothèses 33
3.3 Approche de Prony 34
3.3.1 Équations du système 35
3.3.2 Éléments propres et résidus 36
3.3.3 Obtention des paramètres 39
3.4 Conclusion 41

CHAPITRE 4 ALGORITHME ERA / PRONY 43
4.1 Introduction 43
4.2 Architecture logicielle 43
4.2.1 Algorithme ERA/Prony 44
4.2.2 LAPACK© 47
4.2.3 NR© 49
4.3 Fonctionnement global 49
4.3.1 Arguments d'entrée 51
4.3.2 Arguments de retour 54
4.4 Notes sur la programmation 58
4.4.1 Convention du Fortran90 dans un environnement C 58
4.4.2 Gestion de la mémoire physique 60
4.4.3 Portabilité 61
4.5 Conclusion 61

CHAPITRE 5 VALIDATION DE L'ALGORITHME 62
5.1 Introduction 62
5.2 Contexte ScopeView© 62
5.3 Étude de cas 1 : Présentation théorique 63
5.3.1 Description de l'étude 64
5.3.2 Résultats 66
5.3.3 Discussion 68
5.4 Étude de cas 2 : Réseau à quatre machines de Kundur 68
5.4.1 Réseau 69
5.4.2 Description de l'étude 69
5.4.3 Résultats 71
5.4.4 Discussion 78
5.5 Étude de cas 3 : Perte de synchronisation LG4-Boucherville 79
5.5.1 SMDA 79
5.5.2 Description de l'étude 80
5.5.3 Résultats 81
5.5.4 Discussion 83
5.6 Conclusion 83

CONCLUSION 84

ANNEXE I Déclaration des fonctions du fichier AMOD.H (MAIN) 87

ANNEXE II Définition des fonctions du fichier AMOD.C (MAIN) 89

ANNEXE III Déclaration des fonctions du fichier ERA.H 91

ANNEXE IV Définition des fonctions du fichier ERA.C 94

ANNEXE V Déclaration des fonctions du fichier APRON.H 104

ANNEXE VI Définition des fonctions du fichier APRON.C 108

ANNEXE VII Déclaration des fonctions du fichier DATA.H 121

ANNEXE VIII Définition des fonctions du fichier DATA.C 123

ANNEXE IX Déclaration des fonctions du fichier YMODEL.H 126

ANNEXE X Définition des fonctions du fichier YMODEL.C 127

ANNEXE XI Déclaration des fonctions du fichier UTIL.H 129

ANNEXE XII Définition des fonctions du fichier UTIL.C 131

ANNEXE XIII Écoulement de puissance de l'étude 2 132

ANNEXE XIV Données du SMDA de l'étude de cas 3 135

BIBLIOGRAPHIE 136

LISTE DES TABLEAUX

Page

Tableau 5.1 Paramètres de Prony recherchés de l'étude 1 64

Tableau 5.2 Paramètres de Prony de l'analyse modale de l'étude 1.....66

Tableau 5.3 Paramètres de Prony de la simulation TGR zone 1 71

Tableau 5.4 Paramètres de Prony de la simulation TGR zone 2 72

Tableau 5.5 Paramètres de Prony de la simulation PSS zone 1.....75

Tableau 5.6 Paramètres de Prony de la simulation PSS zone 2.....75

Tableau 5.7 Paramètres de Prony de l'étude du SMDA 81

LISTE DES FIGURES

Page

Figure 2.1 Représentation synoptique d'un système décrit par équations d'état. 21

Figure 3.1 Représentation par forme d'onde de l'approche de Prony.....34

Figure 3.2 Représentation synoptique d'une fonction de transfert. 35

Figure 4.1 Architecture logicielle.....44

Figure 4.2 Format *row-major order.* 60

Figure 4.3 Format *column-major order.* 60

Figure 5.1 Formes d'onde des modes recherchés de l'étude 1.....65

Figure 5.2 Fenêtre d'appel ScopeView© de l'étude 1. 66

Figure 5.3 Signaux et FFT de l'étude 1.....67

Figure 5.4 Réseau à quatre machines de Kundur. 69

Figure 5.5 Fenêtre d'appel ScopeView© de la simulation TGR zone 1. 70

Figure 5.6 Fenêtre d'appel ScopeView© de la simulation TGR zone 2. 70

Figure 5.7 Fenêtre d'appel ScopeView© de la simulation PSS zone 1. 71

Figure 5.8 Fenêtre d'appel ScopeView© de la simulation PSS zone 2. 71

Figure 5.9 Signaux et FFT de la simulation TGR zone 1.....73

Figure 5.10 Signaux et FFT de la simulation TGR zone 2.....74

Figure 5.11 Signaux et FFT de la simulation PSS zone 1.....76

Figure 5.12 Signaux et FFT de la simulation PSS zone 2.....77

Figure 5.13 Configuration du SMDA d'Hydro-Québec. 80

Figure 5.14 Fenêtre d'appel ScopeView© de l'étude du SMDA.....81

Figure 5.15 Signaux et FFT de l'étude du SMDA. 82

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

AESOPS	Analysis of Essentially Spontaneous Oscillations in Power Systems
ATLAS	Automatically Tuned Linear Algebra Software
BLAS	Basic Linear Algebra Subprograms
C	Langage de programmation C
CC	Courant Continu
C ANSI	C language American National Standards Institute
cLAPACK	C language Linear Algebra PACKage
ERA	Eigensystem Realization Algorithm
EMTP-RV	Electromagnetic Transient Program – Revised Version
EQM	Erreur Quadratique Moyenne
ÉTS	École de technologie supérieure
FFT	Fast Fourier Transform
Fortran90	Formula transformer langage – 1992 ANSI Standard
F2C	Convertisseur Fortran à C
GCC	Gnu C Compiler
GPS	Global Positioning System
IREQ	Institut de recherche d'Hydro-Québec
LAPACK	Linear Algebra PACKage
LG4	Centrale La Grande – 4
LogPack	Logarithm Package
LU	Lower and Upper triangular matrix factorization

MAIN	Fonction principale
MIMO	Multi-Input-Multi-Output
NR	Numerical Recipes
PMU	Power Management Unit
PSS	Power System Stabilizer
PURIFY	Rational PURIFY dynamic software analysis tool
QR	Algorithme d'analyse modale par décomposition matricielle
SCADA	Supervisory Control And Data Acquisition
SMA	Selective Modal Analysis
SISO	Single-Input-Single-Output
SMDA	Système de Mesure du Décalage Angulaire
SVD	Singular Value Decomposition
TGR	Thyristor exciter with a transient Gain Reduction

LISTE DES SYMBOLES ET UNITÉS DE MESURE

a_k	Amplitude du mode k
δ	Delta de Dirac
e_f	Tension de champ
f_k	Fréquence du mode k, Hertz
$H(t)$	Matrice des réponses à l'impulsion du domaine continu
$H(k)$	Matrice des réponses à l'impulsion du domaine discret
$H_{\alpha\beta}(k-1)$	Matrice de Hankel composée des paramètres de Markov
$\widehat{H}_{\alpha\beta}(k-1)$	Matrice de Hankel composée d'échantillons numérique
$\widehat{H}_{\alpha\beta}(k)$	Matrice de Hankel associée aux matrices d'état
$\widehat{H}_n(k)$	Matrice de Hankel d'ordre réduit
Hz	Unité – Hertz
km	Unité - kilomètre
kV	Unité - kilovolt
MVA	Unité - mégavoltampère
MW	Unité - mégawatt
p.u.	Par unité
s	Unité - seconde
σ_k	Amortissement du mode k
τ_{eg}	Torque électrodynamique, p.u.

θ_k	Angle de déphasage du mode *k*, radian
ω_k	Fréquence du mode *k*, radian par seconde
ζ_k	Amortissement relatif du mode *k*

INTRODUCTION

L'analyse modale est un procédé de traitement du signal qui appartient à la théorie de l'identification des systèmes. Ce procédé permet de mesurer les comportements dynamiques d'un système soumis à une ou plusieurs sources de vibrations. Les comportements, appelés modes, s'identifient par des propriétés d'oscillation et représentent l'effet des sources de vibration du système. Les applications de ce type d'identification sont universelles et se retrouvent dans les domaines de l'électricité, de l'aérospatiale, de l'ingénierie civile, de la santé, etc.

Le contexte du présent travail de recherche concerne plus précisément les phénomènes oscillatoires sur les réseaux électriques, en plus d'applications mathématiques générales. Il existe principalement deux approches de réalisation d'une analyse modale, soit le calcul direct des valeurs propres et la méthode par réponse temporelle.

La première approche par calcul direct des valeurs propres contient des techniques par modélisation complète et sélective du système. Les techniques par modélisation complète sont basées sur l'efficacité reconnue de l'algorithme de diagonalisation QR. Cependant, elles sont inappropriées aux contraintes de calcul des réseaux électriques qui peuvent impliquer jusqu'à 30 000 variables. Quant à elles, les techniques par modélisation sélective réduisent l'analyse du système en plusieurs modèles simplifiés, mais sont susceptibles aux erreurs de conditions initiales et à l'élimination de modes entre les zones modélisées.

La deuxième approche par réponse temporelle permet d'éviter plusieurs de ces inconvénients. Le principal avantage de cette approche est de pouvoir réaliser l'analyse modale d'un signal sans connaître les particularités du système d'origine, ce qui en fait d'ailleurs la méthode prescrite dans le développement de cette recherche. La méthode proposée consiste à associer les concepts de la théorie du contrôle et ceux de l'analyse paramétrique. Son but est de créer un modèle d'état d'ordre minimal dont la réponse à l'impulsion est identique au signal temporel du système d'origine, pour ensuite y appliquer

une technique d'estimation des paramètres modaux par valeurs propres. Ainsi, la méthode permet de mesurer les comportements dynamiques et d'analyser leurs causes et effets possibles sur le système.

L'objectif du travail est de réaliser un produit logiciel intégrant cette méthode d'analyse modale et de valider son utilisation avec l'aide d'études de cas sur les signaux numériques. Le produit logiciel doit être en mesure d'obtenir les paramètres d'amplitude, de fréquence, de déphasage et d'amortissement de chacun des modes observés. Aussi, il doit fournir une reconstitution temporelle du signal analysé à partir de la somme des éléments dynamiques identifiés. Il doit également inclure une fonctionnalité de sélection de la fenêtre d'étude fréquentielle et s'appliquer à des phénomènes de l'ordre de 0,05 à 60 Hz. Du point de vue informatique, il doit être portable et affranchi de toute limitation logicielle ou légale. Un rapport de recherche doit présenter la documentation relative à tous les aspects du produit final.

Le rapport de recherche est divisé en cinq chapitres. Dans le premier chapitre, une brève revue de la littérature permet de situer le travail parmi les connaissances scientifiques actuelles sur le domaine en question. Le second chapitre présente la méthode de réalisation d'un système d'état basé sur la technique *Eigensystem Realization Algorithm* (ERA) et son adaptation au contexte du travail. Dans le troisième chapitre, les notions de paramètres de Prony et d'approche résiduelle par valeurs propres complètent la description méthodologique de l'analyse modale privilégiée. Le quatrième explique l'architecture et le fonctionnement du produit logiciel mis au point avec l'algorithme ERA/Prony. Finalement, le cinquième énonce trois études de cas, de complexité graduelle, analysées par l'entremise d'un logiciel de traitement de signal intégrant le produit développé.

En guise de conclusion, le travail résume l'ensemble des thèmes abordés et la validité de la solution d'analyse modale proposée. Quelques recommandations sont énumérées afin de suggérer d'éventuelles pistes de recherche ou certaines améliorations logicielles particulières.

CHAPITRE 1

REVUE DE LA LITTÉRATURE

La littérature disponible sur l'analyse modale est très abondante étant donné son application générale aux sciences physiques. L'ensemble des écrits regroupe principalement deux types de documents. Certains auteurs s'intéressent davantage aux procédures mathématiques relatives à la linéarisation des systèmes et à la stabilité dynamique. D'autres utilisent plutôt ces procédures pour développer des techniques limitées à des applications spécifiques. Parmi tous ces documents, les plus intéressants dans le contexte de cette recherche sont ceux qui traitent d'une approche par modèle d'état détaillée, mais réalisable par un algorithme informatique et applicable à un signal discrétisé.

De manière plus générale, Rao *et al.* (1992, p. 283) exposent l'ensemble des solutions linéaires par modèle d'état aux problèmes d'estimation de systèmes non linéaires. Ils proposent plusieurs pistes de solutions sur l'identification des sinusoïdes amorties et abordent les méthodes robustes pour contrer les erreurs de précision finie. Suite à ces travaux, Juang (1994, p. 121) propose une méthode condensée ERA/Prony avec matrice de Hankel telle que développée dans cette recherche. Lardies (1999, p. 543-558) traite d'une méthode ERA/Prony qui inclut un procédé d'estimation de l'ordre minimal du système basé sur la covariance d'une matrice d'observabilité augmentée.

Pour les applications spécifiques aux réseaux, Kundur *et al.* (1994, p. 799) citent plusieurs techniques d'étude sélectives des valeurs propres associées à des éléments particuliers. C'est le cas de l'algorithme AESOPS utilisé pour calculer les modes d'oscillation au rotor d'une génératrice et de l'analyse modale sélective (SMA) qui permet d'associer des modes aux variables d'état d'une zone isolée. Du point de vue du contrôle, Kamwa *et al.* (2000, p. 326-334) perfectionnent la méthode ERA/Prony pour identifier de manière optimale les points de mesure et de contrôle sur les modes observés dans un contexte de système *Multi-Input-Multi-Output* (MIMO).

CHAPITRE 2

IDENTIFICATION DU MODÈLE D'ÉTAT

2.1 Introduction

La première partie du projet consiste à réaliser un modèle d'état continu d'ordre minimal à partir d'un signal temporel échantillonné $y_{ech}(k)$. Pour arriver à ce système boîte-noire, une méthode nommée *Eigensystem Realization Algorithm* (ERA) adaptée de Juang et Pappa (1985, p. 620-627) est employée dans le but de donner un aspect linéaire au problème d'estimation des paramètres dynamiques. De plus, du point de vue du système, la limitation au signal de sortie comme seul élément d'information demande certaines hypothèses pour appuyer cette approche.

Dans le présent chapitre, on s'attarde d'abord à ces hypothèses pour ensuite exposer la méthode de réalisation ERA dans ses principales étapes. La première étape établit la relation entre la matrice des réponses à l'impulsion d'un modèle d'état et les paramètres de Markov pour introduire l'utilisation de la matrice de Hankel. L'étape suivante démontre comment extraire un système d'état discret de la matrice de Hankel grâce à une décomposition par valeurs singulières. Finalement, un passage classique du domaine discret à continu permet d'identifier les matrices d'état du modèle recherché.

2.2 Hypothèses

2.2.1 L'approche signal

La première hypothèse généralise le signal de sortie en le traduisant comme la réponse d'un système à une excitation par bruit blanc dont le spectre fréquentiel est défini par $\left\|H(j\omega)\right\|_{-\infty\le\omega\le\infty} = 1$. Dans le domaine temporel, cette affirmation signifie de considérer le

signal $y_{ech}(k)$ comme une sortie $y(t)$ d'un système dont la valeur d'entrée $u(t)$ est une impulsion d'amplitude infinie de la forme :

$$\delta(t) = \lim_{\Delta \to 0} \frac{u(t) - u(t-\Delta)}{\Delta} \tag{2.1}$$

$$u(t) = \begin{cases} \delta, & t = 0 \\ 0, & t \geq 1 \end{cases} \tag{2.2}$$

La démarche proposée ne tient pas compte des sources d'entrée connues, et ne cherche pas à les relier au comportement de $y(t)$, mais seulement à définir un modèle qui reproduit les valeurs du signal échantillonné lorsqu'excité par une impulsion δ.

2.2.2 Équivalence du modèle boîte-noire

Une seconde hypothèse présume que plusieurs modèles d'état peuvent décrire le comportement d'un même système. Cette affirmation est basée sur la transformation de similarité. Une modification du vecteur d'état selon $\bar{x} = M^{-1}x$ produit un système d'état équivalent d'après :

$$\left\{ \begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) + Du(t) \end{aligned} \right\} \Leftrightarrow \left\{ \begin{aligned} \dot{\bar{x}}(t) &= (M^{-1}AM)\bar{x}(t) + (M^{-1}B)u(t) \\ y(t) &= (MC)\bar{x}(t) + Du(t) \end{aligned} \right\} \tag{2.3}$$

où $M(n \times n)$ est régulière et inversible pour des valeurs propres distinctes de A. Cette transformation confirme que la relation entre $u(t)$ et $y(t)$ demeure inchangée pour une altération interne du système sous certaines conditions. L'hypothèse justifie donc la recherche d'un modèle d'ordre minimal à partir des conditions mentionnées.

2.3 Méthode de réalisation ERA

2.3.1 Équations du système

Bien que les comportements dynamiques d'un système soient de nature non linéaire, Kamwa (1993, p. 1.1) stipule que « Un système est dynamique si son évolution se traduit par des formes variant avec le temps, de telle manière que les propriétés prévalant à un instant particulier soient reliées avec celles observées aux instants antérieurs. » Ainsi, la connaissance de ces propriétés se traduit par des équations d'état qui dictent les liens dynamiques entre l'entrée(s) $u(t)$, l'état(s) $x(t)$ et la sortie(s) $y(t)$ et permettent de rendre le système linéaire selon :

$$\boxed{\begin{aligned}\dot{x}(t) &= Ax(t) + Bu(t)\\ y(t) &= Cx(t) + Du(t)\end{aligned}} \tag{2.4}$$

avec : $u(t)$ = vecteur d'entrée de dimension p
$y(t)$ = vecteur de sortie de dimension r
$x(t_0) = x_0$
$A(n \times n)$, $B(n \times p)$, $C(r \times n)$, $D(r \times p)$

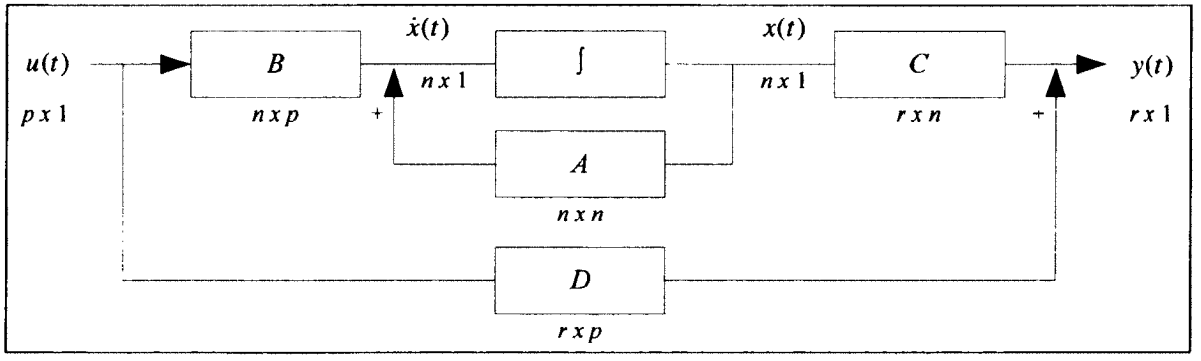


Figure 2.1 Représentation synoptique d'un système décrit par équations d'état.
Tirée de Bensoussan (2008, p. 115)

Le système recherché est de configuration *Single-Input-Single-Output* (SISO) avec une entrée $u(t)$ considérée comme connue. Bensoussan (2008, p. 115) dit que « le terme $Du(t)$ ne

fait que s'ajouter à la sortie du système et n'influence pas la dynamique du système », ce qui rend son utilisation obsolète. En réduisant pour un ordre minimal, l'équation est ramenée à une expression scalaire et équivalente de la forme:

$$\boxed{\begin{aligned}\dot{x}(t) &= \hat{A}x(t) + \hat{B}u(t)\\ y(t) &= \hat{C}x(t)\end{aligned}} \qquad (2.5)$$

avec : $\hat{A}(n_{\min}\ x\ n_{\min})$

$\hat{B}(n_{\min}\ x\ 1)$

$\hat{C}(1\ x\ n_{\min})$

L'approche de la méthode ERA et le contexte de signal échantillonné nécessitent d'inclure l'équation sous sa forme discrétisée :

$$\boxed{\begin{aligned}x(k+1) &= Fx(k) + Gu(k)\\ y(k) &= Px(k)\end{aligned}} \qquad (2.6)$$

avec : $F(n_{\min}\ x\ n_{\min})$

$G(n_{\min}\ x\ 1)$

$P(1\ x\ n_{\min})$

$k = 0,\ 1,\ 2,\ \ldots$

2.3.2 Approche de la réalisation ERA

La réalisation ERA abordée dans ce chapitre est conditionnée par son rôle mathématique dans l'intégralité de l'analyse. Ainsi, pour assurer la robustesse des paramètres estimés, l'approche ERA utilise des matrices d'état qui doivent être prédisposées aux techniques de décomposition qui leur sont appliqués. Dans la littérature, Rao décrit la forme matricielle de ces deux étapes :

> « Dans l'approche de modélisation par espace d'état au problème d'extraction de sinusoïdes, les valeurs propres de F sont les principales quantités que l'on désire connaître. Donc, la tâche d'identifier une série robuste de paramètres

> dynamique est réduite à trouver une réalisation permettant la décomposition par valeurs propres de la matrice de retour d'état (i.e. ses valeurs propres sont insensibles aux perturbations de l'entrée). Par algèbre matricielle, il est connu qu'une forme normale (i.e. une matrice qui commute avec sa transposée complexe ou Hermitienne, $FF^H = F^H F$) constitue une telle matrice. » (Rao *et al.* 1992, p. 286)

En réponse à cette demande particulière, la méthode ERA fait appel à une étape de décomposition par valeurs singulières pour générer la matrice d'état du domaine discret *F*. Cette technique remplit les conditions de similarité de l'hypothèse d'équivalence du modèle en 2.2.2.

$$M^{-1}AM = \begin{pmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \lambda_2 & \\ & & \ddots \\ 0 & & \lambda_n \end{pmatrix} \qquad \text{pour} \quad A = \frac{\log(F)}{\Delta t} \tag{2.7}$$

La démonstration de la méthode débute avec la relation entre les valeurs du signal échantillonné $y_{ech}(k)$ et les paramètres de Markov d'un système. Pour se faire, l'équation d'état de *y(t)* (2.4) est d'abord reformulée à l'aide de la matrice de transition d'état. De cette équation, la matrice des réponses à l'impulsion est extraite de l'équation à l'aide du théorème de convolution pour des conditions initiales nulles. Cette matrice est alors exprimée en fonction des matrices du système d'état afin d'être comparée aux paramètres de Markov obtenus par la description séquentielle des sorties *y(k)* du modèle d'état.

Matrice de réponses à l'impulsion

La matrice de transition d'état permet de prédire l'évolution d'un état d'après sa valeur au temps t_0 en fonction de la matrice *A* du système :

$$x(t) = \Phi(t, t_0) x(t_0) \tag{2.8}$$

en fonction de

$$\Phi(t, t_0) = e^{A(t-t_0)} \tag{2.9}$$

La matrice de transition d'état peut ainsi être remplacée dans l'équation de sortie du système d'état comme le démontre Kamwa (1993, p. 24) pour obtenir « la réponse du système vue des variables mesurables » qui s'exprime selon :

$$y(t) = C\ \Phi(t, t_0)\ x_0 + C \int_{t_0}^{t} \Phi(t, t_0)\ Bu(\tau)\, d\tau \tag{2.10}$$

Si l'on suppose des conditions initiales nulles pour $x_0 = 0$ avec un temps initial $t_0 = 0$, l'équation est simplifiée, faisant apparaître l'intégrale de convolution :

$$y(t) = \int_{0}^{t} H(t-\tau) u(\tau)\, d\tau \tag{2.11}$$

ce qui implique une matrice des réponses à l'impulsion de la forme :

$$\boxed{H(t) = C\Phi(t)B} \tag{2.12}$$

discrétisée par :

$$\boxed{H(k) = PF^k G} \tag{2.13}$$

Cette matrice est directement liée à la recherche des paramètres $\hat{A}, \hat{B}, \hat{C}$ selon l'hypothèse que le signal $y_{ech}(k)$ est considéré comme la sortie d'un système excité par une impulsion.

Signal et paramètres de Markov

Les paramètres de Markov sont extraits d'une série de forme séquentielle (*weighting sequence description*) proposée par Juang (1994, p. 23) « pour une même condition initiale $x_0 = 0$ selon les entrées $u(i)$ $(i = 0,\ 1,\ 2,\ \ldots, k)$ » :

$$\left.\begin{cases} x(0) = 0, \\ y(0) = Du(0), \\ x(1) = Gu(0), \\ y(1) = PGu(0) + Du(1), \\ x(2) = FGu(0) + Gu(1), \\ y(2) = PFGu(0) + PGu(0) + Du(2), \\ \quad \vdots \end{cases}\right\} \Leftrightarrow \left\{\begin{array}{l} x(k) = \sum_{i=1}^{k} F^{i-1}Gu(k-i), \\ y(k) = \sum_{i=1}^{k} PF^{i-1}Gu(k-i) + Du(k) \end{array}\right\} \quad (2.14)$$

Il explique que « si l'on substitue les variables d'entrée par $u(0) = 1$ et $u(i) = 0$ $(i = 1,\ 2,\ \ldots, k)$, les résultats de $y(k)$ peuvent être assemblés selon une séquence de réponses à l'impulsion de la forme Y_k, dénommées paramètres de Markov » :

$$Y_0 = D, \quad Y_1 = PG, \quad Y_2 = PFG, \quad Y_k = PF^{k-1}G \quad (2.15)$$

$$\boxed{y(k) = \sum_{i=0}^{k} Y_i u(k-i)} \quad (2.16)$$

En appliquant l'identité de réciprocité à la séquence de Markov, l'intégrale de convolution est retrouvée une fois de plus. Une comparaison terme à terme entre $H(k)$ (2.13) et Y_k (2.15) permet de valider le lien entre les paramètres de Markov d'un système donné et l'échantillonnage $y_{ech}(k)$.

2.3.3 Matrice de Hankel

La matrice de Hankel dans sa forme générale est normalement construite des paramètres de Markov du signal, comme le démontre Ho et Kalman (1965, p. 449-459) dans les premiers travaux sur la réalisation minimale par espace d'état :

$$H_{\alpha\beta}(k-1)=\begin{bmatrix} Y_k & Y_{k+1} & \cdots & Y_{k+\beta-1} \\ Y_{k+1} & Y_{k+2} & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ Y_{k+\alpha-1} & Y_{k+\alpha} & \cdots & Y_{k+\alpha+\beta-2} \end{bmatrix} \tag{2.17}$$

La matrice de Hankel adaptée pour ce travail est issue de l'unification du signal étudié et des paramètres de Markov démontrée précédemment, ce qui permet d'adapter $H_{\alpha\beta}(k-1)$ (2.17) selon :

$$\widehat{H}_{\alpha\beta}(k-1)=\begin{bmatrix} y_{ech}(k) & y_{ech}(k+1) & \cdots & y_{ech}(k+\beta-1) \\ y_{ech}(k+1) & y_{ech}(k+2) & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ y_{ech}(k+\alpha-1) & y_{ech}(k+\alpha) & \cdots & y_{ech}(k+\alpha+\beta-2) \end{bmatrix} \tag{2.18}$$

L'unification de ces concepts est d'autant plus importante qu'elle met en relation les informations de sortie échantillonnées $y_{ech}(k)$ à la recherche des matrices F, G, P suite à une substitution de la séquence Y_k (2.15) dans $\widehat{H}_{\alpha\beta}(k-1)$ (2.18) pour :

$$\widehat{H}_{\alpha\beta}(k-1)\equiv\begin{bmatrix} PF^{k-1}G & PF^{k}G & \cdots & PF^{k+\beta-2}G \\ PF^{k}G & PF^{k+1}G & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ PF^{k+\alpha-2}G & PF^{k+\alpha-1}G & \cdots & PF^{k+\alpha+\beta-3}G \end{bmatrix} \tag{2.19}$$

Une matrice de Hankel, réalisée à partir du premier échantillon $k=1$, est factorisée de manière à obtenir les matrices d'observabilité et de gouvernabilité d'après la mise en évidence suivante :

$$\widehat{H}_{\alpha\beta}(0) \equiv \begin{bmatrix} PG & PFG & \cdots & PF^{\beta-1}G \\ PFG & PF^2G & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ PF^{\alpha-1}G & PF^{\alpha}G & \cdots & PF^{\alpha+\beta-2}G \end{bmatrix} \tag{2.20}$$

$$\widehat{H}_{ij}(0) = \begin{bmatrix} P \\ PF \\ \vdots \\ PF^{\alpha-1} \end{bmatrix} \cdot \begin{bmatrix} G & FG & \cdots & F^{\beta-2}G \end{bmatrix} = \mathbb{Q}_{\alpha}\mathbb{C}_{\beta} \tag{2.21}$$

De même, une matrice de Hankel réalisée à partir de l'échantillon $k=2$ fait apparaître le terme F par une factorisation semblable :

$$\widehat{H}_{\alpha\beta}(1) = \begin{bmatrix} PFG & PF^2G & \cdots & PF^{\beta}G \\ PF^2G & PF^3G & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ PF^{\alpha}G & PF^{\alpha+1}G & \cdots & PF^{\alpha+\beta-1}G \end{bmatrix} = \mathbb{Q}_{\alpha}F\mathbb{C}_{\beta} \tag{2.22}$$

de manière générale :

$$\boxed{\widehat{H}_{\alpha\beta}(k) = \mathbb{Q}_{\alpha}F^{k}\mathbb{C}_{\beta}} \tag{2.23}$$

Bien que ces factorisations démontrent théoriquement que des informations cruciales sur le système sont contenues dans la forme de Hankel, elles ne sont pas applicables du point de vue pratique sur les valeurs de $y_{ech}(k)$. Il est donc nécessaire de proposer un outil de décomposition qui unit les concepts théoriques aux données pratiques.

2.3.4 Décomposition SVD

La décomposition par valeurs singulières *Singular Value Decomposition* (SVD) est la technique priorisée pour répondre aux besoins de factorisation d'une matrice de Hankel $\widehat{H}_{\alpha\beta}$ composée d'échantillons et préparer ultérieurement le terme F à une décomposition par valeurs propres. Cette technique possède aussi des avantages de discernement sur le niveau de l'ordre minimal à établir pour la matrice, tel que le précise Rao :

> « Il est démontré par Wilkinson (1965) que les valeurs et vecteurs singuliers d'une matrice sont relativement insensibles aux perturbations dans les entrées de la matrice et aux erreurs de précision finie. De ce fait, la distribution des valeurs singulières est une bonne indication du rang numérique d'une matrice. Les valeurs numériques qui auraient dû égaler zéro seront suffisamment petites en présence d'erreurs de précision finie dans leur calcul et le rang de la matrice peut être estimé par la quantité de termes élevés. [...] C'est propriétés font de la SVD un outil numériquement robuste pour l'analyse moderne des signaux.» (Rao *et al.* 1992, p. 289)

La SVD propose la décomposition d'une matrice en trois sous-matrices de même ordre d'après la relation suivante :

$$\widehat{H}_{\alpha\beta}(0) = U_{\alpha\beta}\Sigma_{\alpha\beta}{V_{\alpha\beta}}^{H} \tag{2.24}$$

Les matrices $U_{\alpha\beta}$ et $V_{\alpha\beta}$ de cette décomposition sont orthogonales et représentent respectivement les vecteurs singuliers gauches et droits de la matrice $\widehat{H}_{\alpha\beta}$. Elles répondent toutes deux à la relation ${U_{\alpha\beta}}^{H}U_{\alpha\beta} = I$ et ${V_{\alpha\beta}}^{H}V_{\alpha\beta} = I$. La matrice $\Sigma_{\alpha\beta}$ est de forme diagonale et contient les valeurs singulières de la matrice $\widehat{H}_{\alpha\beta}$ en ordre décroissant.

$$\Sigma_{\alpha\beta} = diag(\sigma_1,\ \sigma_2,\ldots,\ \sigma_n,\ \sigma_{n+1},\ldots,\ \sigma_L)$$
$$\text{avec } \sigma_1 \geq \sigma_2 \geq \cdots\ \sigma_i \geq \sigma_{n+1} \geq \cdots \sigma_L \geq 0 \tag{2.25}$$

Estimation de l'ordre du système

La distribution des valeurs de $\Sigma_{\alpha\beta}$ est décisive, car c'est à partir de celle-ci qu'est déterminé l'ordre minimal n du modèle d'état équivalent $\hat{A}, \hat{B}, \hat{C}$. Ainsi, il est crucial de choisir statistiquement une valeur σ_n comme point milieu entre les valeurs singulières dominantes et celles qui sont indésirables à la spécification du modèle. Suite à l'obtention de l'ordre n, l'équation $\hat{H}_{\alpha\beta}(0)$ (2.24) est reformulée :

$$\hat{H}_n(0) = U_n \Sigma_n V_n^H \tag{2.26}$$

puis factorisée :

$$\hat{H}_n(0) = (U_n \Sigma_n^{1/2})(\Sigma_n^{1/2} V_n^H) \tag{2.27}$$

En comparant cette décomposition à celle de $\hat{H}_{ij}(0)$ (2.21), on obtient directement les équations d'ordre minimal pour les matrices $\mathbb{Q}_\alpha$ et $\mathbb{C}_\beta$, soit :

$$\mathbb{Q}_n = (U_n \Sigma_n^{1/2}); \quad \mathbb{C}_n = (\Sigma_n^{1/2} V_n^H) \tag{2.28}$$

De la même façon, cette décomposition jumelée à $\hat{H}_{\alpha\beta}(1)$ (2.22), introduit le terme F d'ordre minimal d'après :

$$\hat{H}_n(1) = (U_n \Sigma_n^{1/2}) F_n (\Sigma_n^{1/2} V_n^H) \tag{2.29}$$

et sa mise en évidence selon la propriété d'orthogonalité des matrices U_n et V_n :

$$\boxed{F_n = (\Sigma_n^{-1/2} U_n^H) \hat{H}_n(1) (V_n \Sigma_n^{-1/2})} \tag{2.30}$$

2.3.5 Détermination du modèle boîte-noire

Une fois la matrice F_n et les vecteurs $\mathbb{C}_n$ et $\mathbb{Q}_n$ obtenus, l'identification du modèle d'état se complète à partir des notions d'opérateurs matriciels comme le démontre Juang (1994, p. 136) « en effet, pour $E_m^T = [I_m \quad O_m \quad \dots \quad O_m]$, où m représente le nombre de sorties et $E_r = [I_r \quad O_r \quad \dots \quad O_r]$, où r représente le nombre d'entrées », l'équation Y_k (2.15) est reformulée :

$$Y_k = E_m^T \widehat{H}_{\alpha\beta}(k) E_r \tag{2.31}$$

En comparant l'équation de Y_k (2.15) et de $\widehat{H}_{\alpha\beta}(k)$ (2.23), de même qu'en appliquant l'effet de la réduction de l'ordre en (2.28), l'équation (2.31) implique :

$$G_n = (\Sigma_n^{1/2} V_n^H E_r); \quad P_n = (E_m^T U_n \Sigma_n^{1/2}) \tag{2.32}$$

Avec la forme de E_r et de E_m^T associée à la configuration SISO du modèle, les équations sont réduites à :

$$\begin{aligned} G_n &= (\Sigma_n^{1/2} V_n^H) = \mathbb{C}_n \\ P_n &= (U_n \Sigma_n^{1/2}) = \mathbb{Q}_n \end{aligned} \tag{2.33}$$

et finalement grâce aux transformations du domaine discret à continu :

$$\boxed{\begin{aligned} \widehat{A} &= \log(F_n)/\Delta t \\ \widehat{B} &= G_n \Big/ \left(\widehat{A}^{-1}(F_n - I)\right) \\ \widehat{C} &= P_n * \Delta t \end{aligned}} \tag{2.34}$$

2.4 Conclusion

Les démonstrations mathématiques de ce chapitre soutiennent qu'il est possible d'obtenir un modèle d'état continu d'ordre minimal à partir de tout signal échantillonné en ne connaissant que le contenu et la période d'échantillonnage Δt de celui-ci. La méthode d'identification du modèle d'état se résume à cinq étapes :

1. Créer une matrice de Hankel pleine $\widehat{H}_{\alpha\beta}(0)$ à partir de $y_{ech}(1)$.
2. Obtenir l'ordre n du modèle par analyse des valeurs singulières de $\widehat{H}_{\alpha\beta}(0)$.
3. Obtenir les termes G_n et P_n de la matrice réduite $\widehat{H}_n(0)$.
4. Obtenir le terme F_n de la matrice réduite $\widehat{H}_n(1)$, créer à partir de $y_{ech}(2)$.
5. Obtenir $\hat{A}, \hat{B}, \hat{C}$ à l'aide de F_n, G_n, P_n.

Les techniques employées assurent que le modèle boîte-noire produise la réponse la plus conforme possible à $y_{ech}(k)$ lorsqu'excité par une impulsion de Dirac δ. De plus, les termes matriciels employés dans le modèle et calculés par décomposition singulière favorisent l'analyse par valeurs propres grâce aux éléments orthogonaux qu'implique la SVD.

L'hypothèse de réponse à l'impulsion amenée au début de ce chapitre est applicable au reste de la méthodologie. En effet, en étendant cette idée au domaine de Laplace, il est possible de considérer un système dont l'équation de sortie correspond à sa fonction de transfert. Cette perspective est mise à profit au chapitre suivant par les liens qui unissent le modèle d'état identifié et une fonction de transfert de forme résiduelle.

La méthode ERA d'origine développée par Juang et Pappa (1985, p. 620-627) propose normalement de réduire la matrice de Hankel en éliminant certaines rangées ou colonnes tout en préservant le premier terme Y_k pour alléger la tâche du calcul de la SVD. Dans le cadre de cette recherche, le processus est plutôt remplacé par une étape de décimation du signal,

effectuée préalablement à la formation de la première matrice $\widehat{H}_{\alpha\beta}(0)$. Ce choix s'explique par le souci de maximiser l'information à l'utilisateur tout en offrant un niveau de contrôle plus approfondi au développeur. Les enjeux de cette modification son détaillés au chapitre 4.

CHAPITRE 3

ESTIMATION DES PARAMÈTRES

3.1 Introduction

Le présent chapitre constitue le point culminant de la méthodologie, où le comportement dynamique du système est extrait d'après son identification obtenue précédemment. Ainsi, à partir du modèle d'état identifié au chapitre 2, il est possible d'obtenir les caractéristiques précises des différents modes contenus dans le signal. Pour se faire, il est nécessaire d'émettre certaines hypothèses qui posent les bases du présent développement. Par la suite, l'approche de Prony est explicitée et les équations du système sont revues de manière à introduire les notions de fonction de transfert résiduelle et d'éléments propres. À partir de cette approche, une technique par pôle/résidu permet d'obtenir les paramètres recherchés et d'être finalement en mesure de reconstruire un signal $\hat{y}_{ech}(k)$ équivalent à $y_{ech}(k)$.

3.2 Hypothèses

La première hypothèse détermine que le signal étudié ne produit que des valeurs propres distinctes lors de sa décomposition. Dans un contexte théorique, cette idée peut sembler erronée, surtout lorsque le risque de pôles multiples est augmenté par des éléments similaires à l'intérieur du système. Cependant, l'étude de cas pratiques démontre le contraire :

> « Pour des réseaux électriques, cette hypothèse peut être restrictive considérant que certaines fonctions de transfert de stabilisateur présentent par conception des pôles multiples. Mais le fait demeure que les modes dominants du système (modes oscillatoires d'origine électromécanique) seront généralement distincts, tout en étant parfois très rapprochés les uns des autres. » (Kamwa *et al.* 1997, p. 76)

Il est donc légitime de généraliser cette idée et de stipuler qu'en pratique, les cas de pôles multiples sont inexistants.

La seconde hypothèse reprend celle d'un système répondant à l'impulsion, $u(t) = \delta$, tel que vue la section 2.2.1. Cette fois-ci, cette perspective est appliquée au contexte d'une fonction de transfert dans le domaine de Laplace et du signal d'entrée qui en découle selon :

$$u(s) = \int_0^\infty \delta(t)\, e^{-st} dt = 1 \tag{3.1}$$

3.3 Approche de Prony

La méthode d'estimation des paramètres du signal retenue dans ce travail s'établit à partir de l'approche de Prony. Cette approche représente tout signal sinusoïdal $y(t)$ comme une somme de sinusoïdes indépendantes possédant leurs propres composantes d'amplitude initiale a_k, de taux d'amortissement σ_k, de fréquence d'oscillation ω_k et de déphasage θ_k tel que :

$$y(t) = \sum_{k=1}^{n} a_k e^{(\sigma_k t)} \cos(\omega_k t + \theta_k) \tag{3.2}$$

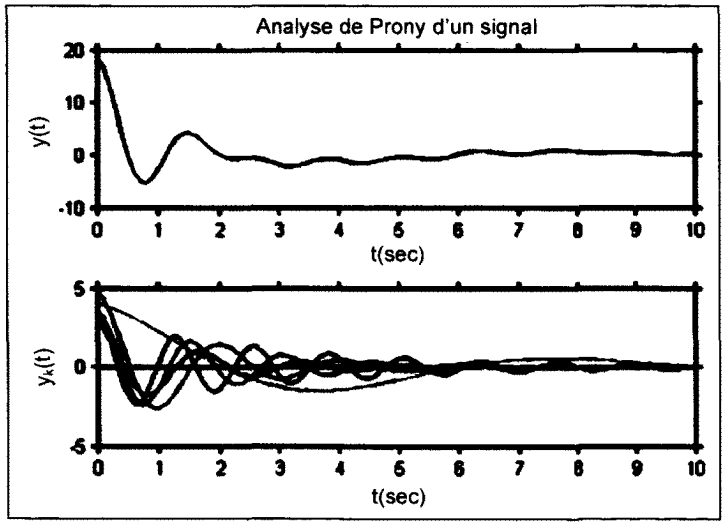
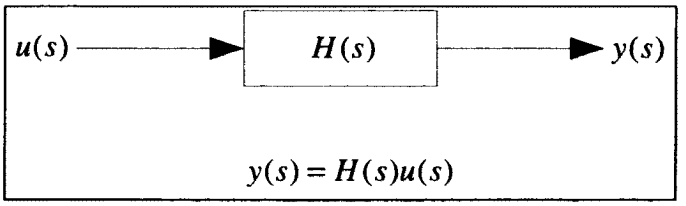


Figure 3.1 Représentation par forme d'onde de l'approche de Prony.

3.3.1 Équations du système

Pour déterminer le comportement dynamique du système, il est primordial d'ajuster la définition par équations d'état à une description par fonction de transfert dans le domaine de Laplace pour un système simple tel que :



(3.3)

Figure 3.2 Représentation synoptique d'une fonction de transfert.

La transformée de Laplace inverse de $\Phi(t)$ dans l'équation de *H(t)* (2.12) modifie l'expression de $y(s)$ conformément à :

$$y(s) = C\left(sI - A\right)^{-1} Bu(s) \tag{3.4}$$

En combinant cette équation aux hypothèses de valeurs propres distinctes pour *A* et d'entrée impulsionnelle pour $u(s) = 1$, Kundur *et al.* (1994, p. 720) démontre que l'expression de sortie peut être ramenée à :

$$y(s) = \sum_{k=1}^{n}\left[\frac{R_k}{s - \lambda_k}\right] \tag{3.5}$$

La transformée de Laplace inverse de cette expression redéfinie l'expression $y(t)$ (3.2) conformément à des notions de résidus et de valeurs propres du système :

$$y(t) = R_1 e^{\lambda_1 t} + R_2 e^{\lambda_2 t} + \ \ldots + R_n e^{\lambda_n t} \tag{3.6}$$

Les sections subséquentes démontrent de quelle façon calculer ces termes à partir des éléments générés au chapitre précédent et en retirer les paramètres dynamiques recherchés.

3.3.2 Éléments propres et résidus

La décomposition par éléments propres d'un modèle d'état permet d'identifier une partie importante de son comportement dynamique. En effet, lorsque le conditionnement de A est adéquat, les renseignements sur la fréquence et l'amortissement de chaque mode peuvent être obtenus grâce aux valeurs propres λ_k de celle-ci. De plus, ce sont les vecteurs propres droits et gauches de cette même matrice qui déterminent les termes résiduels R_k associés au système.

Valeurs et vecteurs propres

Il convient de rappeler que les éléments propres d'une matrice s'associent de manière à résoudre les équations mathématiques suivantes:

$$\begin{aligned} A\phi_k &= \lambda_k \phi_k, \\ (A - \lambda_k I)\phi_k &= 0 \end{aligned} \tag{3.7}$$

où le vecteur ϕ_k contient les vecteurs propres droits associés aux valeurs propres λ_k selon :

$$\begin{aligned} \phi_k &= [\phi_1,\ \phi_2,\ \ldots\ ,\ \phi_n] \\ \lambda_k &= [\lambda_1,\ \lambda_2,\ \ldots\ ,\ \lambda_n] \end{aligned} \tag{3.8}$$

De même, les vecteurs propres gauches ψ_k sont les termes qui vérifient :

$$\begin{aligned} \psi_k A &= \lambda_k \psi_k \\ \psi_k &= [\psi_1,\ \psi_2,\ \ldots\ ,\ \psi_n] \\ \lambda_k &= [\lambda_1,\ \lambda_2,\ \ldots\ ,\ \lambda_n] \end{aligned} \tag{3.9}$$

Pour des éléments propres de nature complexe, les vecteurs propres gauches se déterminent par la pseudo-inversion des vecteurs propres droits. Cette opération normalise les éléments droits et gauches tels que :

$$\Psi^H = \Phi \tag{3.10}$$

$$\psi_i \phi_j = \begin{cases} 0, & i \neq j \\ 1, & i = j \end{cases} \tag{3.11}$$

Caractéristiques temporelles et oscillatoires

L'équation de $y(t)$ (3.6) prouve le lien entre les valeurs propres du modèle et son comportement temporel suivant : $\lambda_k \Rightarrow e^{\lambda_k t}$. Kundur résume bien les différents cas de valeurs propres pouvant survenir et leurs effets sur la stabilité du mode auquel chacune est associée :

> «Une valeur propre réelle correspond à un mode non oscillatoire. Une valeur propre négative représente un mode amorti. Plus cette valeur est élevée, plus le mode tend à s'amortir rapidement. Les valeurs du résidu et du vecteur propre associées à un mode réel sont aussi réelles.
>
> Les valeurs propres complexes surviennent toujours par paires de conjugués complexes et chaque paire correspond à un seul mode oscillatoire. Les valeurs de résidus et de vecteurs propres associées à un mode conjugué complexe forment une même paire produisant une valeur de y(t) réelle pour tout temps *t* d'après :

$$(a+jb)e^{(\sigma-j\omega)t} + (a-jb)e^{(\sigma+j\omega)t} \ \Rightarrow\ e^{\sigma t}\sin(\omega t + \theta) \tag{3.12}$$

ce qui représente une sinusoïde amortie lorsque le terme σ_k est de signe négatif.

L'amortissement d'une valeur propre complexe est donné par sa composante réelle, alors que la fréquence d'oscillation est donnée par sa composante imaginaire. Une composante réelle négative représente une oscillation amortie, tandis qu'une composante positive représente une oscillation d'amplitude croissante. Donc, dans le cas d'une valeur propre complexe : » (Kundur *et al.* 1994, p. 711-712)

$$\boxed{\begin{aligned} \lambda_k &= \sigma_k \pm j\omega_k \\ f_k &= \omega_k / 2\pi \end{aligned}} \tag{3.13}$$

L'étude des valeurs propres de la matrice $\hat{A}$ permet de mettre à jour les paramètres d'amortissement et de fréquence d'oscillation des modes à identifier.

Matrice des résidus

La matrice des résidus est mise en évidence en comparant les termes des expressions de sortie du signal *y(s)* en (3.4) et (3.5) pour former :

$$R_k = \left[(s-\lambda_k) C (sI - A)^{-1} B \right]\Big|_{s=\lambda_k} \tag{3.14}$$

En conjuguant l'hypothèse que les valeurs propres λ_k sont distinctes au fait que les vecteurs propres produits par la matrice *A* sont orthogonaux, le calcul des termes R_k se résume à :

$$R_k = C\phi_k \psi_k^H B \tag{3.15}$$

Une substitution pour les vecteurs propres de $\hat{A}$ et les termes $\hat{B}$ et $\hat{C}$ conclue le lien avec la réalisation ERA :

$$\boxed{\hat{R}_k = \hat{C}\phi_k\psi_k^H\hat{B}} \tag{3.16}$$

3.3.3 Obtention des paramètres

À partir de la décomposition par valeurs propres et du calcul des résidus, tous les éléments sont réunis pour estimer la totalité des paramètres dynamiques du signal $y_{ech}(k)$. Il est toutefois nécessaire d'ajuster la décomposition en fraction partielle de $y(s)$ (3.5) pour inclure les valeurs de conjugués complexes $\sigma_k \pm j\omega_k$ et la forme connue de $\hat{R}_k$:

$$y(s) = \sum_{i=1}^{l}\left[\frac{\hat{R}_i}{s-\sigma_i}\right] + \sum_{j=1}^{\frac{n-l}{2}}\left[\frac{\hat{R}_j}{s-\sigma_j - j\omega_j}\right] + \sum_{j=1}^{\frac{n-l}{2}}\left[\frac{\hat{R}_j}{s-\sigma_j + j\omega_j}\right] \tag{3.17}$$

qui mène finalement à :

$$\boxed{\hat{y}_{ech}(k) = \sum_{i=1}^{l}\hat{R}_i e^{\sigma_i k\Delta t} + 2\left(\sum_{j=1}^{\frac{n-l}{2}}\left|\hat{R}_j\right| e^{\sigma_j k\Delta t}\cos\left(\omega_j k\Delta t + \theta_j\right)\right)} \tag{3.18}$$

Il est alors possible de décrire entièrement le comportement dynamique du système par la méthode de Prony, suivant qu'un mode soit issu d'une valeur propre réelle ou d'une paire de conjugués complexes. La forme explicite de $\hat{y}_{ech}(k)$ proposée marque ainsi une différence dans le calcul des paramètres de chaque mode et dans leurs comportements oscillatoires :

➢ Pour un mode issu d'une valeur propre réelle, les paramètres sont :

$$\boxed{\begin{aligned} a_k &= \hat{R}_i \\ \sigma_k &= \sigma_i \\ \omega_k &= 0 \\ \theta_k &= 0 \end{aligned}} \tag{3.19}$$

➢ Pour un mode issu d'une paire de conjugués complexes, les paramètres sont :

$$\boxed{\begin{aligned} a_k &= \frac{\left|\hat{R}_j\right|}{2} \\ \sigma_k &= \sigma_j \\ \omega_k &= \omega_j \\ \theta_k &= \arg\left(\hat{R}_j\right) \end{aligned}} \tag{3.20}$$

Amortissement relatif

La donnée dynamique σ_k d'un mode indique l'évolution de l'amortissement qui affecte le mode *k*, à chaque échantillon du signal. Ainsi, un mode qui possède un fort taux d'amortissement et une faible fréquence d'oscillation peut dessiner une courbe enveloppe semblable à celle d'un mode possédant un faible taux d'amortissement et une fréquence élevée. Bien que l'amortissement relatif ne soit pas une caractéristique dynamique physique, il permet d'exprimer le taux de décroissance (ou croissance) d'un mode de manière proportionnelle, plutôt qu'en fonction de sa fréquence d'oscillation. Cette information est des plus pertinente pour comparer l'implication de chacun des modes sur la forme globale du signal analysé. L'amortissement relatif se calcule d'après :

$$\boxed{\zeta_k = \frac{-\sigma_k}{\sqrt{\sigma_k^2 + \omega_k^2}} \Rightarrow \{0 \le \zeta_k \le 1\}} \tag{3.21}$$

3.4 Conclusion

La démarche mathématique présentée dans ce chapitre démontre qu'il est possible d'obtenir l'ensemble des paramètres de Prony d'un signal à partir d'un modèle d'état continu, si celui-ci peut être soumis à une décomposition par valeurs propres. Cette démarche comporte quatre étapes principales :

1. Décomposer la matrice $\hat{A}$ en valeurs et vecteurs propres λ_k et ϕ_k.
2. Obtenir les vecteurs propres gauches ψ_k par la pseudo-inversion de ϕ_k.
3. Obtenir les termes résiduels $\hat{R}_i$ et $\hat{R}_j$ selon $\hat{B}$, $\hat{C}$, ϕ_k et ψ_k.
4. Obtenir les paramètres a_k, σ_k, ω_k, θ_k en fonction des égalités associées aux valeurs réelles ou conjuguées complexes de λ_k.

Le calcul d'un signal $\hat{y}_{ech}(k)$ en tous points semblable au signal original $y_{ech}(k)$ est rendu possible grâce aux composantes dynamiques obtenues. Le calcul s'effectue directement à partir de l'équation (3.18) avec une même référence pour $k=1$ et une période d'échantillonnage Δt identique.

La pertinence de certains modes dans la reconstitution de $y_{ech}(k)$ doit par contre être réévaluée selon la nature du signal. Parfois, des modes qui semblent inopportuns à la reproduction du signal en constituent une partie importante. C'est le cas d'un mode à fréquence nulle ($\omega_k \cong 0$) qui peut représenter un phénomène d'intérêt lorsque le signal analysé provient d'une lecture en p.u.. Le même genre de réflexion s'applique pour un mode sans amortissement ($\sigma_k \cong 0$) qui se trouve à être la fondamentale d'une lecture de tension sinusoïdale à 60Hz.

L'échantillonnage numérique peut aussi produire un lot de modes indésirables comme les erreurs de troncature ($a_k \gg$, $\sigma_k \gg$) ou de bruit numérique ($a_k \ll$, $\sigma_k \ll$) s'étant infiltré malgré les procédures mises en place.

Ces notions et celle de l'amortissement relatif s'ajoutent donc à la démarche théorique proposée jusqu'ici pour créer un algorithme d'analyse modale ERA/Prony robuste face aux données numériques de l'étude. Le chapitre suivant s'attarde à l'architecture et au fonctionnement de cet algorithme.

CHAPITRE 4

ALGORITHME ERA / PRONY

4.1 Introduction

Ce chapitre présente le produit logiciel mis au point à partir de la méthodologie mathématique expliquée aux chapitres précédents. Le produit en question permet de conduire une analyse modale complète sur un signal numérique, peu importe sa provenance et selon des conditions spécifiques, aux choix de l'utilisateur. Le produit livré s'exécute à l'intérieur de n'importe quel programme hôte possédant des propriétés de traitement du langage C et d'affichage graphique.

Le chapitre décrit premièrement l'architecture logicielle à partir du rôle de chacun de ses fichiers. Par la suite, les interactions qui animent cette architecture sont résumées par le fonctionnement global et la description des arguments d'entrée et de retour. La dernière partie du chapitre s'intéresse à la fois aux spécificités des langages de programmation, à la portabilité et à la gestion de mémoire physique du code source.

4.2 Architecture logicielle

L'architecture du programme se divise en trois bibliothèques logicielles. Ces bibliothèques interagissent entre elles et le programme hôte d'après les tâches exécutées par leurs fonctions. Les bibliothèques *Linear Algebra PACKage* (LAPACK©) et *Numerical Recipes* (NR©) proviennent de sources externes et sont intégrées à travers l'algorithme ERA/Prony. Cet algorithme représente la troisième bibliothèque et par le fait même, le contenu informatique développé dans cette recherche. La structure et les liens qui régissent le programme sont mis en évidence par le schéma bloc suivant :

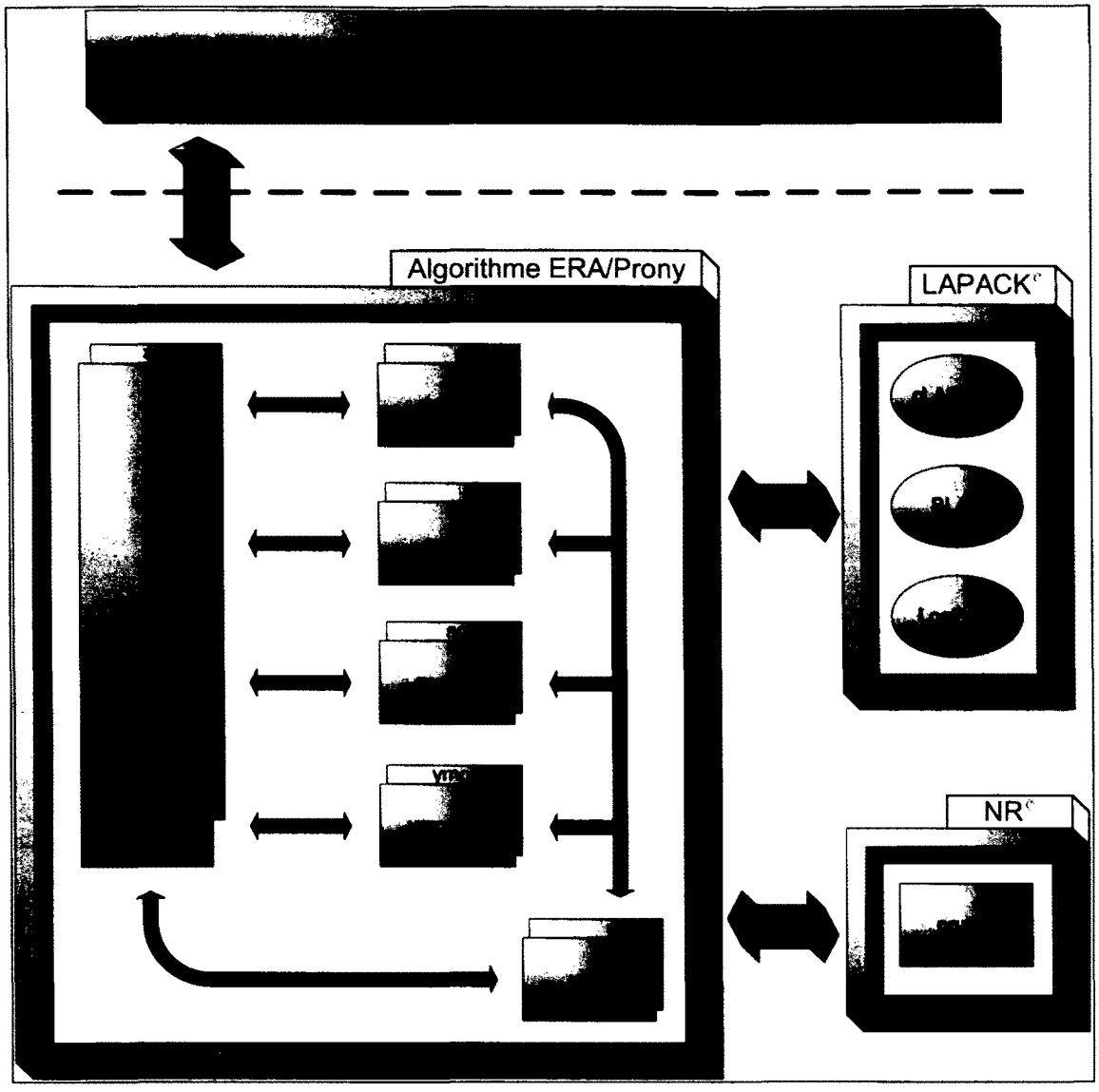


Figure 4.1 Architecture logicielle.

4.2.1 Algorithme ERA/Prony

La bibliothèque de l'algorithme ERA/Prony constitue la pierre angulaire de ce programme d'analyse modale. C'est à l'intérieur de celle-ci que l'on retrouve le fichier de fonction principale (MAIN) et les fichiers complémentaires regroupant les fonctions spécifiques à chaque étape de la méthode. Un fichier de fonctions utilitaires est aussi inclus pour empêcher des redondances inutiles dans l'algorithme.

Les liens entre les fonctions sont établis de façon à respecter les étapes du calcul et simplifier la consultation du code. Le fichier MAIN a l'entière responsabilité de contrôler l'échange d'informations entre les fichiers complémentaires. Le fichier de fonctions utilitaires et les bibliothèques LAPACK© et NR© sont quant à eux accessible à tous les niveaux de l'algorithme.

L'algorithme ERA/Prony comporte deux types de fichiers. Les fichiers de code source (*.c) contiennent la définition complète des fonctions C et les références aux bibliothèques internes ou externes qu'elles utilisent. Chacun de ces fichiers s'accompagne d'un fichier d'en-tête (*.h), dans lequel on retrouve la déclaration des fonctions, en plus d'informations sur leur rôle, leurs arguments d'entrée et ceux qu'elles retournent. Pour des raisons de convivialité, le contenu complet de ces fichiers est restreint à l'annexe.

Fichier AMOD.C

Le fichier « amod.c » contient les fonctions qui exécutent l'algorithme et gèrent le trafic d'information avec le programme hôte. Les principales tâches exécutées par les fonctions de « amod.c » sont :

- Traiter l'appel du programme hôte.
- Orchestrer l'exécution de l'algorithme.
- Acheminer les arguments de retour au programme hôte.
- Libérer la mémoire physique occupée par les arguments de retour suite à leur utilisation.

Fichier DATA.C

Le fichier « data.c » contient les fonctions qui préparent les données du signal afin d'optimiser le temps de calcul de l'algorithme et approuver son exécution. Les principales tâches exécutées par les fonctions de « data.c » sont :

- Valider les arguments d'entrée soumis par l'utilisateur.
- Prétraiter le signal d'après les caractéristiques de l'analyse demandée :
  - Déterminer de manière optimale le taux de décimation et les dimensions de la matrice de Hankel par rapport au signal original.
  - Réduire le signal par décimation et ajuster les arguments impliqués.
- Produire les messages d'avertissement ou d'erreur selon le cas.

Fichier ERA.C

Le fichier « era.c » gère les fonctions permettant d'obtenir un modèle d'état réduit $\hat{A}$, $\hat{B}$, $\hat{C}$, dont la réponse à l'impulsion correspond au signal $y_{ech}(k)$ analysé par l'utilisateur. Les principales tâches exécutées par les fonctions de « era.c » sont :

- Assembler les matrices de Hankel.
- Appeler la décomposition par valeurs singulières d'une matrice réelle.
- Estimer l'ordre minimal d'une décomposition par valeurs singulières.
- Identifier le système d'état discret d'ordre minimal.
- Identifier le système d'état continu d'ordre minimal.

Fichier APRON.C

Le fichier « apron.c » gère les fonctions permettant d'obtenir les paramètres dynamiques a_k, σ_k, ω_k et θ_k des modes oscillatoires du signal échantillonné. Les principales tâches exécutées par les fonctions de « apron.c » sont :

- Appeler la décomposition par éléments propres d'une matrice.
- Produire la matrice des résidus d'une décomposition par éléments propres.
- Déterminer les valeurs de tous les paramètres de Prony.
- Filtrer les informations retournées en fonction des limites fixées par l'utilisateur.

- Produire les messages de mode de fonctionnement.

Fichier YMODEL.C

Le fichier « ymodel.c » contient une fonction qui calcule le signal $\hat{y}_{ech}(k)$ à partir des paramètres dynamiques retenus à cet effet.

Fichier UTIL.C

Le fichier utilitaire regroupe les fonctions et les structures de nature générale, communes à plusieurs fonctions de l'algorithme. Les principales tâches exécutées par les fonctions de « util.c » sont :

- Définir les structures de type : vecteur, matrice, paramètres de Prony et message.
- Évaluer l'égalité entre deux nombres réels.
- Gérer la mémoire tampon associée aux messages.

4.2.2 LAPACK©

La bibliothèque logicielle LAPACK© inclut des routines de solutions aux problèmes d'algèbre linéaire comme la factorisation de matrices, ou la résolution d'équations linéaires. Cette bibliothèque provient de travaux de l'université du Tennessee distribués gratuitement via l'organisme Netlib[1]. Les routines sont codées en langage Fortran90 et s'optimisent pour la portabilité sur différents types de processeur. Les fonctions de LAPACK© sont appelées indirectement par l'entremise de ses trois sous-programmes : cLAPACK©, BLAS© et LogPack©.

---

[1] http://www.netlib.org/lapack/

cLAPACK©

Le sous-programme cLAPACK© regroupe les fonctions d'algèbre linéaire évoluées qui sont appelées par l'algorithme ERA/Prony. Les fonctions de cLAPACK© sont directement obtenues de LAPACK© par une conversion du langage Fortran90 à C, à l'aide du compilateur F2C© de Netlib[2]. Les principales tâches exécutées par les fonctions de cLAPACK© sont :

- Calculer les valeurs et vecteurs singuliers d'une matrice de nombres réels par un algorithme *divide-and-conquer.*
- Calculer les valeurs et vecteurs propres normalisés d'une matrice de nombres réels.
- Calculer l'inverse d'une matrice de nombres réels par un algorithme de décomposition LU.
- Calculer le pseudo-inverse d'une matrice de nombres complexes par un algorithme de factorisation QR.

BLAS©

Le sous-programme BLAS© regroupe les fonctions d'algèbre linéaire simples qui sont appelées par l'algorithme ERA/Prony et les fonctions plus évoluées de cLAPACK©. Ce sous-programme est inclus à même la bibliothèque LAPACK© en langage Fortran90 et la conversion de ses fonctions au langage C s'effectue avec F2C©. Les principales tâches exécutées par les fonctions de BLAS© sont :

- Calculer le produit entre une matrice et un vecteur de nombres réels.
- Calculer le produit entre deux matrices de nombres réels.
- Calculer le produit entre une matrice et un vecteur de nombres complexes.

---

[2] http://www.netlib.org/f2c/

LogPack©

Le sous-programme LogPack© regroupe les fonctions d'algèbre linéaire permettant de calculer le logarithme d'une matrice selon une variante de la technique de mise à l'échelle inverse, au carré, selon Cheng *et al.* (2001, p. 1112-1125). Ce sous-programme provient des travaux d'un consortium entre le département de mathématiques et de sciences informatiques de l'université de Manchester[3]. Ses fonctions sont construites à partir de celles de LAPACK© et converties en langage C avec F2C©.

4.2.3 NR©

La bibliothèque NR© contient un ensemble d'outils, de fonctions et de programmes de calculs numériques écrits pour le langage C. Cette bibliothèque provient des travaux de H. Press *et al.* (1992). Son utilisation par l'algorithme ERA/Prony se limite toutefois aux déclarations d'éléments matriciels et vectoriels et à la gestion de leur mémoire physique.

4.3 Fonctionnement global

Pour arriver à une analyse modale complète, l'exécution du programme se divise en quatre étapes principales :

1. Appel de l'algorithme.
2. Prétraitement du signal.
3. Analyse modale ERA/Prony.
4. Retour des paramètres de Prony et du signal reconstitué.

---

[3] http://www.maths.manchester.ac.uk/~higham/PCMF/

L'analyse est lancée par le programme hôte lorsqu'il appelle la fonction principale du fichier « amod.c ». Une fois les arguments d'entrée saisis, cette fonction enclenche l'exécution automatique de l'analyse ERA/Prony pour les conditions déterminées lors de l'appel.

Premièrement, la période d'échantillonnage et la plage de fréquence demandées sont évaluées par le fichier « data.c » pour établir le niveau de prétraitement nécessaire au signal. Selon le cas, le signal est décimé pour assurer que l'ordre de la matrice de Hankel reste inférieur ou égal à 1000. Ce traitement a pour but de réduire le temps de calcul de l'algorithme tout en respectant les résultats que produirait une étude exhaustive de l'échantillonnage. Les irrégularités au niveau des valeurs d'arguments et des limites de l'analyse sont aussi détectées à cette étape. À tout moment, si une erreur est détectée, l'exécution de l'algorithme est arrêtée et un message décrivant les circonstances est retourné à l'utilisateur. L'analyse modale s'effectue ainsi sur un échantillon optimisé, dans des conditions contrôlées.

Deuxièmement, la démarche ERA/Prony est réalisée par les fichiers « era.c » et « apron.c » en s'appuyant directement sur les techniques développées dans la méthodologie. Les composantes dynamiques obtenues sont utilisées parallèlement par le fichier « ymodel.c » et le fichier « amod.c » pour produire des arguments de retour selon le niveau de filtrage fréquentiel et les limites d'amplitude et d'amortissement fixés par l'utilisateur. Le premier argument contient les paramètres dynamiques regroupés et classés en ordre décroissant d'amplitude. Le deuxième argument comprend les valeurs de chacun des échantillons du signal reconstruit $\hat{y}_{ech}(k)$.

Pour terminer, ces arguments et les avertissements sont envoyés à un programme apte à produire des résultats sous forme de tableaux et de graphiques cartésiens. Une fois les informations affichées, le programme hôte se doit d'appeler une fonction du fichier « amod.c » destinée à libérer la mémoire physique toujours occupée par l'algorithme.

4.3.1 Arguments d'entrée

Les arguments d'entrée du programme sont constitués de l'information du signal et des conditions d'analyse recherchées par l'utilisateur. Au total, l'algorithme possède neuf arguments obligatoires et un optionnel. Ces arguments sont requis par la fonction principale du fichier « amod.c » d'après un ordre et des types spécifiques. La déclaration de la fonction principale du fichier « amod.c » et les caractéristiques de chacun de ses arguments se décrivent comme suit :

amod(**double*** *y*, **int** *ndata*, **double** *t_ech*, **double** *f_low*, **double** *f_high*, **double** *alim*, **double** *zlim*, **double** *trig*, **int** *mod_order*, **int** *filter);*

1. Signal échantillonné
   - Nom : *y*
   - Type : Pointeur de nombres réels.
   - Définition : L'adresse mémoire où se trouve la totalité de l'échantillonnage sous forme de nombres réels.

2. Nombre d'éléments
   - Nom : *ndata*
   - Type : Nombre entier.
   - Définition : Le nombre total d'échantillons que contient le signal.

3. Période d'échantillonnage
   - Nom : *t_ech*
   - Type : Nombre réel.
   - Définition : La période de temps qui sépare chacun des échantillons contigus du signal.

4. Limite de fréquence basse
   - Nom : *f_low*
   - Type : Nombre réel.
   - Définition : La limite fréquentielle inférieure des modes oscillatoires retenus pour $\hat{y}_{ech}(k)$.
     - Cette limite doit être déterminée de façon à ce qu'au moins une période de *f_low* puisse être contenue dans la fenêtre temporelle analysée : $\{trig \leq t \leq t_{fin}\}$.
     - Cette limite doit être comprise entre : $\{0.05\text{Hz} \leq f\_low < f\_high\}$.

5. Limite de fréquence haute
   - Nom : *f_high*
   - Type : Nombre réel.
   - Définition : La limite fréquentielle supérieure des modes oscillatoires retenus pour $\hat{y}_{ech}(k)$.
     - Cette limite doit éviter le risque de sous-échantillonnage défini pour : $f\_high \leq (1/(4*t\_ech))$.

6. Limite d'amplitude
   - Nom : *alim*
   - Type : Nombre réel.
   - Définition : L'écart proportionnel maximal entre la plus grande et la plus petite amplitude des modes oscillatoires retenus pour affichage à l'utilisateur.
     - Cette limite fixe la plus petite valeur d'amplitude à partir de laquelle les modes sont retenus selon : $a_k \geq \left(\frac{a_{k_{max}}}{alim}\right)$.
     - Cette limite doit être comprise entre 1 et 100.

7. Limite d'amortissement relatif
   - Nom : *zlim*
   - Type : Nombre réel.
   - Définition : La limite d'amortissement relatif minimale des modes oscillatoires retenus pour affichage à l'utilisateur.
     - Cette limite doit être comprise entre : $\{0 \leq \text{zlim} \leq 0.707\}$.

8. Temps de déclenchement
   - Nom : *trig*
   - Type : Nombre réel.
   - Définition : L'indication temporelle qui fixe le temps du début de l'étude.
     - Cette indication doit être déterminée de manière à pouvoir observer au minimum une période de *f_low*.
     - Cette indication doit être comprise entre le début et la fin du signal et est automatiquement arrondie au premier échantillon inférieur disponible.

9. Ordre du modèle (automatique / manuel)
   - Nom : *mod_order*
   - Type : Nombre entier.
   - Définition : Le nombre d'éléments retenus lors de la réalisation minimale ERA du système.
     - Valeur de « 0 » : l'ordre du modèle est automatiquement déterminé selon la distribution des valeurs singulières issues de la matrice de Hankel.
     - Valeurs supérieures à « 0 » : l'ordre du modèle est déterminé manuellement selon le choix de l'utilisateur.
     - L'exécution en mode manuel est forcée en mode automatique si l'ordre choisi est plus grand que l'ordre maximal permis par le calcul de la matrice de Hankel.

10. Filtrage (optionnel)
    - Nom : *filter*
    - Type : Nombre entier.
    - Définition : Élément optionnel qui définit le type de filtrage appliqué aux composantes dynamiques retenues pour $\hat{y}_{ech}(k)$.
        - Valeur : « aucune » ou « 0 »
            - Filtrage des conjugués complexes.
            - Filtrage des modes instables : $\{0 < \sigma_k \leq 0.001\}$ (nécessaire en présence de bruit numérique).
            - Passe-bande parfait entre : $\{f\_low \leq f_k \leq f\_high\}$.
        - Valeur : « 1 »
            - Filtrage des conjugués complexes.
        - Valeur : « 2 »
            - Filtrage des conjugués complexes.
            - Filtrage des modes instables $\{0 < \sigma_k \leq 0.001\}$ (nécessaire en présence de bruit numérique).

4.3.2 Arguments de retour

Les arguments de retour du programme sont encapsulés et transmis au programme hôte par « amod.c » via un élément de type *struct* dénommé « t_amod_res ». La totalité des informations destinées à l'utilisateur s'y retrouve en trois sous-éléments de type *struct*. La déclaration de la structure de sortie se présente comme suit :

```
typedef struct{
        t_prony prony_usr;
        t_vec ymodel;
        t_amod_msg msg;
} t_amod_res;
```

Paramètres de Prony

Les paramètres de Prony de chacun des modes renvoyés à l'utilisateur sont contenus dans l'élément « t_prony » de « t_amod_res ». Cette structure est sous-divisée en cinq pointeurs de nombres réels correspondants aux composantes dynamiques des modes et un élément entier, contenant la quantité de modes retenus. La déclaration d'une structure « t_prony» est la suivante :

```
typedef struct{
        double *mag;
        double *damp;
        double *fn;
        double *theta_deg;
        double *zeta;
        int qty;
} t_prony;
```

avec les équivalences aux valeurs théoriques en (3.19) et (3.20) :

$$\text{mag} = a_k$$
$$\text{damp} = \sigma_k$$
$$\text{fn} = \left(\frac{\omega_k}{2\pi}\right)$$
$$\text{theta\_deg} = \left(\frac{(\theta_k * 180)}{\pi}\right)$$
$$\text{zeta} = \zeta_k$$

Signal échantillonné reconstruit

Le signal échantillonné reconstruit $\hat{y}_{ech}(k)$ est contenu dans l'élément « t_vec » de « t_amod_res ». Cette structure contient le vecteur des valeurs calculées pour chacun des

échantillons et la quantité de données qu'il contient. La déclaration d'une structure « t_vec » est la suivante :

```
typedef struct{
        double * vec;
        int nc;
} t_vec;
```

Le calcul de chaque échantillon est basé sur la formule $\hat{y}_{ech}(k)$ démontrée en (3.18) et ajusté pour les variables logicielles :

$$\text{ymodel.vec}[k] = \sum_{i=1}^{l} \text{mag}_i * e^{\text{damp}_i k\Delta t} + 2\left(\sum_{j=1}^{\frac{n-l}{2}} \text{mag}_j * e^{\text{damp}_j k\Delta t} \cos\left(\left(\text{fn}_j * 2\pi \right) k\Delta t + \left(\text{theta\_deg}_j * (\pi/180) \right) \right) \right)$$

Avertissements et erreurs

Les messages d'avertissement et d'erreur sont compris dans l'élément « t_amod_msg » de « t_amod_res ». Cette structure contient deux chaînes de caractères et deux nombres entiers qui indiquent la présence et le contenu des messages. Dans le cas d'une erreur, l'exécution de l'algorithme s'arrête et envoie le message correspondant au programme hôte. Dans le cas d'un avertissement, l'exécution se poursuit et le ou les messages sont envoyés à la fin de l'exécution. La déclaration d'une structure « t_amod_msg » est la suivante :

```
typedef struct{
        char * warnmsg;
        char * errmsg;
        int warncode;
        int errcode;
} t_amod_msg;
```

Les différents messages d'erreur et d'avertissement, ainsi que les cas pouvant les provoquer sont les suivants :

- *"Arret: Valeur de declenchement hors signal"*
  - Survient pour une valeur d'argument *trig* supérieure ou égale à la limite temporelle du signal.

- *"Arret: Valeurs de frequences non conformes"*
  - Survient pour une valeur d'argument *f_low* plus élevée que celle de l'argument *f_high.*
  - Survient pour des arguments *f_low* et *f_high* de valeurs égales.
  - Survient pour une valeur d'argument *f_low* inférieure à 0,05.
  - Survient pour une valeur d'argument *f_high* négative.

- *"Arret: Sous-echantillonnage "*
  - Survient pour une valeur d'argument *f_low* trop élevée qui ne génère pas suffisamment d'échantillons pour continuer l'analyse. La limite fréquentielle inférieure doit pouvoir contenir au moins 100 échantillons en fonction de : $(f\_low*t\_ech)^{-1} \geq 100$.

- *"Arret: Donnees insuffisantes "*
  - Survient pour une valeur d'argument de *trig* qui réduit le signal à un nombre d'échantillons inférieur au minimum nécessaire à l'exécution de l'algorithme.

- *"Avertissement : Etude limitee entre* `(Hz <-> Hz)` *"*
  - Survient pour une largeur de bande trop grande entre *f_low* et *f_high.* La validité de l'analyse est limitée à la fenêtre définie par l'avertissement. L'algorithme est optimisé pour traiter indépendamment chaque borne et l'ajuster au plus près de sa valeur initiale.

- Survient pour une valeur d'argument de *trig* qui réduit le signal à une quantité d'échantillons inférieure au minimum nécessaire pour couvrir la largeur de bande demandée.
  - La comparaison graphique entre $y_{ech}(k)$ et $\hat{y}_{ech}(k)$ permet de valider le contenu des données.

➢ *"Avertissement : Sur-echantillonnage"*

- Survient pour une valeur d'argument *f_low* trop basse qui demande un nombre d'échantillons hors des limites de traitement de l'algorithme. Le taux de décimation est ajusté à la hausse, mais risque d'entraîner des erreurs dans la plage de fréquence étudiée.
  - La comparaison graphique entre $y_{ech}(k)$ et $\hat{y}_{ech}(k)$ permet de valider le contenu des données.

4.4 Notes sur la programmation

4.4.1 Convention du Fortran90 dans un environnement C

Bien que les fonctions des sous-programmes de la bibliothèque LAPACK© soient compilées pour un environnement en langage C, leurs utilisations doivent respecter les conventions d'appel et d'arguments du langage Fortran90, soit :

➢ Appeler les fonctions par la référence de leurs arguments. Cette convention implique l'utilisation de pointeurs pour remplacer les éléments scalaires ou chaîne de caractères. Dans le cas des chaînes de caractères, le langage Fortran90 est *case sensitive* et évalue seulement le premier terme. L'exemple suivant illustre les changements qu'impliquent ces conventions :

- Appel en langage Fortran90 :
  - call dgXX('Upper', 5, 5, A, 5, ipiv, info)

- Appel en langage C, selon les conventions du Fortran90 :
  - M = N = LDA = 5;
  - char s = 'U';
  - dgXX_(&s, &M, &N, A, &LDA, ipiv, info);

➢ Ordonner les arguments de données matricielles selon le standard *column-major order*. Cette convention implique de modifier les donnés normalement assemblées selon le standard *row-major order*, avant et après chacune de leurs utilisations par les sous-programmes de LAPACK©. De plus, la convention Fortran90 demande de fournir les données de la matrice dans un vecteur et de l'accompagner de son nombre de rangées et de colonnes. L'exemple suivant illustre les changements impliqués par ces conventions :

- Forme matricielle utilisée en langage C :
  - Argument : Double pointeur de réels (`double** a[i][j]`).
  - Format : *row-major order.*

- Forme matricielle modifiée pour le langage Fortran90 :
  - Argument : Pointeur de réels (`double* a[k]`).
    Nombres entiers (`int i,j`).
  - Format : *column-major order.*

La différence entre le format *row-major order* et *column-major order* se distingue par l'ordre d'enregistrement des donnés de la matrice d'après les schémas suivants :

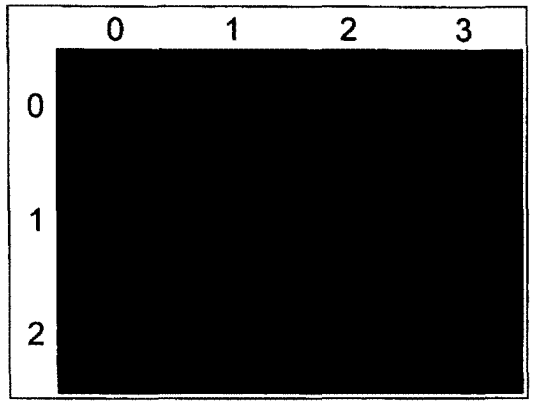


Figure 4.2 Format *row-major order.*

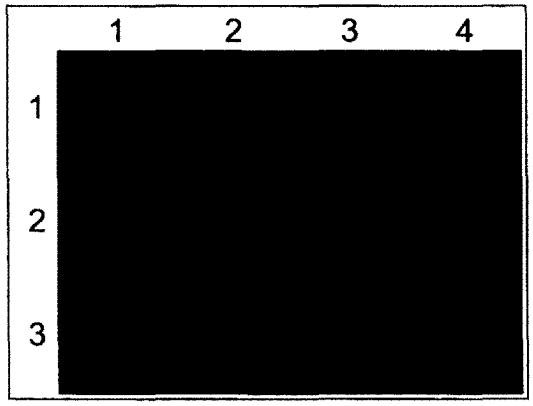


Figure 4.3 Format *column-major order.*

4.4.2 Gestion de la mémoire physique

Le code source de ce travail est protégé contre les débordements et le coulage de mémoire, ainsi qu'à l'accès aux zones interdites. Cette gestion adéquate de la mémoire se confirme par la validation de l'algorithme à l'aide du logiciel PURIFY©[4] qui permet de détecter ces types d'erreurs de programmation.

---

[4] http://www-01.ibm.com/software/awdtools/purify/

4.4.3 Portabilité

La portabilité de l'outil logiciel est l'un des aspects les plus importants du développement de l'algorithme. La propriété multiplateforme du programme est assurée par l'utilisation de code source libre et le respect des normes C ANSI dans l'ensemble des bibliothèques. Le programme peut ainsi être compilé pour des plateformes Windows, Linux ou même Sun à l'aide d'un simple compilateur *Gnu C Compiler* (GCC) standard. Les droits de distribution sont respectés par la présence des informations de droits d'auteur dans l'en-tête des fichiers concernés. Le programme fourni avec ce document est développé afin d'être compilé pour une architecture de 32-bit, mais peut être ajusté ultérieurement pour une optimisation à 64-bit.

4.5 Conclusion

Les caractéristiques du produit logiciel présenté dans ce chapitre en font un outil d'analyse modale polyvalent et applicable dans un maximum d'environnements logiciels. Premièrement, sa propriété de portabilité lui permet d'être affranchi de tous logiciels, services ou mises à jour externes qui pourraient limiter sa distribution à grande échelle. Deuxièmement, la quantité et la complexité des arguments d'entrée son largement compensées par les différents niveaux de contrôle qu'ils procurent. Bien sûr, un certain niveau de connaissance sur la stabilité dynamique et le traitement de signal est attendu de tout utilisateur potentiel. Il est tout de même fortement suggéré de garantir la présence d'un document de référence, disponible à partir du logiciel incorporant le produit.

CHAPITRE 5

VALIDATION DE L'ALGORITHME

5.1 Introduction

Ce chapitre présente trois études de cas d'analyse modale dans le but d'évaluer le produit logiciel mis au point à partir d'un algorithme ERA/Prony. Les études sont une représentation des cas les plus fréquemment rencontrés en analyse de signal numérique. La première analyse modale suggère l'utilisation de l'algorithme sur des signaux conçus à partir de fonctions mathématiques simples. Tous les paramètres recherchés dans cette étude sont déterminés à l'avance afin d'estimer le degré de précision exacte de l'algorithme. La deuxième étude est une simulation de phénomènes oscillatoires sur un réseau haute-tension, ce qui augmente la complexité de l'analyse et ajoute une dimension de cause à effet aux paramètres obtenus. Finalement, la troisième traite d'un échantillonnage provenant de lectures réelles du réseau d'Hydro-Québec pour démontrer la pertinence de l'algorithme dans des conditions d'analyse plus sévères. Pour les trois études de cas, les résultats sont présentés sous la forme de tableaux et de graphiques obtenus d'un programme de traitement de signal intégrant le produit logiciel présenté au chapitre précédent

5.2 Contexte ScopeView©

Le logiciel d'acquisition de signal ScopeView© est utilisé à titre de programme hôte dans le cadre du processus de validation de l'algorithme ERA/Prony. Ce logiciel, développé par Hydro-Québec[5], agit en tant qu'outil de traitement mathématique et graphique des signaux numériques. Le logiciel est optimisé pour traiter l'information d'une multitude de sources et peut être compilé sur la plupart des plateformes informatiques connues. L'importation de l'algorithme ERA/Prony dans le code source de ScopeView© fait apparaître l'élément

---

[5] Tous droits réservés © 2001-2007, Hydro-Québec.

« amod » à la liste des fonctions mathématiques du logiciel. Pour réaliser les études proposées dans ce chapitre, certaines fonctions auxiliaires de ScopeView© sont appelées en plus de « amod ». Voici un résumé des fonctions utilisées :

1. Fonction « *amod(sig, f_low, f_high, alim, zlim, trig, man_order, filter)* » : Retourne les résultats de l'analyse modale du signal *sig*. L'argument *sig* contient le signal et la période d'échantillonnage.

2. Fonction « *fft(sig)* » : Retourne le spectre de la transformée de Fourier rapide (FFT), du signal *sig*.

3. Fonction « *ramp(tstart, tend, per, slope)* » : Retourne un vecteur d'échantillons qui s'étendent de *tstart* à *tend*, selon un période *per* et une pente *slope*.

4. Fonction « *lim(sig, xstart, xend)* » : Retourne le contenu du signal *sig*, limité entre les points *xstart* et *xend* de l'axe des abscisses.

5. Fonction « *xf(sig)* » : Retourne la dernière valeur de l'axe des abscisses du signal *sig*.

6. Fonction « *xshift(sig,val)* » : Retourne le contenu du signal *sig*, déplacé à la valeur *val* sur l'axe des abscisses.

5.3 Étude de cas 1 : Présentation théorique

La première étude présente l'analyse modale d'un signal oscillatoire, dont le contenu fréquentiel est préalablement connu. Cette approche théorique permet d'évaluer les performances de l'algorithme sur des signaux de faible complexité. De plus, la présence de modes ayant un faible écart de fréquence démontre les avantages que peut présenter la méthode par rapport aux résultats d'une analyse FFT standard.

5.3.1 Description de l'étude

Le signal oscillatoire soumis à l'analyse se compose de trois modes fréquentiels amortis, additionnés à une fondamentale de 60 Hz. Les paramètres recherchés et les formes d'onde associées à ces quatre modes sont résumés dans le tableau et la figure qui suivent :

Tableau 5.1 Paramètres de Prony recherchés de l'étude 1

Mode	**Amplitude** a_k	**Amortissement** σ_k	**Fréquence** f_k	**Déphasage** θ_k
1	1,00	0,00	60,00 Hz	0°
2	1,00	-10,00	30,00 Hz	0°
3	1,00	-5,00	10,50 Hz	0°
4	1,00	-5,00	10,00 Hz	0°

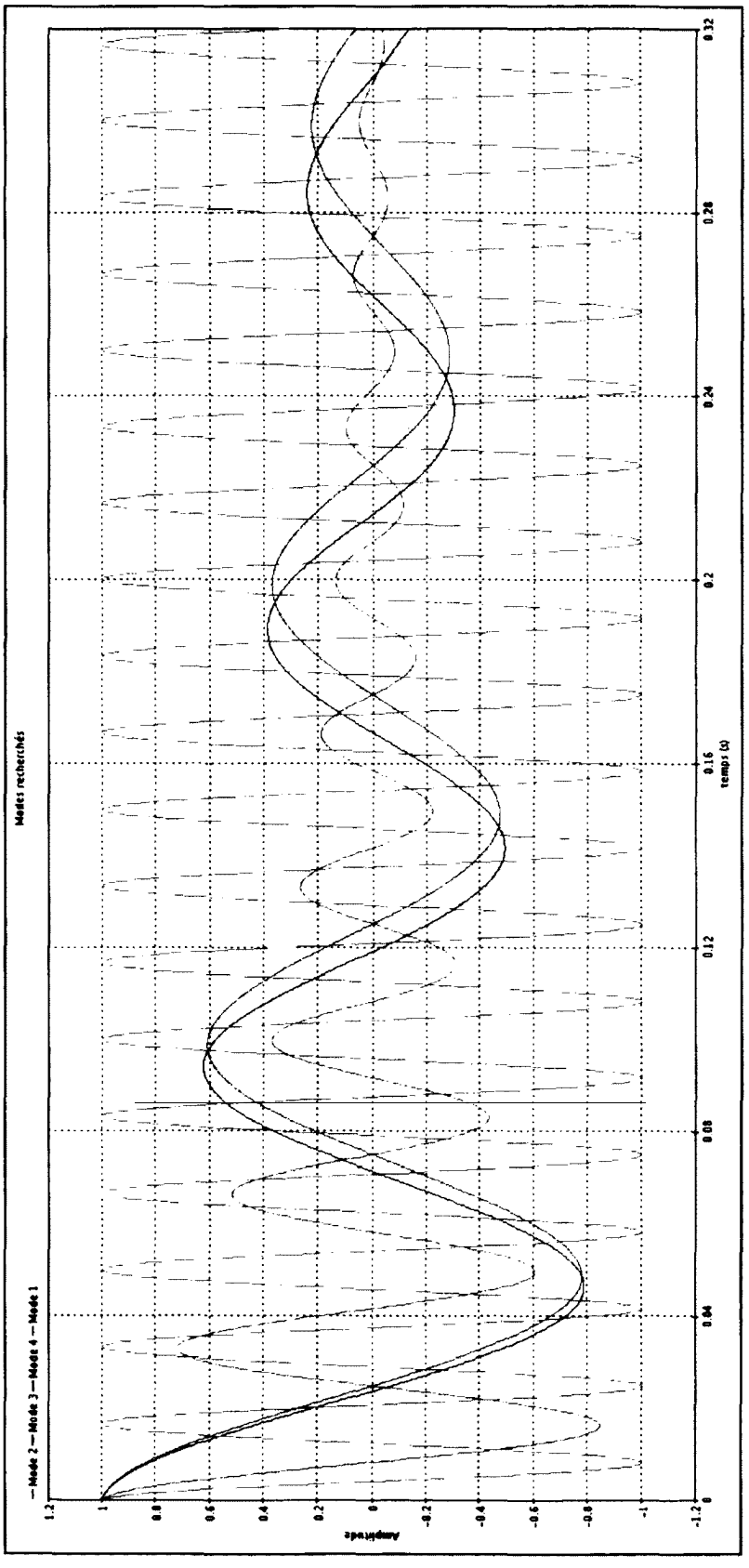


Figure 5.1 Formes d'onde des modes recherchés de l'étude 1.

L'analyse modale est effectuée sur un signal théorique, réalisé à l'aide des fonctions « ramp », « exp » et « cos » de ScopeView©. L'algorithme ERA/Prony est appelé par la version ScopeView© de la fonction « amod ». Les arguments sont choisis pour cadrer avec les données de l'étude et l'ordre du modèle est manuellement ajusté après un essai en mode automatique. L'ensemble des fonctions et des arguments saisis pour cette étude se regroupent dans la fenêtre de sélection des signaux de ScopeView© :

	t=ramp(0,3,1e-4,1)	Base temporel d'échantillons	☐
	a0 = cos(2*pi*60*t)	Mode 1	☐
	a1=exp(-10*t)*cos(2*pi*30*t)	Mode 2	☐
	a2=exp(-5*t)*cos(2*pi*10.5*t)	Mode 3	☐
	a3=exp(-5*t)*cos(2*pi*10*t)	Mode 4	☐
	sig=a0+a1+a2+a3	Signal oscillatoire théorique	☑
	trig = 0.0		☐
	sres=amod(sig,10,61,100,0.707,trig,8,0)	Appel de l'algorithme ERA/Prony	☐
	xshift(sres,trig)	Signal reconstruit	☑
	fft(sig)	FFT du signal échantillonné	☑
	fft(sres)	FFT du signal reconstruit	☑
			☐

Figure 5.2 Fenêtre d'appel ScopeView© de l'étude 1.

5.3.2 Résultats

Le tableau et la figure qui suivent résument les résultats obtenus de l'analyse modale. Le tableau liste les paramètres de Prony, tandis que la figure compare les formes d'onde et les FFT de chacun des signaux :

Tableau 5.2 Paramètres de Prony de l'analyse modale de l'étude 1

Mode	Amplitude a_k	Amortissement σ_k	Fréquence f_k	Déphasage θ_k
1	1,000	0,00	59,99 Hz	-1,08°
2	1,005	-10,00	30,00 Hz	-0,53°
3	1,005	-5,00	10,50 Hz	-0,18°
4	1,005	-5,00	10,00 Hz	-0,19°

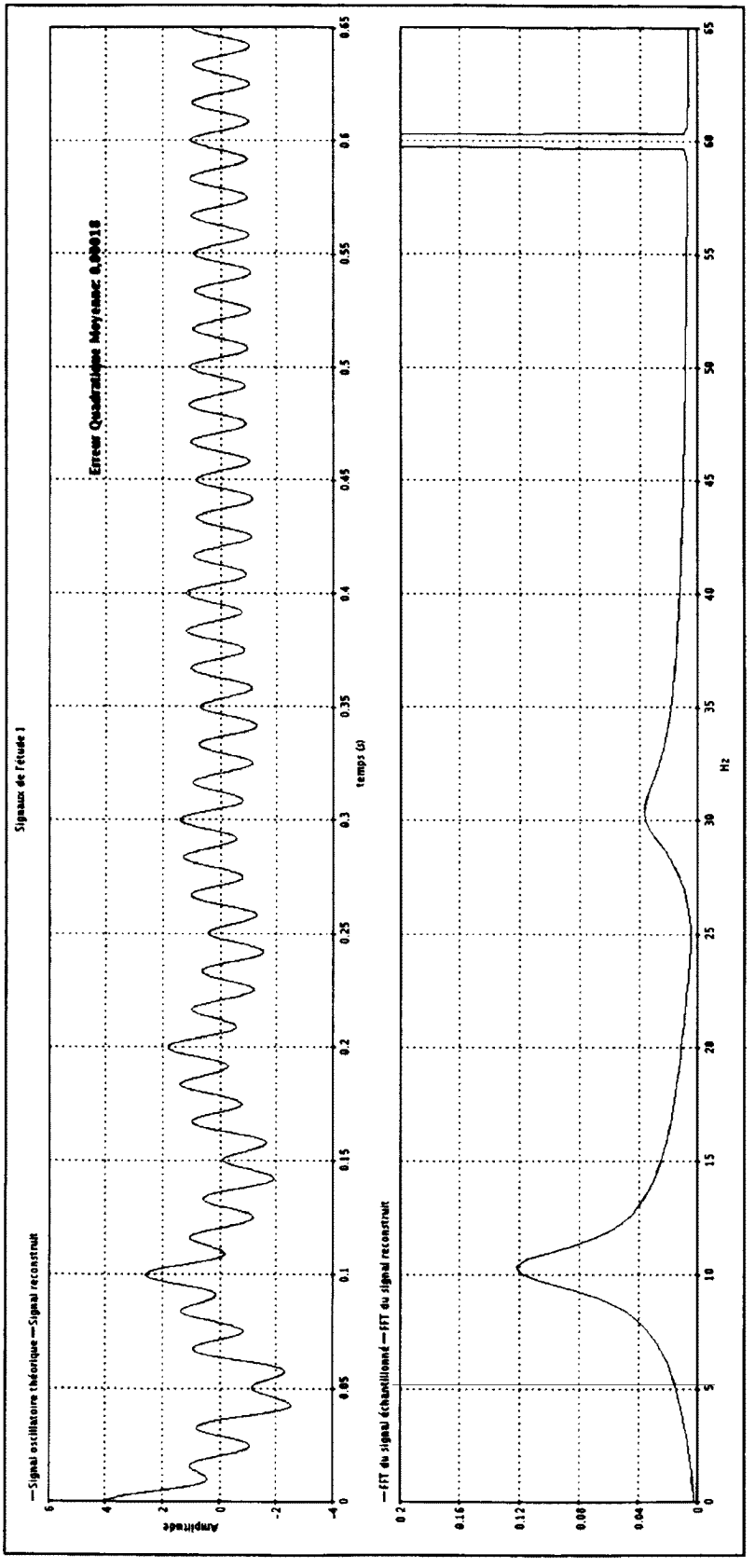


Figure 5.3 Signaux et FFT de l'étude 1.

5.3.3 Discussion

À la lumière des résultats obtenus, il est évident que la marge d'erreur des données produites par l'algorithme est négligeable. D'abord, les paramètres de Prony sont quasiment identiques à ceux recherchés, alors que les plus grands écarts se limitent à une erreur moyenne de 0,5° sur le déphasage. La comparaison graphique entre les signaux et leur FFT porte à la même conclusion, car les différences entre les courbes sont pratiquement inexistantes comme le prouve l'erreur quadratique moyenne (EQM) d'amplitude de 0,00018.

De plus, il est intéressant de noter que sur le graphique des transformées de Fourier, la forme du spectre laisse supposer la présence d'un mode à environ 10,25 Hz, alors que dans les faits, il s'agit de la sommation de deux modes à 10 et 10,5 Hz. Cet exemple illustre parfaitement la nature qualitative de la transformée de Fourier, comparativement à l'approche quantitative, par des valeurs précises, de l'analyse modale. Une transformée de Fourier effectuée sur le même signal avec un échantillonnage plus fin pourrait éventuellement améliorer la résolution du spectre, mais cette idée se limite à des signaux de simulation, ou entièrement modifiables.

D'après ces diverses observations, l'étude démontre clairement l'efficacité et l'utilité de l'algorithme pour une situation théorique contrôlée.

5.4 Étude de cas 2 : Réseau à quatre machines de Kundur

La deuxième étude concerne l'analyse modale des signaux synthétiques provenant de simulations sur le réseau à quatre machines de Kundur. L'intérêt pour ce réseau provient de sa simplicité, de son abondante publication et de ses conditions favorables aux études sur les oscillations à basse fréquence. Les simulations sont réalisées sur le logiciel EMTP-RV[©6].

---

[6] Tous droits réservés © 2003-2008, CEATI International.

5.4.1 Réseau

Le réseau étudié est symétrique et relié par deux lignes à 230 kV de 220 km de longueur. Les lignes sont modélisées selon une configuration π. Chaque région est équipée de deux machines synchrones de 900 MVA / 20 kV. Les machines ont les mêmes paramètres, excepté pour une inertie de H=6,5s dans la zone 1 et de H=6,75s dans la zone 2. Les charges présentes sont modélisées comme impédance constante et réparties de façon à ce que la zone 1 exporte 400 MW dans la zone 2. Le réseau produit deux modes d'oscillation locaux et un mode d'oscillation interzone. Il est représenté par le schéma suivant :

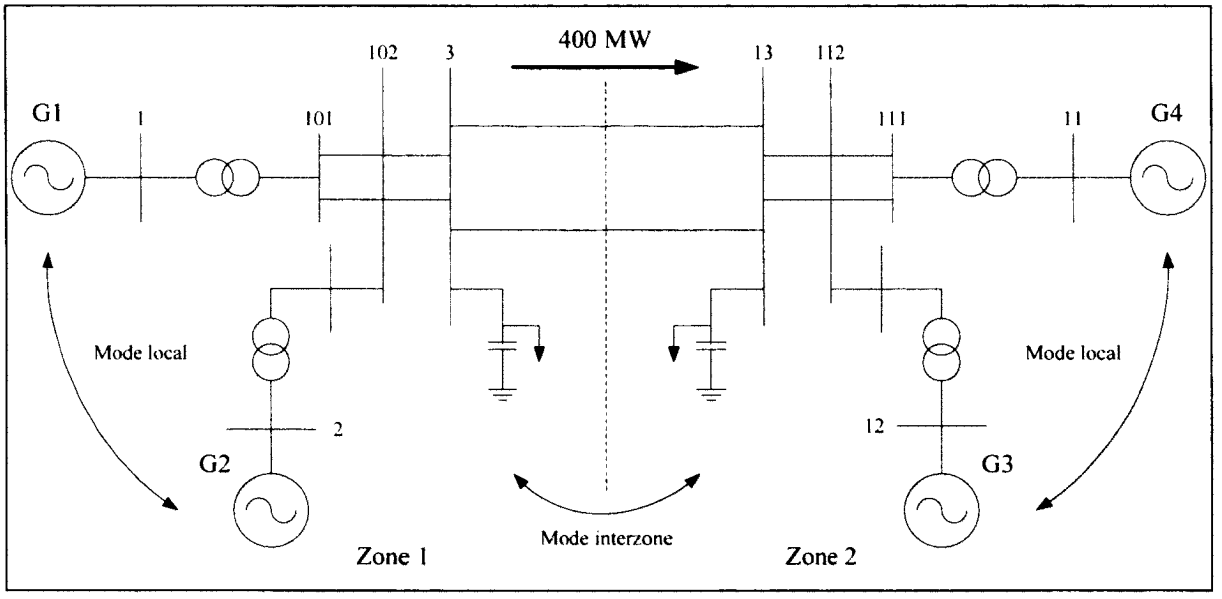


Figure 5.4 Réseau à quatre machines de Kundur.
Adaptée de Kamwa *et al.* (1997, p. 50)

5.4.2 Description de l'étude

L'étude propose l'analyse modale des oscillations du réseau, suite à une perturbation temporaire de la tension de champ e_f sur la machine G1. La perturbation est simulée au temps $t = 2$ secondes par un échelon d'une durée de 0,5 seconde. L'analyse compare le comportement du torque électrodynamique τ_{eg} aux machines G1 et G3. L'étude est séparée

en deux séries de simulations qui comportent différents systèmes d'excitation du champ. La première série est effectuée avec un contrôle par *Thyristor exciter with a transient Gain Reduction* (TGR), tandis qu'un *Power System Stabilizer* (PSS) est ajouté à la deuxième. Le but de cette étude parallèle est de démontrer l'utilité de l'algorithme dans l'identification des modes locaux et interzones, pour des situations de stabilité dynamique différentes.

Pour chacune des simulations, l'analyse modale est réalisée sur les données générées par EMTP-RV©. L'algorithme ERA/Prony est appelé par la version ScopeView© de la fonction « amod ». Les arguments sont ajustés pour une étude à basse fréquence de l'ordre de 0,05 à 3 Hz. Seuls les modes possédant un amortissement relatif inférieur à 0,25 sont retenus à l'affichage et les ordres de modèles sont ajustés individuellement afin de produire les formes d'onde les plus conformes possible. L'ensemble des fonctions et des arguments saisis pour cette étude se regroupent dans les fenêtres de sélection des signaux de ScopeView© :

	sig=Teg_G1_2	Signal de la simulation
	trig=2.5	
	sres=amod(sig,0.05,3,100,.25,trig,7,0)	Appel de l'algorithme ERA/Prony
	yshift=xshift(sres,trig)+0.779467	Signal reconstruit
	fft(lim(sig,trig,xf(sig)))	FFT du signal de la simulation
	fft(sres)	FFT du signal reconstruit

Figure 5.5 Fenêtre d'appel ScopeView© de la simulation TGR zone 1.

	sig=Teg_G3_1	Signal de la simulation
	trig=2.5	
	sres=amod(sig,0.05,3,100,.25,trig,7,0)	Appel de l'algorithme ERA/Prony
	xshift(sres,trig)+0.800394	Signal reconstruit
	fft(lim(sig,trig,xf(sig)))	FFT du signal de la simulation
	fft(sres)	FFT du signal reconstruit

Figure 5.6 Fenêtre d'appel ScopeView© de la simulation TGR zone 2.

sig=Teg_G1_3	Signal de la simulation
trig=2.5	
sres=amod(sig,0.05,3,100,.25,trig,27,0)	Appel de l'algorithme ERA/Prony
yshft=xshift(sres,trig)+0.779443	Signal reconstruit
fft(lim(sig,trig,xf(sig)))	FFT du signal de la simulation
fft(sres)	FFT du signal reconstruit

Figure 5.7 Fenêtre d'appel ScopeView© de la simulation PSS zone 1.

sig=Teg_G3	Signal de la simulation
trig=2.5	
sres=amod(sig,0.05,3,100,.25,trig,7,0)	Appel de l'algorithme ERA/Prony
yshft=xshift(sres,trig)+0.800372	Signal reconstruit
fft(lim(sig,trig,xf(sig)))	FFT du signal de la simulation
fft(sres)	FFT du signal reconstruit

Figure 5.8 Fenêtre d'appel ScopeView© de la simulation PSS zone 2.

5.4.3 Résultats

Simulations avec TGR

Les tableaux et les figures qui suivent résument les résultats obtenus de l'analyse modale sur les simulations avec contrôle du champ par TGR. Les tableaux listent les paramètres de Prony, tandis que les figures comparent les formes d'onde et les FFT de chacun des signaux :

Tableau 5.3 Paramètres de Prony de la simulation TGR zone 1

Mode	Amplitude a_k	Fréquence f_k	Déphasage θ_k	Amortissement ζ_k
1	0,0042	1,043 Hz	94,93°	0,097
2	0,0010	0,539 Hz	3,58°	-0,004

Tableau 5.4 Paramètres de Prony de la simulation TGR zone 2

Mode	**Amplitude** a_k	**Fréquence** f_k	**Déphasage** θ_k	**Amortissement** ζ_k
1	0,0058	0,539 Hz	-90,06°	-0,004
2	0,0005	1,057 Hz	92,70°	0,094
3	0,0001	0,787 Hz	-82,81°	0,176

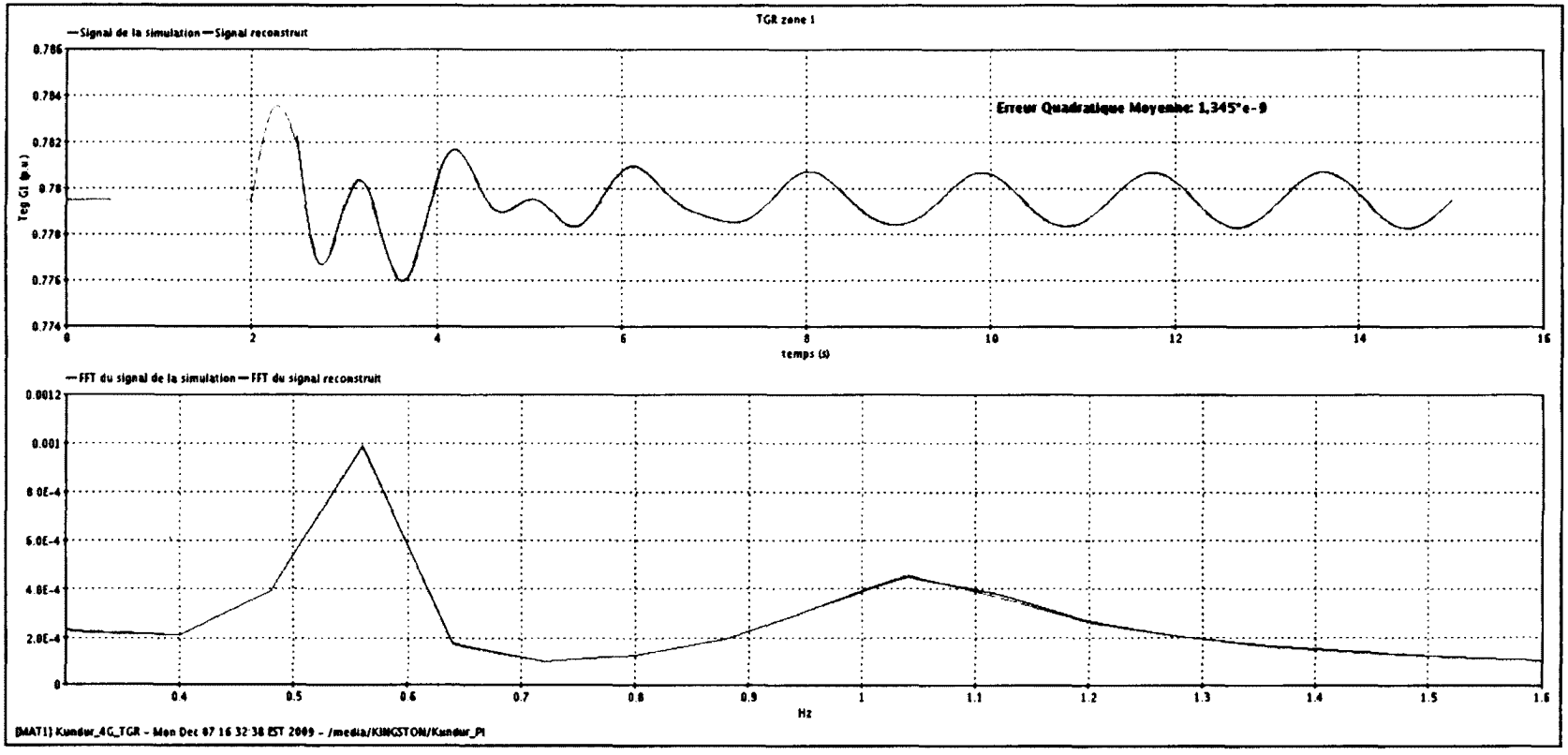


Figure 5.9 Signaux et FFT de la simulation TGR zone 1.

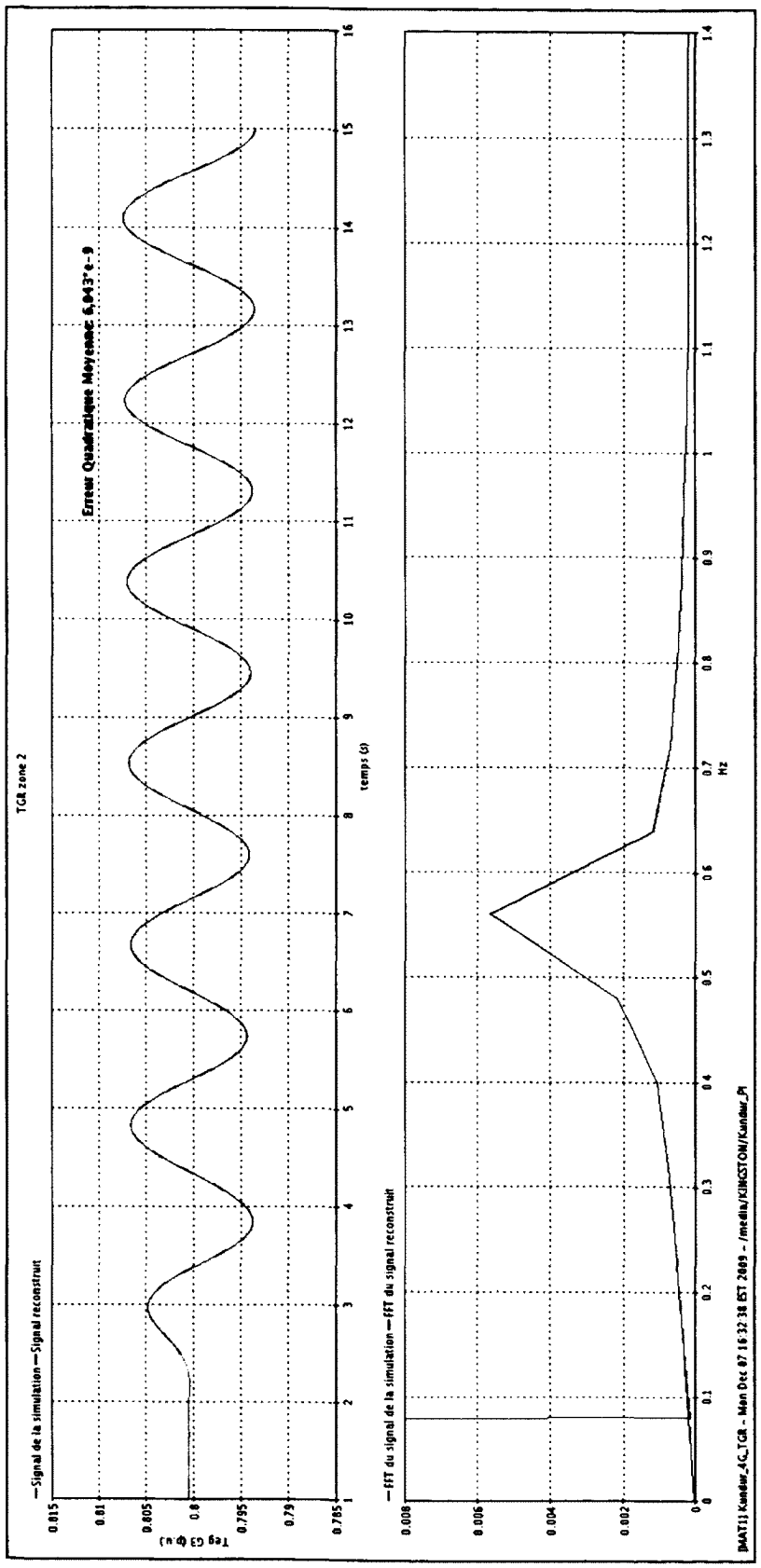


Figure 5.10 Signaux et FFT de la simulation TGR zone 2.

Simulations avec PSS

Les tableaux et les figures qui suivent résument les résultats obtenus de l'analyse modale sur les simulations par contrôle du champ avec PSS. Les tableaux listent les paramètres de Prony, tandis que les figures comparent les formes d'onde et les FFT de chacun des signaux :

Tableau 5.5 Paramètres de Prony de la simulation PSS zone 1

Mode	**Amplitude a_k**	**Fréquence f_k**	**Déphasage θ_k**	**Amortissement ζ_k**
1	0,0040	1,084 Hz	97,91°	0,1
2	0,0012	0,593 Hz	-27,53°	0,039

Tableau 5.6 Paramètres de Prony de la simulation PSS zone 2

Mode	**Amplitude a_k**	**Fréquence f_k**	**Déphasage θ_k**	**Amortissement ζ_k**
1	0,0060	0,593 Hz	-91,60°	0,036
2	0,0004	1,088 Hz	97,54°	0,076

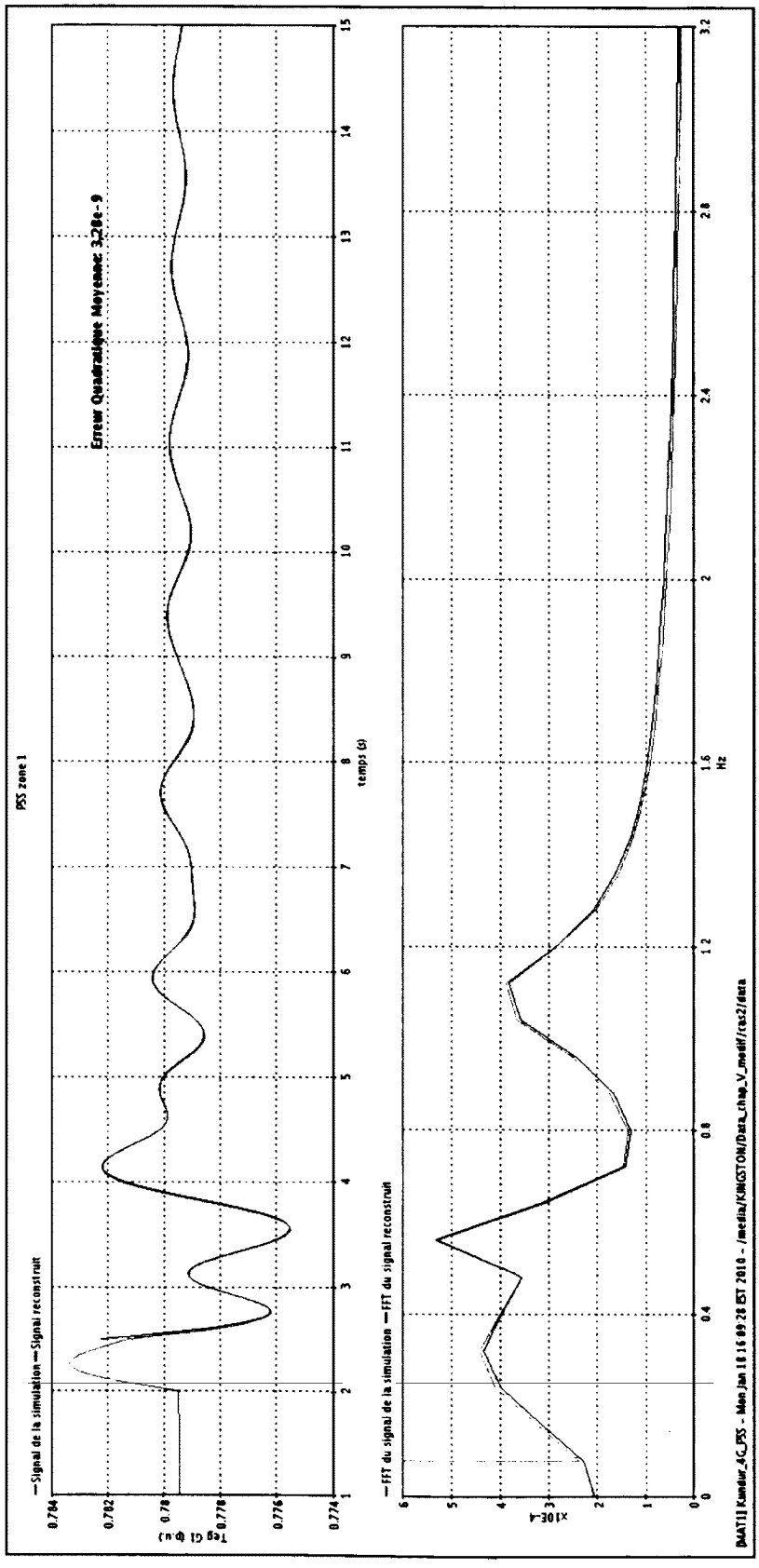


Figure 5.11 Signaux et FFT de la simulation PSS zone 1.

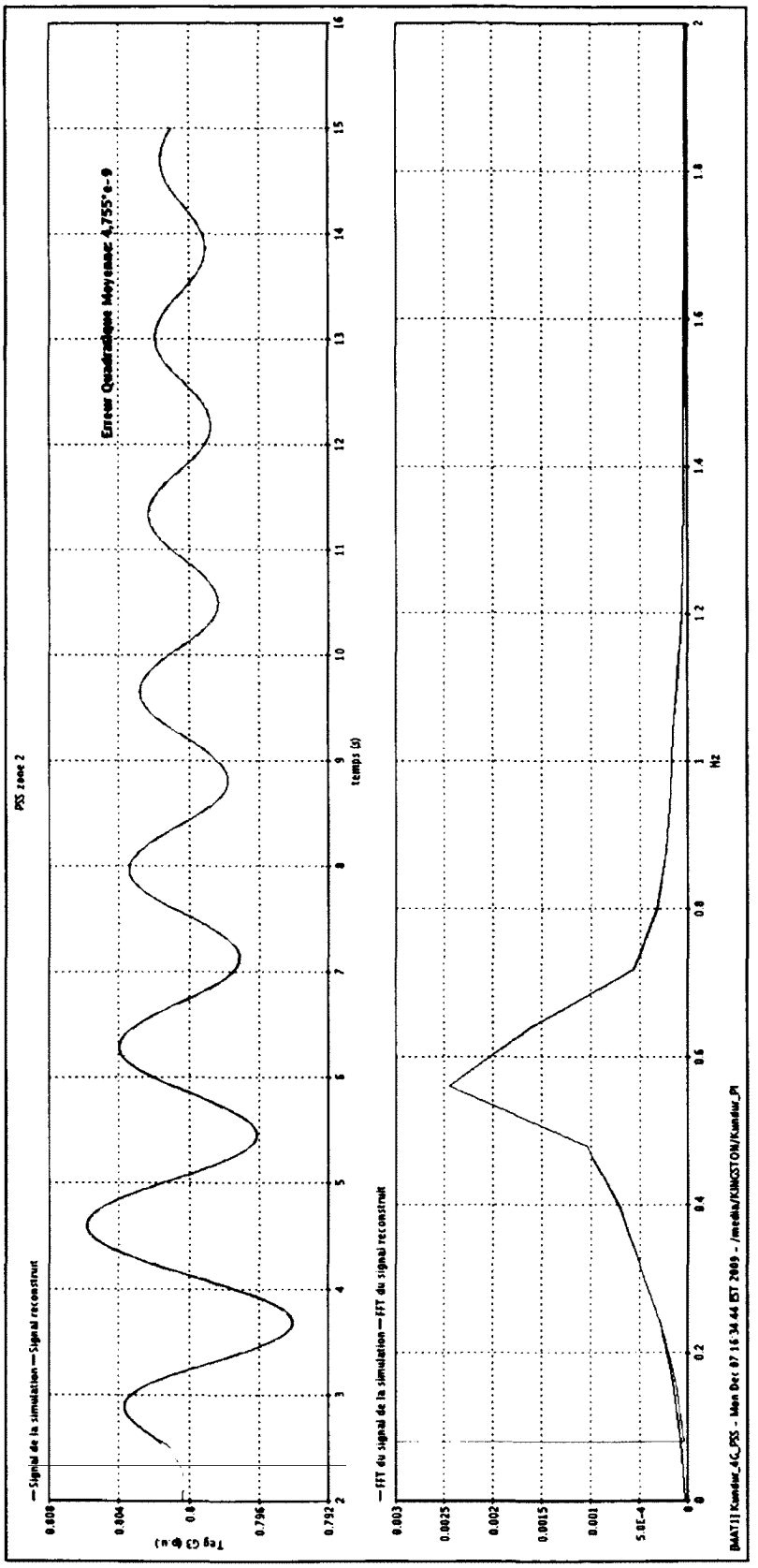


Figure 5.12 Signaux et FFT de la simulation PSS zone 2.

5.4.4 Discussion

Pour chacune des simulations, la comparaison des formes d'onde produit des EQM d'ordre 10^{-9} p.u. et des ressemblances notables entre les FFT, ce qui confirme l'exactitude des paramètres de Prony. Les deux simulations produisent des valeurs différentes pour des conditions de stabilité opposées, comme le prouve l'évolution de leurs graphiques respectifs.

D'une part, la simulation avec TGR permet de remarquer la présence d'un mode instable à 0,593 Hz dans les deux zones. Il s'agit du mode interzone provenant de l'effet de balancement entre les groupes (G1-G2) et (G3-G4). Par ailleurs, l'effet sur la zone 2 est beaucoup plus marqué, alors que l'amplitude initiale du mode interzone est près de six fois supérieure à celle de la zone 1. Pour les modes locaux, on remarque plus facilement le comportement du mode de la zone 1 à 1,043 Hz, car son amplitude initiale est près de quatre fois plus grande que celle du mode interzone. En comparaison, l'amplitude initiale du mode local de la zone 2, à 1,057 Hz, est dix fois inférieure à celle du mode interzone. Par contre, les deux cas de mode local présentent des comportements amortis. C'est donc dire que la perturbation engendre un comportement globalement instable, plus important dans la zone 2 et un comportement local stable, mais beaucoup plus distinct entre les machines G1 et G2.

De la même façon que pour l'étude théorique, les FFT ne précisent pas tous les phénomènes impliqués, car le graphique de la zone 2 semble, a priori, ne contenir qu'un seul mode.

D'autre part, la simulation avec PSS produit le même mode interzone de 0,593 Hz, mais avec un amortissement relatif positif. Ce comportement correspond aux prévisions de Kundur *et al.* (1994, p. 816) sur l'étude de ce réseau « les modes interzones et locaux sont très bien amortis lorsque les PSS sont ajoutés aux TGR ». L'effet du mode interzone sur la zone 2 est encore une fois plus prononcé, de même que la présence du mode local sur la zone 1.

Cette étude démontre donc l'efficacité et l'utilité de l'algorithme sur des signaux synthétiques provenant de simulations.

5.5 Étude de cas 3 : Perte de synchronisation LG4-Boucherville

La dernière analyse s'effectue sur un signal réel, provenant d'une lecture sur le réseau à 735 kV d'Hydro-Québec. L'utilisation de l'algorithme dans un contexte pratique, d'une plus grande complexité, constitue l'intérêt premier de cette étude. La lecture est obtenue grâce au système de mesure du décalage angulaire (SMDA) installé sur le réseau d'Hydro-Québec.

5.5.1 SMDA

Le SMDA est une solution de mesure à grande échelle mise en place par Hydro-Québec, afin de permettre une supervision centralisée du comportement dynamique de l'ensemble de son réseau. La version actuelle du SMDA est conçue à l'aide de huit appareils de mesure programmables (PMU), répartis aux points clés du réseau à 735 kV. Ces PMU sont reliés par l'entremise d'un système GPS qui permet de synchroniser les données sur l'angle, la fréquence et les distorsions harmoniques avec un délai inférieur à 1 μs. Ces données sont acheminées au système de *supervisory control and data acquisition* (SCADA) du centre de contrôle à un taux d'échantillonnage de 60 Hz. Arrivées au centre de contrôle, les données sont corrigées pour les erreurs de transmission GPS et une reconstitution temporelle adéquate, avant d'être finalement sauvegardées comme lecture. La configuration actuelle du SMDA est représentée par le graphique suivant :

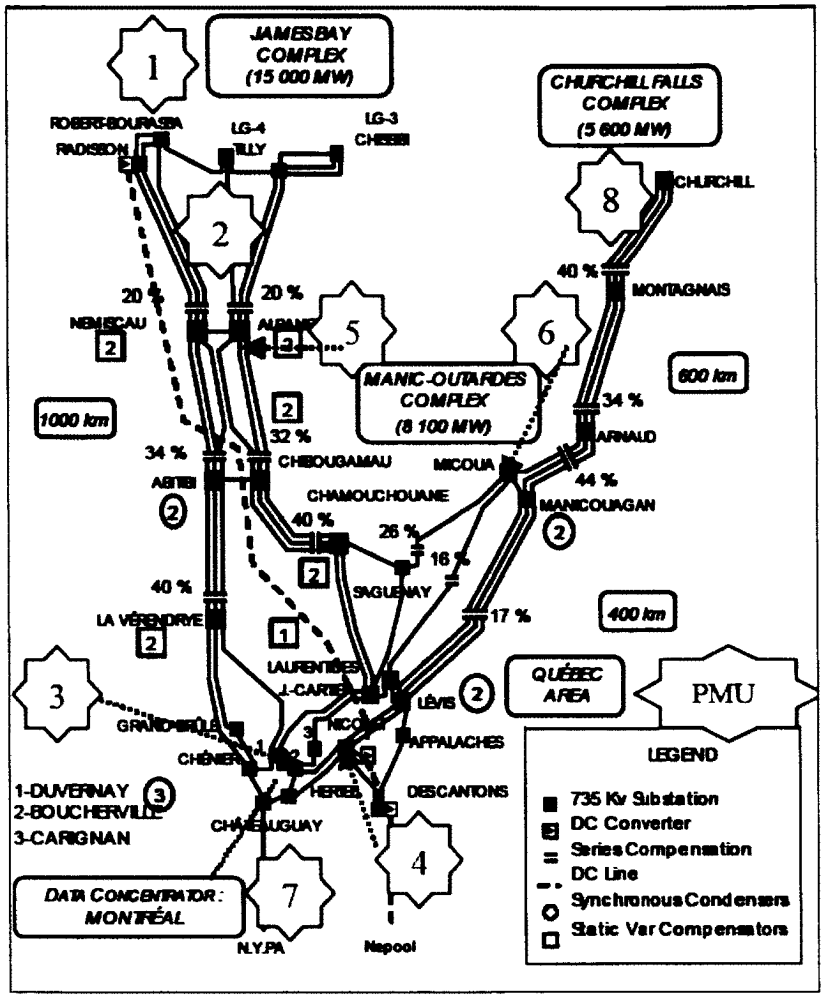


Figure 5.13 Configuration du SMDA d'Hydro-Québec.
Tirée de Kamwa *et al.* (2006, p. 2)

5.5.2 Description de l'étude

La lecture en question décrit l'oscillation d'angle relatif au poste Boucherville suite à une perte de production de la centrale LG4. L'analyse modale est réalisée sur les données du SMDA. L'algorithme ERA/Prony est appelé par la version ScopeView© de la fonction « amod ». La forme de l'onde étudiée et une étude préliminaire avec filtrage complet suggèrent de meilleurs résultats en fixant l'argument *filter* à 1, ce qui inclut les modes de toutes fréquences et ceux dont l'amortissement presque nul a une influence sur la correction CC du graphique reconstruit. Seuls les modes possédant un amortissement relatif inférieur à 0,25 sont retenus pour l'affichage et l'ordre du modèle est ajusté manuellement pour produire

les formes d'onde les plus conformes possible. L'ensemble des fonctions et des arguments saisis pour cette étude se regroupent dans la fenêtre de sélection des signaux de ScopeView© :

	sig=_signal_1	Signal du SMDA
	trig = 4.79	
	sres = amod(sig,0.05,10,100,0.25,trig,27,1)	Appel de l'algorithme ERA/Prony
	y=xshift(sres,trig)	Signal reconstruit
	fft(lim(sig,trig,xf(sig)))	FFT du signal du SMDA
	fft(y)	FFT du signal reconstruit

Figure 5.14 Fenêtre d'appel ScopeView© de l'étude du SMDA.

5.5.3 Résultats

Le tableau et la figure qui suivent résument les résultats obtenus de l'analyse modale sur la lecture du SMDA. Le tableau liste les paramètres de Prony, tandis que la figure compare les formes d'onde et les FFT de chacun des signaux :

Tableau 5.7 Paramètres de Prony de l'étude du SMDA

Mode	Amplitude a_k	Fréquence f_k	Déphasage θ_k	Amortissement ζ_k
1	1,277	0,681 Hz	-57,62°	0,200
2	0,286	0,870 Hz	-178,26°	0,047
3	0,180	1,138 Hz	-59,10°	0,040
4	0,173	1,423 Hz	48,78°	0,061
5	0,033	0,253 Hz	-155,13°	0,053

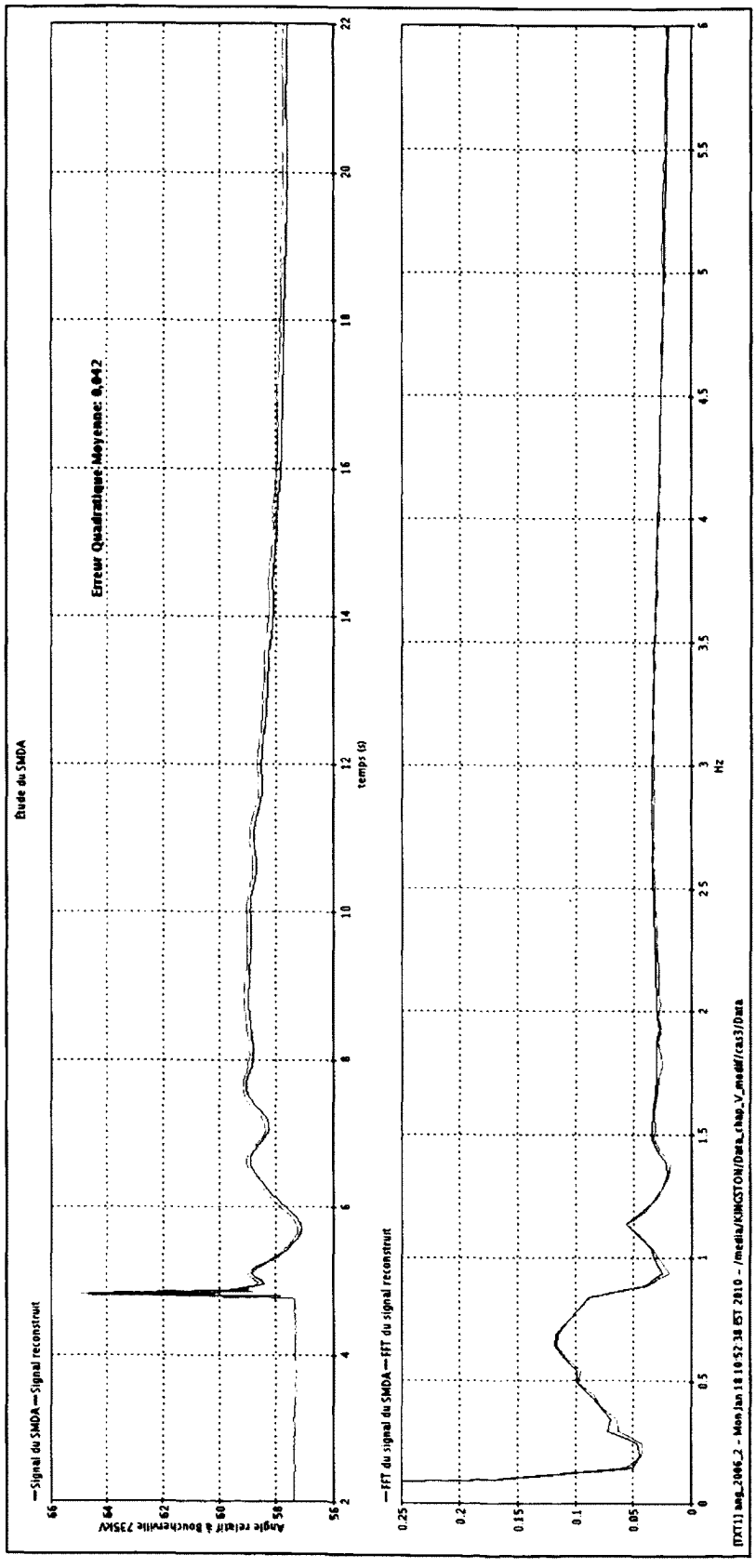


Figure 5.15 Signaux et FFT de l'étude du SMDA.

5.5.4 Discussion

Malgré le riche contenu dynamique du signal analysé, la comparaison des résultats témoigne de données paramétriques représentatives de la réalité, alors que les formes d'ondes produisent une EQM de 0,042° et que les FFT démontrent de faibles écarts relatifs. Cependant, ces données n'expliquent pas à elles seules les causes ou les effets des modes observés. Donc, lorsque les circonstances connues d'une étude sont limitées, l'analyse modale devient plutôt un outil d'hypothèses préliminaires à une recherche plus approfondie.

Par exemple, les modes à surveiller peuvent être identifiés avec les paramètres d'amortissement relatif et d'amplitude. C'est le cas des modes 2 à 4 qui possèdent tous une amplitude substantielle, reliée à un comportement oscillatoire faiblement amorti. Bien que le premier mode retenu possède une amplitude initiale beaucoup plus élevée que les quatre autres modes, son amortissement de 20 % trahit sa faible implication à long terme sur la forme d'onde. Cette dernière conclusion pourrait cependant être bien différente si la FFT du signal agissait à titre de seul élément d'observation, car le mode à 0,681 Hz semble être dominant sur ce graphique.

D'après ces diverses observations, on peut affirmer que cette d'étude démontre une efficacité concluante de l'algorithme sur des signaux de lectures réelles, échantillonnées à des taux beaucoup moins élevés que ceux d'études théoriques.

5.6 Conclusion

Les résultats obtenus dans ce chapitre mènent à la conclusion que l'algorithme ERA/Prony, associé à un programme hôte adéquat, se révèle un produit d'analyse modale efficace et performant sous plusieurs conditions d'utilisation. Dans certains cas, il supplante presque l'utilisation de la FFT qui devient plutôt un procédé de validation qu'un outil d'analyse en soi.

CONCLUSION

Ce travail de recherche a permis de répondre au besoin de développement d'un outil portable d'analyse modale par réponse temporelle. En effet, le produit logiciel réalisé permet de quantifier chacune des propriétés dynamiques d'un signal oscillatoire non linéaire. De plus, l'outil utilise ces propriétés afin de reproduire une forme d'onde temporelle pour valider graphiquement les résultats obtenus. L'algorithme de l'outil est programmé en langage C et produit les résultats sous forme d'une structure de donnée. Le contenu du rapport de recherche énonce les solutions mathématiques et logicielles menant au produit final.

En premier lieu, il est suggéré de traiter le signal à analyser comme une réponse à l'impulsion d'un système inconnu. Ce système est ensuite identifié sous la forme d'un modèle d'état par une méthode de réalisation ERA. La démarche justifie la permutation des paramètres de Markov de la matrice de Hankel par les échantillons du signal analysé. Elle établit ensuite les relations mathématiques avec les matrices d'état du domaine continu pour arriver au modèle d'état recherché. L'estimation des paramètres est obtenue à l'aide de l'étude des valeurs propres du modèle d'état et du calcul de ses éléments résiduels, d'après une méthodologie en lien avec l'approche par paramètres de Prony.

Par la suite, le fonctionnement du produit logiciel est fondé sur une solution d'architecture organisée en trois bibliothèques logicielles composées de fichiers et de sous-programmes. Chaque méthode est associée à un fichier spécifique alors que les sous-programmes représentent des solutions LAPACK© reliées aux problèmes d'algèbre linéaire. Du point de vue du fonctionnement, l'exécution de l'algorithme est entièrement contrôlée par l'appel d'une fonction principale. La fonction reçoit les informations sur le signal et les conditions d'analyse fixées par l'utilisateur. Elle retourne les paramètres de Prony et le signal reconstitué sous la forme de vecteurs d'éléments réels. Cet échange d'informations avec l'utilisateur s'effectue par l'entremise d'un programme hôte, en fonction d'une interface utilisateur propre à celui-ci.

La dernière étape démontre l'étendue des applications du produit logiciel réalisé dans cette recherche. Le premier cas réalise la décomposition d'un signal oscillatoire simple, conçue avec des variables connues qui assurent une validation exacte. La deuxième étude prouve la capacité de l'algorithme à analyser le contenu modal de signaux complexes, provenant de simulations sur un réseau électrique. Les résultats obtenus sur les phénomènes interzones confirment l'avantage de la méthode ERA/Prony sur les méthodes sélectives. Le dernier cas confirme la possibilité d'analyser le contenu dynamique d'un signal qui provient de mesures échantillonnées à chaque cycle sur le réseau d'Hydro-Québec.

Malgré les qualités de robustesse et de polyvalence démontrées par l'algorithme ERA/Prony, le produit logiciel peut toujours être amélioré et optimisé pour certaines applications plus précises. Voici quelques recommandations qui vont en ce sens :

- Pour une importation dans un environnement de calcul puissant, il est suggéré d'utiliser l'argument *mod_order* de la fonction principale pour comparer le signal reconstruit et le signal d'origine avec une méthode des moindres carrés. Cette solution permet de déterminer l'ordre optimal du modèle de façon itérative, sans modifier le code source d'origine.

- Lardies (1998, p. 543-558) propose une estimation directe de l'ordre optimal à l'aide d'une évaluation asymptotique de l'implication de chacun des états d'une matrice d'observabilité $\widehat{\mathbb{Q}}_p\Big|_{p\gg n}$ et de sa matrice de covariance $\mathrm{cov}[vec(\widehat{\mathbb{Q}}_p)]$. Cette technique implique de modifier le code source du fichier « era.c » pour y inclure les méthodes d'inéquation de Chebyschev et du produit de Kronecker.

- Suivant les travaux de Kamwa *et al.* (2000, p. 326-334), il est possible de perfectionner l'algorithme pour une approche système, lorsque les entrées et les sorties sont connues. Cette technique MIMO évalue la gouvernabilité et l'observabilité de chaque mode oscillatoire, à chacune des entrées et des sorties du système. Elle s'appuie sur un calcul des facteurs de participation à l'aide de la matrice

des résidus, ce qui permet d'établir une stratégie de contrôle par analyse modale. Elle possède plusieurs avantages tels que :

- Évaluer l'impact d'une sortie mesurable en tenant compte de toutes les entrées de commande du système.
- Évaluer l'impact d'une entrée de commande en tenant compte de toutes les mesures disponibles.
- Produire des mesures d'observabilité/gouvernabilité normalisées par rapport à la matrice des résidus, ce qui limite les valeurs maximales à 1.

L'intégration de cette méthode demande cependant de nombreuses modifications, autant au niveau du code source que de l'interface utilisateur et exige des mesures de validation par simulation du système.

- La bibliothèque logicielle *Automatically Tuned Linear Algebra Software* (ATLAS©) de Whaley *et al.* (2000) optimise la performance des fonctions LAPACK© en ajustant leur exécution à l'architecture matérielle de l'ordinateur sur lequel est importé le produit. Cette solution peut augmenter les performances de portabilité de l'outil, mais requiert des connaissances précises et une manipulation experte de l'utilisateur.

ANNEXE I

Déclaration des fonctions du fichier AMOD.H (MAIN)

```
/*****************************************************************************/
/* PAR     : Mathieu Perron        (ETS - GREPCI)                            */
/* FICHIER : amod.h                                                          */
/* DATE    : 8 avril 2010                                                    */
/*****************************************************************************/
/*                                                                           */
/* Ce fichier inclut la declaration des fonctions qui agissent a titre       */
/* de MAIN entre le produit logiciel et le programme hote.                   */
/*                                                                           */
/*===========================================================================*/
/*===========================================================================*/
#ifndef AMOD_H
#define AMOD_H

#include "util.h"

typedef struct{

        t_prony prony_usr;
        t_vec ymodel;
        t_amod_msg msg;

} t_amod_res;

/*===========================================================================*/
/*---------------------------------------------------------------------------*/
/*                        DECLARATION DES FONCTIONS                          */
/*---------------------------------------------------------------------------*/
/*===========================================================================*/
/*===========================================================================*/

/*---------------------------------------------------------------------------*/
/*                        FONCTION         amod                              */
/*---------------------------------------------------------------------------*/
/*                                                                           */
/*      La fonction amod agit a titre de MAIN entre le produit               */
/*      logiciel et le programme hote.                                       */
/*      RETOUR : Le contenu complet de l'analyse modale:                     */
/*               - L'amplitude, la frequence (Hz), l'amortissement,          */
/*               le dephasage (deg) et l'amortissement relatif de            */
/*               chaque mode,                                                */
/*               - Le vecteur du signal reconstruit,                         */
/*                 Les messages d'avertissements ou d'arrets.                */
/*      PARAMETRES : le signal echantillonne,                                */
/*               - la taille de l'echantillon,                               */
/*               - la periode d'echantillonnage,                             */
/*               - la limite de frequence basse (>=0.05Hz),                  */
/*               - la limite de frequence haute,                             */
/*               - la limite de l'ecart proportionnel d'amplitudes           */
/*                       MAX/MIN (1<>100),                                   */
/*               - la limite d'amortissement relatif (<=0.707),              */
/*               - le temps de debut de l'etude,                             */
/*               - l'ordre du modele d'analyse choisit (0=auto),             */
/*               - (opt.) elimination du filtre passe-bande                  */
/*                       (f_low<>f_high)  pour val=0.                        */
/*---------------------------------------------------------------------------*/
t_amod_res amod(double* y, int ndata, double t_ech, double f_low, double f_high, double alim,
double zlim, double trig, int mod_order, int filter);
```

```
/*-------------------------------------------------------------------------*/
/*                        FONCTION amod_res_free                           */
/*-------------------------------------------------------------------------*/
/*                                                                         */
/*       La fonction amod_res_free permet de liberer la memoire des        */
/*       elements retournes par la fonction amod lorsque le client         */
/*       termine l'utilisation de l'algorithme. Elle doit etre appelee     */
/*       par le programme hote.                                            */
/*       RETOUR : aucun                                                    */
/*       PARAMETRES : le contenu complet envoye au client par amod.        */
/*-------------------------------------------------------------------------*/
void amod_res_free(t_amod_res client);

#endif
```

ANNEXE II

Définition des fonctions du fichier AMOD.C (MAIN)

```
/*****************************************************************************/
/* PAR     : Mathieu Perron        (ETS - GREPCI)                            */
/* FICHIER : amod.c                                                          */
/* DATE    : 8 avril 2010                                                    */
/*****************************************************************************/
/*                                                                           */
/* Ce fichier inclut la definition des fonctions qui agissent a titre        */
/* de MAIN entre le produit logiciel et le programme hote.                   */
/*                                                                           */
/*===========================================================================*/
/*===========================================================================*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <malloc.h>
#include "nr.h"
#include "nrutil.h"
#include "apron.h"
#include "ymodel.h"
#include "amod.h"
#include "data.h"
#include "era.h"
#define NI 1
#define NJ 1

/*===========================================================================*/
/*---------------------------------------------------------------------------*/
/*                        DEFINITION DES FONCTIONS                           */
/*---------------------------------------------------------------------------*/
/*===========================================================================*/
/*===========================================================================*/

/*---------------------------------------------------------------------------*/
/*                        FONCTION         amod                              */
/*---------------------------------------------------------------------------*/

/* ATTENTION: les valeurs du vecteur 'y' commencent a l'indice 0 et non 1 */

t_amod_res amod(double* y, int ndata, double rate, double f_low, double f_high, double alim,
double zlim, double trig, int mod_order, int use_band_pass){

        /*Variables*/
        int l;
        t_data_inf info;
        t_data_dec real_dec;
        t_abc system;
        t_prony prony_tot, output;
        t_vec model;
        t_amod_msg msg;
        t_amod_res client;

        memset(&client, 0, sizeof(t_amod_res));

        /*** Step1: Data processing ***/
        data_chk(ndata, rate, f_low, f_high, trig, &client.msg);

                /*Exit*/
                if (msg.errcode)
                {
                        client.msg = msg;
                        return client;
                }

        info = data_info(ndata, rate, f_low, f_high, trig, &client.msg);
```

```
                /*Exit*/
                if (client.msg.errcode)
                {
                        return client;
                }

        real_dec = data_dec(y, ndata, info.dec, rate, trig);

        /*** Step2: State-space system identification ***/
        system = era(real_dec.data, info.nr, info.nc, NI,NJ,info.rate_dec,mod_order,&client.msg);

                /*Exit*/
                if (client.msg.errcode)
                {
                        free_dmatrix(system.a.matrix,1,system.a.nr,1,system.a.nc);
                        free_dvector(system.b.vec,1,system.b.nc);
                        free_dvector(system.c.vec,1,system.c.nc);
                        return client;
                }

        /*** Step3: Prony parameter estimatation ***/
        prony_tot = apron_tot(system, f_low, f_high, use_band_pass);
                free_dmatrix(system.a.matrix,1,system.a.nr,1,system.a.nc);
                free_dvector(system.b.vec,1,system.b.nc);
                free_dvector(system.c.vec,1,system.c.nc);

        output = apron_usr(prony_tot, zlim, alim);

        /*** Step4: Output model validation ***/
        model = ymodel(prony_tot, ndata, rate, trig);
                free(prony_tot.mag);
                free(prony_tot.damp);
                free(prony_tot.fn);
                free(prony_tot.theta_deg);
                free(prony_tot.zeta);

        /*** ERA/Prony Final Output ***/
        client.prony_usr = output;
        client.ymodel = model;

        return client;
}

/*-------------------------------------------------------------------------*/
/*                        FONCTION         amod_res_free                   */
/*-------------------------------------------------------------------------*/
void amod_res_free(t_amod_res client){

        free(client.prony_usr.zeta);
        free(client.prony_usr.theta_deg);
        free(client.prony_usr.fn);
        free(client.prony_usr.damp);
        free(client.prony_usr.mag);
        if(client.msg.warnmsg){
                free(client.msg.warnmsg);
        }
        if(client.msg.errmsg){
                free(client.msg.errmsg);
        }
        free_dvector(client.ymodel.vec,1,client.ymodel.nc);
}
```

ANNEXE III

Déclaration des fonctions du fichier ERA.H

```
/*****************************************************************************/
/* PAR     : Mathieu Perron        (ETS - GREPCI)                            */
/* FICHIER : era.h                                                           */
/* DATE    : 8 avril 2010                                                    */
/*****************************************************************************/
/*                                                                           */
/* Ce fichier inclut la declaration des fonctions qui forment                */
/* l'algorithme de realisation minimale du systeme (ERA) qui permet          */
/* d'obtenir un modele d'etat (A,B,C) reduit et dont la reponse a            */
/* l'impulsion correspond au signal temporel analyse.                        */
/*                                                                           */
/*===========================================================================*/
/*===========================================================================*/
#ifndef ERA_H
#define ERA_H
#include "util.h"

typedef struct{

        t_matrix u;
        t_matrix v_trans;
        t_matrix v;
        double *w;

} t_svd;

typedef struct{

        t_matrix u_era;
        t_matrix v_trans_era;
        t_matrix v_era;
        double *w_era;
        int nd;

} t_mat_era;

typedef struct{

        double *gov_c;
        double *obs_g;
        int nd;

} t_mat_cg;

typedef struct{

        t_matrix a;
        t_vec b;
        t_vec c;

} t_abc;

/*===========================================================================*/
/*---------------------------------------------------------------------------*/
/*                          DECLARATION DES FONCTIONS                        */
/*---------------------------------------------------------------------------*/
/*===========================================================================*/
/*===========================================================================*/
```

```
/*-------------------------------------------------------------------------*/
/*                        FONCTION         era                             */
/*-------------------------------------------------------------------------*/
/*                                                                         */
/*       La fonction era permet d'obtenir un modele d'etat continu         */
/*       minimal (A,B,C) dont la reponse a l'impulsion correspond          */
/*       au signal temporel analyse.                                       */
/*       RETOUR : Le modele d'etat sous la forme de 3 matrices A,B,C,      */
/*                - Les messages d'avertissements.                         */
/*       PARAMETRES : Le signal echantillonne et decime,                   */
/*                - le nombre de rangee minimal de la matrice de Hankel,   */
/*                - le nombre de colonne minimal de la matrice de Hankel,  */
/*                - NI=1 (define), NJ=1 (define),                          */
/*                - la periode d'echantillonnage ajustee,                  */
/*                - l'ordre du model determine (ou auto=0).                */
/*                                                                         */
/*-------------------------------------------------------------------------*/
t_abc era(double *real_sel, int nr, int nc, int ni, int nj, double tech, int mod, t_amod_msg *msg);

/*-------------------------------------------------------------------------*/
/*                        FONCTION(static)         genhank                 */
/*-------------------------------------------------------------------------*/
/*                                                                         */
/*       La fonction genhank permet de creer une matrice de Hankel         */
/*       en considerant les donnees du signal comme les parametres de      */
/*       Markov.                                                           */
/*       RETOUR : La matrice de Hankel.                                    */
/*       PARAMETRES : - Le nombre de rangee minimal de la matrice,         */
/*                - le nombre de colonne minimal de la matrice,            */
/*                - NI=1 (define), NJ=1 (define),                          */
/*                - l'indice du premier terme extrait du signal,           */
/*                - le signal echantillonne et decime.                     */
/*                                                                         */
/*-------------------------------------------------------------------------*/
static t_matrix genhank(int nr, int nc, int ni, int nj, int k, double *real_sel);

/*-------------------------------------------------------------------------*/
/*                        FONCTION(static)         svd_uwv                 */
/*-------------------------------------------------------------------------*/
/*                                                                         */
/*       La fonction svd_uwv permet la decomposition singuliere d'une      */
/*       matrice X en 3 matrices selon: X = U*W*V'.                        */
/*       RETOUR : Les matrices: U,W,V',V.                                  */
/*       PARAMETRES : La matrice de Hankel a y_ech(1).                     */
/*                                                                         */
/*-------------------------------------------------------------------------*/
static t_svd svd_uvw(t_matrix hankel);

/*-------------------------------------------------------------------------*/
/*                        FONCTION(static)         mat_era                 */
/*-------------------------------------------------------------------------*/
/*                                                                         */
/*       La fonction mat_era permet de reduire les matrices de la          */
/*       decomposition singuliere de svd_uwv selon l'ordre du modele       */
/*       determine par l'utilisateur ou selon l'analyse des valeurs        */
/*       singulieres contenues dans W.                                     */
/*       RETOUR : Les matrices reduites: U,W^(1/2),V',V.                   */
/*       PARAMETRES : Les matrices de la decomposition svd_uwv.            */
/*                - l'ordre du model determine (ou auto=0).                */
/*                                                                         */
/*-------------------------------------------------------------------------*/
static t_mat_era mat_era(t_svd svd, int mod, t_amod_msg *msg);

/*-------------------------------------------------------------------------*/
/*                        FONCTION(static)         mat_cg                  */
/*-------------------------------------------------------------------------*/
/*                                                                         */
/*       La fonction mat_cg permet de calculer les matrices C et G         */
/*       selon:   C = U*W^(1/2)*tech; G = W^(1/2)*V.                       */
/*       RETOUR : Les matrices C et G.                                     */
/*       PARAMETRES : Les matrices reduites: U,W^(1/2),V',V.               */
/*                - la periode d'echantillonnage ajustee.                  */
/*                                                                         */
/*-------------------------------------------------------------------------*/
static t_mat_cg mat_cg(t_mat_era era, double tech);
```

```
/*-------------------------------------------------------------------------*/
/*                        FONCTION(static)          mat_f                  */
/*-------------------------------------------------------------------------*/
/*                                                                         */
/*      La fonction mat_f permet de calculer la matrice d'etat             */
/*      du domaine discret F selon: F = W^(-1/2)*U'*hank2*V*W^(-1/2).      */
/*      RETOUR : La matrice F.                                             */
/*      PARAMETRES : La matrice de Hankel a y_ech(2),                      */
/*               - les matrices reduites: U,W^(1/2),V',V.                  */
/*                                                                         */
/*-------------------------------------------------------------------------*/
static t_matrix mat_f(t_matrix hank2, t_mat_era era);

/*-------------------------------------------------------------------------*/
/*                        FONCTION(static)          mat_a                  */
/*-------------------------------------------------------------------------*/
/*                                                                         */
/*      La fonction mat_a permet de calculer la matrice d'etat             */
/*      du domaine continu A selon: A = logm(F)/tech.                      */
/*      RETOUR : La matrice A.                                             */
/*      PARAMETRES : La matrice F, le taux d'echantillonnage ajustee.      */
/*                                                                         */
/*-------------------------------------------------------------------------*/
static t_matrix mat_a(t_matrix f, double tech);

/*-------------------------------------------------------------------------*/
/*                        FONCTION(static)          vec_b                  */
/*-------------------------------------------------------------------------*/
/*                                                                         */
/*      La fonction vec_b permet de calculer le vecteur d'etat             */
/*      du domaine continu B selon: B = G / (A^(-1)*(F-I)).                */
/*      RETOUR : Le vecteur d'etat B.                                      */
/*      PARAMETRES : La matrice A, la matrice F, le vecteur G.             */
/*                                                                         */
/*-------------------------------------------------------------------------*/
static t_vec vec_b(t_matrix a, t_matrix f, t_mat_cg g);

/*-------------------------------------------------------------------------*/
/*                        FONCTION(static)          transp                 */
/*-------------------------------------------------------------------------*/
/*                                                                         */
/*      La fonction transp permet de calculer la transposee X'             */
/*      d'une matrice X.                                                   */
/*      RETOUR : La matrice transposee.                                    */
/*      PARAMETRES : La matrice a transposer.                              */
/*                                                                         */
/*-------------------------------------------------------------------------*/
/*static t_matrix transp(t_matrix mat);                                    */
/*-------------------------------------------------------------------------*/
static t_matrix transp(t_matrix mat);

#endif
```

ANNEXE IV

Définition des fonctions du fichier ERA.C

```
/*****************************************************************************/
/* PAR     : Mathieu Perron        (ETS - GREPCI)                            */
/* FICHIER : era.c                                                           */
/* DATE    : 8 avril 2010                                                    */
/*****************************************************************************/
/*                                                                           */
/* Ce fichier inclut la definition des fonctions qui forment                 */
/* l'algorithme de realisation minimale du systeme (ERA) qui permet          */
/* d'obtenir un modele d'etat (A,B,C) reduit et dont la reponse a            */
/* l'impulsion correspond au signal temporel analyse.                        */
/*                                                                           */
/*===========================================================================*/
/*===========================================================================*/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <malloc.h>
#include <assert.h>
#include <time.h>
#include "era.h"
#include "nr.h"
#include "nrutil.h"
#include "f2c.h"
#include "clapack.h"
#include "blaswrap.h"
#define NRANSI

/*===========================================================================*/
/*---------------------------------------------------------------------------*/
/*                        DECLARATION DES FONCTIONS                          */
/*---------------------------------------------------------------------------*/
/*===========================================================================*/
/*===========================================================================*/

/*---------------------------------------------------------------------------*/
/*                          FONCTION         era                             */
/*---------------------------------------------------------------------------*/
t_abc era(double *real_sel, int nr, int nc, int ni, int nj, double tech, int mod, t_amod_msg *msg){

        /*Variables*/
        int k;
        t_matrix hank, hank2, f, a;
        t_svd svd;
        t_mat_era svd_era;
        t_mat_cg cg;
        t_abc abc;
        t_vec vecb;
        t_vec vecc;

        /***Step 1: Hankel matrix (y_ech(1))***/
        k = 1;
        hank = genhank(nr,nc,ni,nj,k,real_sel);

        /***Step 2: Singular value decomposition of Hankel matrix***/
        svd = svd_uvw(hank);
                free_dmatrix(hank.matrix,1,hank.nr,1,hank.nc);

        /***Step 3: Mininum realisation SVD***/
        svd_era = mat_era(svd, mod, msg);
                free_dmatrix(svd.v_trans.matrix,1,svd.v_trans.nr,1,svd.v_trans.nc);
                free_dmatrix(svd.v.matrix,1,svd.v.nr,1,svd.v.nc);
                free_dmatrix(svd.u.matrix,1,svd.u.nr,1,svd.u.nc);
                free_dvector(svd.w,1,svd.u.nc);

        /***Step 4: C - G matrices***/
        cg = mat_cg(svd_era,tech);
        vecc.vec = cg.gov_c;
        vecc.nc = cg.nd;
```

```
        /***Step 5: Discrete state matrices***/
        k = 2;
        hank2 = genhank(nr,nc,ni,nj,k,real_sel);
                free(real_sel);
        f =  mat_f(hank2, svd_era);
                free_dmatrix(hank2.matrix,1,hank2.nr,1,hank2.nc);
                free_dmatrix(svd_era.v_trans_era.matrix,1,svd_era.v_trans_era.nr,1,svd_era.nd);
                free_dmatrix(svd_era.v_era.matrix,1,svd_era.v_era.nr,1,svd_era.nd);
                free_dmatrix(svd_era.u_era.matrix,1,svd_era.u_era.nr,1,svd_era.nd);
                free_dvector(svd_era.w_era,1,svd_era.nd);

        /***Step 6: Continuous state matrices***/
        a = mat_a(f, tech);
        vecb = vec_b(a,f,cg);
                free_dmatrix(f.matrix,1,f.nr,1,f.nc);
                free_dvector(cg.obs_g,1,cg.nd);

        abc.a.matrix = a.matrix;
        abc.a.nr = a.nr;
        abc.a.nc = a.nc;
        abc.b = vecb;
        abc.c = vecc;
        return abc;
}

/*-------------------------------------------------------------------------*/
/*                        FONCTION(static)          genhank                */
/*-------------------------------------------------------------------------*/
static t_matrix genhank(int nr, int nc, int ni, int nj, int k, double *real_sel){

        /*Variables*/
        int ir, ic, ji, ti, cc, rc, tmax;
        double **a, y;
        t_matrix hank;

        a = dmatrix(1,nr,1,nc);

        for(ir=1;ir<=nr;ir++){
                ji = nj*(ir-1);
                rc = ir-1;
                for(ic=1;ic<=nc;ic++){
                        ti = ni*(ic-1);
                        y = real_sel[(ji+k+ti)];
                        cc = ic-1;
                        a[rc+1][cc+1] = y;
                }
        }
        tmax = ji+k+ti;

        /*Output*/
        hank.matrix = a;
        hank.nr = nr;
        hank.nc = nc;
        hank.tmax = tmax;

        return hank;
}

/*-------------------------------------------------------------------------*/
/*                        FONCTION(static)          svd_uwv                */
/*-------------------------------------------------------------------------*/
static t_svd svd_uvw(t_matrix hank){

        /*Variables*/
        long int i, k, l, j, nr, nc, lwork, *iwork, info;
        double *a_cl, *w_cl, *u_cl, *vt_cl, *work;
        double *w, **a, **u, **pt, **v_trans;
        char jobz = 'A';
        t_svd svd;
        t_matrix v;

        nr = hank.nr;
        nc = hank.nc;
        lwork = 7*nr*nr+4*nr;
        w = dvector(1,nc);
        a = dmatrix(1,nr,1,nc);
        u = dmatrix(1,nr,1,nc);
        v_trans = dmatrix(1,nr,1,nc);
        a_cl = malloc(nr*nc*sizeof(double));
```

```
        w_cl = malloc(nr*sizeof(double));
        u_cl = malloc(nr*nc*sizeof(double));
        vt_cl = malloc(nr*nc*sizeof(double));
        work = malloc(lwork*sizeof(double));
        iwork = malloc(8*nr*sizeof(long int));

        /*Matrix copy*/
        for (k=1;k<=nr;k++) {
                for (l=1;l<=nc;l++){
                        a[k][l] = hank.matrix[k][l];
                }
        }

        /*Row to Column Major Order (C to Fortran)*/
        for(i=0;i<=((nr*nc)-1);i++){
                for (l=1;l<=nc;l++) {
                        for (k=1;k<=nr;k++){
                                a_cl[i] = a[k][l];
                                i++;
                         }
                }
        }

        /* Singular Value Decomposition A = U*W*V */
        dgesdd_(&jobz, &nr, &nc, a_cl, &nr, w_cl, u_cl, &nr, vt_cl, &nr, work, &lwork, iwork, &info);

        /*Column to Row Major Order (Fortran to C)*/
        for(i=0;i<=((nr*nc)-1);i++){
                for (l=1;l<=nc;l++) {
                        for (k=1;k<=nr;k++){
                                u[k][l] = u_cl[i];
                                i++;
                         }
                }
        }
        for(i=0;i<=((nr*nc)-1);i++){
                for (l=1;l<=nc;l++) {
                        for (k=1;k<=nr;k++){
                                v_trans[k][l] = vt_cl[i];
                                i++;
                         }
                }
        }
        for(i=0;i<=nr-1;i++){
                for (l=1;l<=nr;l++) {
                        w[l] = w_cl[i];
                        i++;
                }
        }

        /*Output*/
        svd.u.matrix = u;
        svd.u.nr = nr;
        svd.u.nc = nc;
        svd.v_trans.matrix = v_trans;
        svd.v_trans.nr = nr;
        svd.v_trans.nc = nc;
        v = transp(svd.v_trans);
        svd.v.matrix = v.matrix;
        svd.v.nr = nr;
        svd.v.nc = nc;
        svd.w = w;

        free(a_cl);
        free(w_cl);
        free(u_cl);
        free(vt_cl);
        free(work);
        free(iwork);
        free_dmatrix(a,1,nr,1,nc);
        free_dmatrix(pt,1,nr,1,nc);
        return  svd;
}
```

```
/*-------------------------------------------------------------------------*/
/*                        FONCTION(static)          mat_era                */
/*-------------------------------------------------------------------------*/
static t_mat_era mat_era(t_svd svd, int mod, t_amod_msg *msg){

        /*Variables*/
        int k, kk, l, nd=0, nr, nc, warning = 0;
        double *w_min, **u_min, **v_min, **vt_min;
        t_mat_era mat;
        char buff[256];

        nr = svd.u.nr;
        nc = svd.u.nc;

        /*Minimal realization order (nd)*/
        if(((mod >= 1) && (mod > nr)) || (mod < 1) ){

                for(k=1;k<=svd.u.nr;k++){
                        if((svd.w[k]/svd.w[1])>=0.01){
                                nd++;
                                kk = k;
                        }
                }

                if(svd.w[kk+1]/svd.w[1]>=0.0075){
                        nd++;
                }

                if(nd > 20){
                        nd = 10;

                }

                sprintf(buff,"Ordre du modele:%d (automatique)\n", nd);
                appendMsg(&(msg->warnmsg), buff);
                msg->warncode = 1;
        }

        else if( (mod >= 1) && (mod <= nr)){
                nd = mod;
                sprintf(buff,"Ordre du modele:%d (manuel)\n", nd);
                appendMsg(&(msg->warnmsg), buff);
                msg->warncode = 1;
        }

        w_min = dvector(1,nd);
        u_min = dmatrix(1,nr,1,nd);
        v_min = dmatrix(1,nr,1,nd);
        vt_min = dmatrix(1,nr,1,nd);

        /*Reduced sqrt(singular value)*/
        for (k=1;k<=nd;k++) {
                w_min[k] = sqrt(svd.w[k]);
        }
        /*Reduced matrix u*/
        for (k=1;k<=nr;k++) {
                for (l=1;l<=nd;l++){
                        u_min[k][l] = svd.u.matrix[k][l];
                }
        }
        /*Reduced matrix v*/
        for (k=1;k<=nr;k++) {
                for (l=1;l<=nd;l++){
                        v_min[k][l] = svd.v.matrix[k][l];
                }
        }
        /*Reduced matrix v_trans*/
        for (k=1;k<=nr;k++) {
                for (l=1;l<=nd;l++){
                        vt_min[k][l] = svd.v_trans.matrix[k][l];
                }
        }
```

```
        /*Output*/
        mat.w_era = w_min;
        mat.u_era.matrix = u_min;
        mat.u_era.nr = svd.u.nr;
        mat.u_era.nc = nd;
        mat.v_era.matrix = v_min;
        mat.v_era.nr = svd.v.nr;
        mat.v_era.nc = nd;
        mat.v_trans_era.matrix = vt_min;
        mat.v_trans_era.nr = svd.v.nr;
        mat.v_trans_era.nc = nd;
        mat.nd = nd;

        return mat;
}

/*-------------------------------------------------------------------------*/
/*                      FONCTION(static)          mat_cg                   */
/*-------------------------------------------------------------------------*/
static t_mat_cg mat_cg(t_mat_era era, double tech){

        /*Variables*/
        int k, nd = era.nd;
        double *c, *g;
        t_mat_cg mat;

        c = dvector(1,nd);
        g = dvector(1,nd);

        /*C = U*W^(1/2)*tech*/
        for (k=1;k<=nd;k++){
                c[k]= era.u_era.matrix[1][k]*era.w_era[k]*tech;
        }

        /*G = W^(1/2)*V*/
        for (k=1;k<=nd;k++){
                g[k]= era.v_era.matrix[1][k]*era.w_era[k];
        }

        /*Output*/
        mat.gov_c = c;
        mat.obs_g = g;
        mat.nd = nd;

        return mat;
}

/*-------------------------------------------------------------------------*/
/*                      FONCTION(static)          mat_f                    */
/*-------------------------------------------------------------------------*/
static t_matrix mat_f(t_matrix hank2, t_mat_era era){

        /*Variables*/
        long int k, l, j, i, nr, nc, nd;
        long int inv_sq, h_sq, inv_info, *inv_ipiv;
        double **mat_f;
        double *u_f, *v_f, *h_f, *w_f, *i_w, *eye, *f1, *f2, *f3, *ftot;
        double mm_alpha = 1, mm_beta = 0;
        char mm_transa = 'N', mm_transb = 'N', mm3_transb = 'T';
        t_matrix f;

        /*For rectangular matrices(u_f, v_f, f1, f2, f3)*/
        nr = era.u_era.nr;
        nc = era.u_era.nc;
        /*For square matrix(h_f)*/
        h_sq = hank2.nr;
        /*For square matrices(w_f, i_w, eye, ftot)*/
        nd = era.nd;
        inv_sq = nd;

        mat_f = dmatrix(1,nd,1,nd);
        inv_ipiv = malloc(nd*nd*sizeof(long int));
        u_f = malloc(nr*nc*sizeof(double));
        v_f = malloc(nr*nc*sizeof(double));
        h_f = malloc(h_sq*h_sq*sizeof(double));
        w_f = malloc(nd*nd*sizeof(double));
        i_w = malloc(nd*nd*sizeof(double));
        eye = malloc(nd*nd*sizeof(double));
        f1 = malloc(nr*nc*sizeof(double));
```

```
f2 = malloc(nr*nc*sizeof(double));
f3 = malloc(nc*nr*sizeof(double));
ftot = malloc(nd*nd*sizeof(double));

/**Row to Column major order (C to Fortran)**/
/* U_ERA */
for(i=0;i<=(nr*nc)-1;i++){
        for (l=1;l<=nc;l++) {
                for (k=1;k<=nr;k++){
                        u_f[i] = era.u_era.matrix[k][l];
                        i++;
                }
        }
}
/* V_ERA */
for(i=0;i<=(nr*nc)-1;i++){
        for (l=1;l<=nc;l++) {
                for (k=1;k<=nr;k++){
                        v_f[i] = era.v_era.matrix[k][l];
                        i++;
                }
        }
}
/* HANK2 */
for(i=0;i<=(h_sq*h_sq)-1;i++){
        for (l=1;l<=h_sq;l++) {
                for (k=1;k<=h_sq;k++){
                        h_f[i] = hank2.matrix[k][l];
                        i++;
                }
        }
}
/* W_ERA */
for(i=0;i<=(nd*nd)-1;i++){
        w_f[i] = 0;
}
for(i=0;i<=(nd*nd)-1;i=i+(nd+1)){
        w_f[i] = w_f[i]+era.w_era[(i+nd+1)/(1+nd)];
}

/**Identity matrix (eye)**/
for(i=0;i<=(nd*nd)-1;i++){
        eye[i] = 0;
}
for(i=0;i<=(nd*nd)-1;i=i+(nd+1)){
        eye[i] = eye[i]+1;
}

/***---------------------------------- F Matrix ----------------------------------***/

/**--------      1: f1 Matrix = v_era * inv(w)      --------**/
/*inv(w) = w^-1 = (wx = I, solve for x)*/
for(i=0;i<=(nd*nd)-1;i++){
        i_w[i] = eye[i];
}
dgesv_(&inv_sq,&inv_sq,w_f,&inv_sq,inv_ipiv,i_w,&inv_sq,&inv_info);
/*product: v_f * inv(w_f)*/
dgemm_(&mm_transa,&mm_transb,&nr,&nd,&nd,&mm_alpha,v_f,&nr,i_w,&nd,&mm_beta,f1,&nr);

/**--------      2: f2 Matrix = hank2 * f1 --------**/
dgemm_(&mm_transa,&mm_transb,&h_sq,&nc,&nr,&mm_alpha,h_f,&h_sq,f1,&nr,&mm_beta,f2,&h_sq);

/**--------      3: f3 Matrix = inv(w) * transp(u)  --------**/
dgemm_(&mm_transa,&mm3_transb,&nd,&nr,&nd,&mm_alpha,i_w,&nd,u_f,&nr,&mm_beta,f3,&nd);

/**--------      4: ftot Matrix = f3 * f2  --------**/
dgemm_(&mm_transa,&mm_transb,&nd,&nd,&nr,&mm_alpha,f3,&nd,f2,&nr,&mm_beta,ftot,&nd);

/*Column to Row Major Order (Fortran to C)*/
for(i=0;i<=((nd*nd)-1);i++){
        for (l=1;l<=nd;l++) {
                for (k=1;k<=nd;k++){
                        mat_f[k][l] = ftot[i];
                        i++;
                }
        }
}
/***-------------------------------------------------------------------------------***/
```

```
        /*Output*/
        f.matrix = mat_f;
        f.nr = nd;
        f.nc = nd;

        free(ftot);
        free(f3);
        free(f2);
        free(f1);
        free(eye);
        free(i_w);
        free(w_f);
        free(h_f);
        free(v_f);
        free(u_f);
        free(inv_ipiv);
        return f;
}

/*-------------------------------------------------------------------------*/
/*                        FONCTION(static)          mat_a                  */
/*-------------------------------------------------------------------------*/
t_matrix mat_a(t_matrix f, double tech){

        /*Variables*/
        long int i,k,l,nr,nc,n,usey,lwork;
        long int *ipiv;
        double **mat_f,**mat_a,*f_for,*wrk,*x,*work,tol;
        t_matrix a;

        nr = f.nr;
        nc = f.nc;
        n = nr;
        lwork = n*n;
        tol = 0.000001;
        usey = 1;
        mat_f = dmatrix(1,nr,1,nc);
        mat_a = dmatrix(1,nr,1,nc);
        f_for = malloc(nr*nc*sizeof(double));
        wrk = malloc(n*4*n*sizeof(double));
        x = malloc(n*2*sizeof(double));
        ipiv = malloc(n*sizeof(long int));
        work = malloc(lwork*sizeof(double));

        /*Matrix copy*/
        for (k=1;k<=nr;k++) {
                for (l=1;l<=nc;l++){
                        mat_f[k][l] = f.matrix[k][l];
                }
        }

        /*Row to Column Major Order (C to Fortran)*/
        for(i=0;i<=(nc*nr)-1;i++){
                for (l=1;l<=nc;l++) {
                        for (k=1;k<=nr;k++){
                                f_for[i] = mat_f[k][l];
                                i++;
                        }
                }
        }

        /*** A = logm(f)/T ***/
        dgelog_(&n, f_for, &n, wrk, &n, x, &n, ipiv, work, &lwork, &tol, &usey);

        /*Column to Row Major Order (Fortran to C)*/
        for(i=0;i<=((nc*nr)-1);i++){
                for (l=1;l<=nc;l++) {
                        for (k=1;k<=nr;k++){
                                mat_a[k][l] = (f_for[i]) / tech;
                                i++;
                        }
                }
        }
```

```
        /*Output*/
        a.matrix = mat_a;
        a.nr = nr;
        a.nc = nr;

        free(work);
        free(ipiv);
        free(x);
        free(wrk);
        free(f_for);
        free_dmatrix(mat_f,1,nr,1,nc);
        return a;

}

/*-------------------------------------------------------------------------*/
/*                      FONCTION(static)          vec_b                    */
/*-------------------------------------------------------------------------*/
static t_vec vec_b(t_matrix a, t_matrix f, t_mat_cg g){

        /*Variables*/
        long int i,k,l,nr,nc,nd;
        long int dg_or,dg_sq,*dg_ipiv,dg1_info,dg2_info;
        long int mm_sq;
        long int mv_sq, mv_incx = 1, mv_incy = 1;
        double **mat_a,**mat_f,*vec_g,*vec_b,*a_for,*f_for,*g_for,*eye,*i_a,*f2_for,*i_tmp1;
        double mm_alpha = 1, mm_beta = 0, *tmp1;
        double mv_alpha = 1, mv_beta = 0, *b_for;
        char mm_transa = 'N',mm_transb = 'N';
        char mv_trans = 'N';
        t_vec vec;

        nr = a.nr;
        nc = a.nc;
        nd = g.nd;
        /*Square matrices*/
        dg_or = nc*nr;
        dg_sq = nr;
        /*Square matrices*/
        mm_sq = nr;
        mv_sq = nr;

        mat_a = dmatrix(1,nr,1,nc);
        mat_f = dmatrix(1,nr,1,nc);
        vec_g = dvector(1,nd);
        vec_b = dvector(1,nd);
        a_for = malloc(nr*nc*sizeof(double));
        f_for = malloc(nr*nc*sizeof(double));
        g_for = malloc(nd*sizeof(double));
        eye = malloc(nr*nc*sizeof(double));
        i_a = malloc(nr*nc*sizeof(double));
        dg_ipiv = malloc(nc*nr*sizeof(int));
        f2_for = malloc(nr*nc*sizeof(double));
        tmp1 = malloc(nr*nc*sizeof(double));
        i_tmp1 = malloc(nr*nc*sizeof(double));
        b_for = malloc(nd*sizeof(double));

        /**Matrix and vector copy**/
        /* A */
        for (k=1;k<=nr;k++) {
                for (l=1;l<=nc;l++){
                        mat_a[k][l] = a.matrix[k][l];
                }
        }
        /* F */
        for (k=1;k<=nr;k++) {
                for (l=1;l<=nc;l++){
                        mat_f[k][l] = f.matrix[k][l];
                }
        }
        /* G */
        for (l=1;l<=nd;l++) {
                vec_g[l] = g.obs_g[l];
        }
```

```
/**Row to Column Major Order (C to Fortran)**/
/* A */
for(i=0;i<=(nr*nc)-1;i++){
        for (l=1;l<=nc;l++) {
                for (k=1;k<=nr;k++){
                        a_for[i] = mat_a[k][l];
                        i++;
                }
        }
}
/* F */
for(i=0;i<=(nr*nc)-1;i++){
        for (l=1;l<=nc;l++) {
                for (k=1;k<=nr;k++){
                        f_for[i] = mat_f[k][l];
                        i++;
                }
        }
}
/*Vector offset*/
/* G */
for(i=0;i<=nd-1;i++){
        for (l=1;l<=nd;l++){
                g_for[i] = vec_g[l];
                i++;
         }
}

/*Identity matrix (eye)*/
for(i=0;i<=(nr*nc)-1;i++){
        eye[i] = 0;
}
for(i=0;i<=(nr*nc)-1;i=i+(nr+1)){
        eye[i] = eye[i]+1;
}

/***------------- B Matrix = G / (A^-1*(F-I)) = (inv[inv(A)*(F-I)])*G --------------***/

/**---- 1: inv(A) = A^-1 = (Ax = I, solve for x) ----**/
for(i=0;i<=(nr*nc)-1;i++){
        i_a[i] = eye[i];
}
dgesv_(&dg_sq,&dg_sq,a_for,&dg_sq,dg_ipiv,i_a,&dg_sq,&dgl_info);

/**---- 2: (F-I) ----**/
for(i=0;i<=(nr*nc)-1;i++){
        f2_for[i] = f_for[i]-eye[i];
}

/**---- 3: tmp1 = inv(A) * (F-I) ----**/
dgemm_(&mm_transa,&mm_transb,&mm_sq,&mm_sq,&mm_sq,&mm_alpha,i_a,&mm_sq,f2_for,&mm_sq,
        &mm_beta,tmp1,&mm_sq);

/**---- 4: inv(tmp1) = tmp1^-1 = (tmp1*x = I, solve for x) ----**/
for(i=0;i<=(nr*nc)-1;i++){
        i_tmp1[i] = eye[i];
}
dgesv_(&dg_sq,&dg_sq,tmp1,&dg_sq,dg_ipiv,i_tmp1,&dg_sq,&dg2_info);

/**---- 5: B = inv(tmp1) * G ----**/
dgemv_(&mv_trans,&mv_sq,&mv_sq,&mv_alpha,i_tmp1,&mv_sq,g_for,&mv_incx,&mv_beta,b_for,&mv_incy);

/*Column to Row Major Order (Fortran to C)*/
for(i=0;i<=nc-1;i++){
        for (l=1;l<=nc;l++){
                vec_b[l] = b_for[i];
                i++;
         }
}

/***-------------------------------------------------------------------------------***/
```

```
        /*Output*/
        vec.vec = vec_b;
        vec.nc = nc;

        free(b_for);
        free(i_tmp1);
        free(tmp1);
        free(f2_for);
        free(dg_ipiv);
        free(i_a);
        free(eye);
        free(g_for);
        free(f_for);
        free(a_for);
        free_dvector(vec_g,1,nd);
        free_dmatrix(mat_f,1,nr,1,nc);
        free_dmatrix(mat_a,1,nr,1,nc);
        return vec;
}

/*-------------------------------------------------------------------------*/
/*                        FONCTION(static)          transp                 */
/*-------------------------------------------------------------------------*/
static t_matrix transp(t_matrix mat){

        /*Variables*/
        int k,l, nr, nc;
        double **a;
        t_matrix transp;
        nr = mat.nr;
        nc = mat.nc;

        a=dmatrix(1,nc,1,nr);

        /*Transpose*/
        for (k=1;k<=nr;k++) {
                for (l=1;l<=nc;l++){
                        a[l][k] = mat.matrix[k][l];
                }
        }

        /*Output*/
        transp.matrix = a;
        transp.nr = nc;
        transp.nc = nr;

        return transp;
}
```

ANNEXE V

Déclaration des fonctions du fichier APRON.H

```
/*****************************************************************************/
/* PAR     : Mathieu Perron        (ETS - GREPCI)                            */
/* FICHIER : apron.h                                                         */
/* DATE    : 8 avril 2010                                                    */
/*****************************************************************************/
/*                                                                           */
/* Ce fichier inclut la declaration des fonctions qui forment                */
/* l'analyse de Prony du signal qui permet d'obtenir la somme                */
/* des modes de la forme: A*exp(-sigma*t)*cos(wt+theta)                      */
/* a partir des matrices A,B,C.                                              */
/*                                                                           */
/*===========================================================================*/
/*===========================================================================*/
#ifndef APRON_H
#define APRON_H

#include "era.h"

typedef struct{

        doublecomplex *val;
        int val_nr;
        doublecomplex *vec;
        int vec_nr;

} t_eig;

typedef struct {

        doublecomplex *vec;
        int nc;

} t_vec_comp;

/*===========================================================================*/
/*---------------------------------------------------------------------------*/
/*                        DECLARATION DES FONCTIONS                          */
/*---------------------------------------------------------------------------*/
/*===========================================================================*/
/*===========================================================================*/

/*---------------------------------------------------------------------------*/
/*                        FONCTION         apron_tot                         */
/*---------------------------------------------------------------------------*/
/*                                                                           */
/*       La fonction apron_tot genere l'analyse de Prony complete du         */
/*       signal selon les limites en frequence de l'etude.                   */
/*       RETOUR : - L'amplitude, la frequence (Hz), l'amortissement,         */
/*                le dephasage (deg) et l'amortissement relatif de           */
/*                chaque mode.                                               */
/*       PARAMETRES : Les matrices d'etat A,B,C du systeme,                  */
/*                - la limite de frequence basse (>=0.05Hz),                 */
/*                - la limite de frequence haute,                            */
/*                - (opt.) elimination du filtre passe-bande                 */
/*                         (f_low<>f_high)  pour val=1.                      */
/*                          elimination du filtre passe-bande et             */
/*                          modes instables pour val=2.                      */
/*                                                                           */
/*---------------------------------------------------------------------------*/
t_prony apron_tot(t_abc system, double f_low, double f_high, int filter);
```

```
/*-------------------------------------------------------------------------*/
/*                      FONCTION        apron_usr                          */
/*-------------------------------------------------------------------------*/
/*                                                                         */
/*      La fonction apron_usr reduit l'analyse de Prony du signal          */
/*      selon la limite d'amortissement relatif et de proportionnalite     */
/*      de l'ecart d'amplitudes.                                           */
/*      RETOUR : - L'amplitude, la frequence (Hz), l'amortissement,        */
/*               le dephasage (deg) et l'amortissement relatif de          */
/*               chaque mode retenu.                                       */
/*      PARAMETRES : Le contenu de l'analyse complete selon apron_tot,     */
/*               - la limite d'amortissement relatif,                      */
/*               - la limite proportionnelle de l'ecart d'amplitudes       */
/*                      MAX/MIN (1<>100),                                  */
/*                                                                         */
/*-------------------------------------------------------------------------*/
t_prony apron_usr(t_prony prony_tot, double zeta_limit, double amp_limit);

/*-------------------------------------------------------------------------*/
/*                      FONCTION(static)        eig                        */
/*-------------------------------------------------------------------------*/
/*                                                                         */
/*      La fonction eig genere les elements complexes des valeurs (l)      */
/*      et des vecteurs (p) propres a la matrice X selon: (X-lI)*p = 0.     */
/*      RETOUR : - Les vecteurs de valeurs et de vecteurs propres          */
/*               complexes et leur dimensions.                             */
/*      PARAMETRES : La matrice a analyser.                                */
/*                                                                         */
/*-------------------------------------------------------------------------*/
static t_eig eig(t_matrix mat);

/*-------------------------------------------------------------------------*/
/*                      FONCTION(static)        ma_tmp                     */
/*-------------------------------------------------------------------------*/
/*                                                                         */
/*      La fonction ma_tmp multiplie les vecteurs propres gauches          */
/*      a un vecteur d'etat B.                                             */
/*      RETOUR :  - Le vecteur resultant de la multiplication.             */
/*      PARAMETRES : Le vecteur des vecteurs propres complexes droits,     */
/*               - le vecteur d'etat B.                                    */
/*                                                                         */
/*-------------------------------------------------------------------------*/
static t_vec_comp ma_tmp(t_eig eig_vec, t_vec b_vec);

/*-------------------------------------------------------------------------*/
/*                      FONCTION(static)        ms_tmp                     */
/*-------------------------------------------------------------------------*/
/*                                                                         */
/*      La fonction ms_tmp multiplie les vecteurs propres droits           */
/*      a un vecteur d'etat C.                                             */
/*      RETOUR :  - Le vecteur resultant de la multiplication.             */
/*      PARAMETRES : Le vecteur des vecteurs propres complexes droits,     */
/*               - le vecteur d'etat C.                                    */
/*                                                                         */
/*-------------------------------------------------------------------------*/
static t_vec_comp ms_tmp(t_eig eig_vec, t_vec c);

/*-------------------------------------------------------------------------*/
/*                      FONCTION(static)        r_tmp                      */
/*-------------------------------------------------------------------------*/
/*                                                                         */
/*      La fonction r_tmp calcule les valeurs residuels du modele d'etat   */
/*      selon les vecteurs propres droits et gauches et les vecteurs       */
/*      d'etat B et C.                                                     */
/*      RETOUR :  - Le vecteur des valeurs residuels.                      */
/*      PARAMETRES : Le vecteur complexe MA,                               */
/*               - le vecteur complexe MS.                                 */
/*                                                                         */
/*-------------------------------------------------------------------------*/
static t_vec_comp r_tmp(t_vec_comp ma, t_vec_comp ms);
```

```
/*-------------------------------------------------------------------------*/
/*                        FONCTION(static)          mag                    */
/*-------------------------------------------------------------------------*/
/*                                                                         */
/*       La fonction mag calcule les amplitudes de chacun des modes        */
/*       en fonction des valeurs residuels du modele d'etat.               */
/*       RETOUR : Le vecteur d'amplitudes des modes.                       */
/*       PARAMETRES : Le vecteur des valeurs residuels.                    */
/*                                                                         */
/*-------------------------------------------------------------------------*/
static t_vec mag(t_vec_comp r_tmp);

/*-------------------------------------------------------------------------*/
/*                        FONCTION(static)          damp                   */
/*-------------------------------------------------------------------------*/
/*                                                                         */
/*       La fonction damp calcule l'amortissement de chacun des modes      */
/*       en fonction des valeurs propres du modele d'etat.                 */
/*       RETOUR : Le vecteur d'amortissements des modes.                   */
/*       PARAMETRES : Le vecteur des valeurs propres complexes.            */
/*                                                                         */
/*-------------------------------------------------------------------------*/
static t_vec damp(t_eig eig_val);

/*-------------------------------------------------------------------------*/
/*                        FONCTION(static)          fn                     */
/*-------------------------------------------------------------------------*/
/*                                                                         */
/*       La fonction fn calcule la frequence(Hz) de chacun des modes       */
/*       en fonction des valeurs propres du modele d'etat.                 */
/*       RETOUR : Le vecteur de frequences(Hz) des modes.                  */
/*       PARAMETRES : Le vecteur des valeurs propres complexes.            */
/*                                                                         */
/*-------------------------------------------------------------------------*/
static t_vec fn(t_eig eig_val);

/*-------------------------------------------------------------------------*/
/*                        FONCTION(static)          theta_deg              */
/*-------------------------------------------------------------------------*/
/*                                                                         */
/*       La fonction theta_deg calcule le dephasage(deg) de chacun         */
/*       des modes en fonction de l'argument des valeurs residuels         */
/*       complexes du modele d'etat.                                       */
/*       RETOUR : Le vecteur des dephasage(deg) des modes.                 */
/*       PARAMETRES : Le vecteur des valeurs residuels.                    */
/*                                                                         */
/*-------------------------------------------------------------------------*/
static t_vec theta_deg(t_vec_comp r_tmp);

/*-------------------------------------------------------------------------*/
/*                        FONCTION(static)          zeta                   */
/*-------------------------------------------------------------------------*/
/*                                                                         */
/*       La fonction zeta genere l'amortissement relatif de chaque         */
/*       mode selon: zeta = -damp/sqrt((fn_rad)^2+(damp)^2).               */
/*       RETOUR : Le vecteur de dephasages(deg) des modes.                 */
/*       PARAMETRES : Le vecteur complexe R.                               */
/*                                                                         */
/*-------------------------------------------------------------------------*/
static t_vec zeta(t_vec fn, t_vec damp);

/*-------------------------------------------------------------------------*/
/*                        FONCTION(static)          prony_cc               */
/*-------------------------------------------------------------------------*/
/*                                                                         */
/*       La fonction prony_cc permet d'eliminer un terme des modes         */
/*       se presentant sous forme de conjugue complexe.                    */
/*       RETOUR : Le contenu complet et reduit des modes.                  */
/*       PARAMETRES : Le contenu complet, non-reduit des modes.            */
/*                                                                         */
/*-------------------------------------------------------------------------*/
static t_prony prony_cc(t_prony tmp);
```

```
/*-------------------------------------------------------------------------*/
/*                        FONCTION(static)          prony_bp               */
/*-------------------------------------------------------------------------*/
/*                                                                         */
/*       La fonction prony_bp permet de filtrer les modes qui sont         */
/*       en dehors de la fenetre d'etude frequentielle determinee          */
/*       par l'utilisateur.                                                */
/*       RETOUR : Le contenu filtre des modes.                             */
/*       PARAMETRES : Le contenu non-filtre des modes,                     */
/*                - la limite de frequence basse (>=0.05Hz),               */
/*                - la limite de frequence haute.                          */
/*                                                                         */
/*-------------------------------------------------------------------------*/
static t_prony prony_bp(t_prony tmp, double f_low, double f_high);

/*-------------------------------------------------------------------------*/
/*                        FONCTION(static)          prony_ins              */
/*-------------------------------------------------------------------------*/
/*                                                                         */
/*       La fonction prony_ins permet de filtrer les modes dont            */
/*       l'amortissement(damp) est contenu entre 0 et 0.001.               */
/*       RETOUR : Le contenu filtre des modes.                             */
/*       PARAMETRES : Le contenu non-filtre des modes,                     */
/*                                                                         */
/*-------------------------------------------------------------------------*/
static t_prony prony_ins(t_prony tmp);

/*-------------------------------------------------------------------------*/
/*                        FONCTION(static)          prony_zlim             */
/*-------------------------------------------------------------------------*/
/*                                                                         */
/*       La fonction prony_zlim permet de filtrer l'affichage des          */
/*       modes dont l'amortissement relatif est superieur a la limite      */
/*       determinee par l'utilisateur.                                     */
/*       RETOUR : Le contenu filtre des modes.                             */
/*       PARAMETRES : Le contenu non-filtre des modes,                     */
/*                - la limite d'amortissement relatif (<=0.707).           */
/*                                                                         */
/*-------------------------------------------------------------------------*/
static t_prony prony_zlim(t_prony tmp, double zeta_limit);

/*-------------------------------------------------------------------------*/
/*                        FONCTION(static)          prony_alim             */
/*-------------------------------------------------------------------------*/
/*                                                                         */
/*       La fonction prony_alim permet de filtrer l'affichage des          */
/*       modes k dont l'ecart proportionnel d'amplitude (MAX/ak)           */
/*       est inferieur au facteur determine par l'utilisateur.             */
/*       RETOUR : Le contenu filtre des modes.                             */
/*       PARAMETRES : Le contenu non-filtre des modes,                     */
/*                 la limite de l'ecart proportionnel d'amplitudes         */
/*                       (1<>100).                                         */
/*                                                                         */
/*-------------------------------------------------------------------------*/
static t_prony prony_alim(t_prony tmp, double amp_limit);

/*-------------------------------------------------------------------------*/
/*                        FONCTION(static)          prony_dc               */
/*-------------------------------------------------------------------------*/
/*                                                                         */
/*       La fonction prony_dc permet de filtrer l'affichage des            */
/*       modes DC de frequence(fn) nulle.                                  */
/*       RETOUR : Le contenu filtre des modes.                             */
/*       PARAMETRES : Le contenu non-filtre des modes.                     */
/*                                                                         */
/*-------------------------------------------------------------------------*/
static t_prony prony_dc(t_prony tmp);

/*-------------------------------------------------------------------------*/
/*                        FONCTION(static)          prony_sort             */
/*-------------------------------------------------------------------------*/
/*                                                                         */
/*       La fonction prony_sort permet de classer en ordre decroissant     */
/*       d'amplitude les modes affiche.                                    */
/*       RETOUR : Le contenu classe des modes.                             */
/*       PARAMETRES : Le contenu non-classe des modes.                     */
/*                                                                         */
/*-------------------------------------------------------------------------*/
static t_prony prony_sort(t_prony tmp); #endif
```

ANNEXE VI

Définition des fonctions du fichier APRON.C

```
/*****************************************************************************/
/* PAR     : Mathieu Perron        (ETS - GREPCI)                            */
/* FICHIER : apron.c                                                         */
/* DATE    : 8 avril 2010                                                    */
/*****************************************************************************/
/*                                                                           */
/* Ce fichier inclut la definition des fonctions qui forment                 */
/* l'analyse de Prony du signal qui permet d'obtenir la somme                */
/* des modes de la forme: A*exp(-sigma*t)*cos(wt+theta)                      */
/* a partir des matrices A,B,C.                                              */
/*                                                                           */
/*===========================================================================*/
/*===========================================================================*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <malloc.h>
#include <assert.h>
#include <sys/param.h>
#include "f2c.h"
#include "clapack.h"
#include "blaswrap.h"
#include "nr.h"
#include "nrutil.h"
#include "apron.h"
#define NRANSI

/*===========================================================================*/
/*---------------------------------------------------------------------------*/
/*                        DEFINITION DES FONCTIONS                           */
/*---------------------------------------------------------------------------*/
/*===========================================================================*/
/*===========================================================================*/

/*---------------------------------------------------------------------------*/
/*                          FONCTION         apron_tot                       */
/*---------------------------------------------------------------------------*/
t_prony apron_tot(t_abc system, double f_low, double f_high, int filter){

        /*Variables*/
        int l;
        t_eig eigv;
        t_vec_comp ma,ms,r;
        t_vec amp,dn,freq,theta,z;
        t_prony prony_tmp, prony_tmp2,prony_tmp3, prony_tot;

        /***Step 1: Eigenvalues and Residue analysis***/
        eigv = eig(system.a);

        ma = ma_tmp(eigv, system.b);

        ms = ms_tmp(eigv, system.c);

        r = r_tmp(ma,ms);
                free(ma.vec);
                free(ms.vec);

        /***Step 2: Prony parameters calculation***/
        amp = mag(r);
        dn = damp(eigv);
        freq = fn(eigv);
                free(eigv.val);
                free(eigv.vec);
        theta = theta_deg(r);
                free(r.vec);
        z = zeta(freq, dn);
```

```
        /*** Step 3: Output ***/
                /* Complex Conjugate */
        prony_tmp.mag = amp.vec;
        prony_tmp.damp = dn.vec;
        prony_tmp.fn = freq.vec;
        prony_tmp.theta_deg = theta.vec;
        prony_tmp.zeta = z.vec;
        prony_tmp.qty = amp.nc;

                /*  Cleaned */
        if (!filter){
                prony_tmp2 = prony_cc(prony_tmp);
                        free_dvector(amp.vec,1,amp.nc);
                        free_dvector(dn.vec,1,dn.nc);
                        free_dvector(freq.vec,1,freq.nc);
                        free_dvector(theta.vec,1,theta.nc);
                        free_dvector(z.vec,1,z.nc);

                prony_tmp3 = prony_bp(prony_tmp2, f_low, f_high);
                        free(prony_tmp2.mag);
                        free(prony_tmp2.damp);
                        free(prony_tmp2.fn);
                        free(prony_tmp2.theta_deg);
                        free(prony_tmp2.zeta);

                prony_tot = prony_ins(prony_tmp3);
                        free(prony_tmp3.mag);
                        free(prony_tmp3.damp);
                        free(prony_tmp3.fn);
                        free(prony_tmp3.theta_deg);
                        free(prony_tmp3.zeta);
        }

        else if (filter == 1){
                prony_tot = prony_cc(prony_tmp);
                free_dvector(amp.vec,1,amp.nc);
                free_dvector(dn.vec,1,dn.nc);
                free_dvector(freq.vec,1,freq.nc);
                free_dvector(theta.vec,1,theta.nc);
                free_dvector(z.vec,1,z.nc);
        }

        else {
                prony_tmp2 = prony_cc(prony_tmp);
                        free_dvector(amp.vec,1,amp.nc);
                        free_dvector(dn.vec,1,dn.nc);
                        free_dvector(freq.vec,1,freq.nc);
                        free_dvector(theta.vec,1,theta.nc);
                        free_dvector(z.vec,1,z.nc);

                prony_tot = prony_ins(prony_tmp2);
                        free(prony_tmp2.mag);
                        free(prony_tmp2.damp);
                        free(prony_tmp2.fn);
                        free(prony_tmp2.theta_deg);
                        free(prony_tmp2.zeta);
        }

        return prony_tot;
}

/*-------------------------------------------------------------------------*/
/*                      FONCTION         apron_usr                         */
/*-------------------------------------------------------------------------*/
t_prony apron_usr(t_prony prony_tot, double zeta_limit, double amp_limit){

        /*Variables*/
        int l;
        t_prony prony_tmp1, prony_tmp2, prony_tmp3, output;

        /*User limitations*/
        prony_tmp1 = prony_zlim(prony_tot, zeta_limit);
        prony_tmp2 = prony_dc(prony_tmp1);
                        free(prony_tmp1.mag);
                        free(prony_tmp1.damp);
                        free(prony_tmp1.fn);
                        free(prony_tmp1.theta_deg);
                        free(prony_tmp1.zeta);
```

```
        prony_tmp3 = prony_alim(prony_tmp2, amp_limit);
                        free(prony_tmp2.mag);
                        free(prony_tmp2.damp);
                        free(prony_tmp2.fn);
                        free(prony_tmp2.theta_deg);
                        free(prony_tmp2.zeta);
        output = prony_sort(prony_tmp3);
                        free(prony_tmp3.mag);
                        free(prony_tmp3.damp);
                        free(prony_tmp3.fn);
                        free(prony_tmp3.theta_deg);
                        free(prony_tmp3.zeta);

        return output;
}

/*-------------------------------------------------------------------------*/
/*                      FONCTION(static)          eig                      */
/*-------------------------------------------------------------------------*/
static t_eig eig(t_matrix mat){

        /*Variables*/
        long int i=0,j,k,l,jj,vcc=0,nr,nc,n,lda,ldvl,ldvr,lwork,info,nvec=0;
        double **a,*a_for,*wr,*wi,*dummy,*vr,*work,*wkqr;
        char jobvl = 'N',jobvr = 'V';
        t_eig eig;
        doublecomplex *VAL;
        doublecomplex *VEC;

        nr = mat.nr;
        nc = mat.nc;
        n = nr;
        lda = nr;
        ldvl = 1;
        ldvr = nr;
        lwork = 8*nr;
        a = dmatrix(1,nr,1,nc);
        VAL = (doublecomplex*) malloc(nr*sizeof(doublecomplex));
        VEC = (doublecomplex*) malloc(nr*nr*sizeof(doublecomplex));
        a_for = malloc(nr*nc*sizeof(double));
        wr = malloc(nr*sizeof(double));
        wi = malloc(nr*sizeof(double));
        dummy = malloc(1*sizeof(double));
        vr = malloc(nr*ldvr*sizeof(double));
        work = malloc(lwork*sizeof(double));

        /*Matrix copy*/
        for (k=1;k<=nr;k++) {
                for (l=1;l<=nc;l++){
                        a[k][l] = mat.matrix[k][l];
                }
        }

        /*RC Major to CR Major*/
        for (l=1;l<=nc;l++) {
                for (k=1;k<=nr;k++){
                        a_for[i] = a[k][l];
                        i++;
                 }
        }

        /* Eigenvalue */
        dgeev_(&jobvl, &jobvr, &n, a_for, &lda, wr, wi, dummy, &ldvl, vr, &ldvr, work, &lwork, &info);

        /* Value and vector matrices */
        for (k=0;k<=nr-1;k++) {
                VAL[k].r = wr[k];
                VAL[k].i = wi[k];
        }
```

```
        for (k=1;k<=nr;k++) {
                if(VAL[k-1].i!=0){
                        jj=0;
                        for(j=1;j<=nr;j++){
                                VEC[(nr*(k-1))+(j-1)].r = vr[(nr*(k-1))+(j-1)];
                                VEC[(nr*(k-1))+(j-1)].i = vr[(nr*(k-1))+(j+(nr-1))];
                                VEC[(nr*(k-1))+(j-1)+nr].r = vr[(nr*(k-1))+(j-1)];
                                VEC[(nr*(k-1))+(j-1)+nr].i = -(vr[(nr*(k-1))+(j+(nr-1))]);
                                jj++;
                        }
                        k++;
                }

                else{
                        for(j=1;j<=nr;j++){
                                VEC[(nr*(k-1))+(j-1)].r = vr[(nr*(k-1))+(j-1)];
                                VEC[(nr*(k-1))+(j-1)].i = 0;
                        }
                }

        }

        /*Output*/
        eig.val = VAL;
        eig.val_nr = nr;
        eig.vec = VEC;
        eig.vec_nr = nr*nr;

        free(work);
        free(vr);
        free(dummy);
        free(wi);
        free(wr);
        free(a_for);
        free_dmatrix(a,1,nr,1,nc);
        return eig;
}

/*-------------------------------------------------------------------------*/
/*                      FONCTION(static)        ma_tmp                     */
/*-------------------------------------------------------------------------*/
static t_vec_comp ma_tmp(t_eig eig_vec, t_vec b_vec){

        /*Variables*/
        long int i,l,k,kp,nnr,nc,nrhs,lwork,info;
        double *b_copy;
        char trans = 'N';
        doublecomplex *eig_v,*vec_b,*work;
        t_vec_comp vec;

        nnr = eig_vec.vec_nr;
        nc = b_vec.nc;
        lwork = 2*nnr;
        nrhs = 1;
        b_copy = dvector(1,nc);
        eig_v = (doublecomplex*) malloc(nnr*sizeof(doublecomplex));
        vec_b = (doublecomplex*) malloc(nc*sizeof(doublecomplex));
        work = (doublecomplex*) malloc(lwork*sizeof(doublecomplex));

        /**Matrix and vector copy**/
        /* eig_vec */
        for (i=0;i<=(nnr-1);i++) {
                eig_v[i].r = eig_vec.vec[i].r;
                eig_v[i].i = eig_vec.vec[i].i;
        }
        /* B */
        for (l=1;l<=nc;l++){
                        b_copy[l] = b_vec.vec[l];
        }

        /**Vector offset: (C to Fortran) - (Double to complex)**/
        /* B */
        for(i=0;i<=(nc-1);i++){
                for (l=1;l<=nc;l++){
                        vec_b[i].r = b_copy[l];
                        vec_b[i].i = 0;
                        i++;
                 }
        }
```

```
        /***-------- MA Vector = pinv(eig_v) * B --------***/
        zgels_(&trans,&nc,&nc,&nrhs,eig_v,&nc,vec_b,&nc,work,&lwork,&info);

        /*Output*/
        vec.vec = vec_b;
        vec.nc = nc;

        free(work);
        free(eig_v);
        free_dvector(b_copy,1,nc);
        return vec;
}

/*-------------------------------------------------------------------------*/
/*                        FONCTION(static)          ms_tmp                 */
/*-------------------------------------------------------------------------*/
static t_vec_comp ms_tmp(t_eig eig_vec, t_vec c){

        /*Variables*/
        int i,l,k,nnr,nc,incx = 1,incy = 1;
        double *c_copy;
        char trans = 'T';
        doublecomplex *eig_v,*vec_c,*alpha,*beta,*ms;
        t_vec_comp vec;

        nnr = eig_vec.vec_nr;
        nc = c.nc;
        c_copy = dvector(1,nc);
        eig_v = (doublecomplex*) malloc(nnr*sizeof(doublecomplex));
        vec_c = (doublecomplex*) malloc(nc*sizeof(doublecomplex));
        alpha = (doublecomplex*) malloc(1*sizeof(doublecomplex));
        beta = (doublecomplex*) malloc(1*sizeof(doublecomplex));
        ms = (doublecomplex*) malloc(nc*sizeof(doublecomplex));

        /*Alpha and Beta set for zgemv*/
        alpha[0].r = 1;
        alpha[0].i = 0;
        beta[0].r = 0;
        beta[0].i = 0;

        /**Matrix and vector copy**/
        /* eig_vec */
        for (i=0;i<=(nnr-1);i++) {
                eig_v[i].r = eig_vec.vec[i].r;
                eig_v[i].i = eig_vec.vec[i].i;
        }
        /* C */
        for (l=1;l<=nc;l++){
                        c_copy[l] = c.vec[l];
        }

        /**Vector offset: (C to Fortran) - (Double to complex)**/
        /* C */
        for(i=0;i<=(nc-1);i++){
                for (l=1;l<=nc;l++){
                        vec_c[i].r = c_copy[l];
                        vec_c[i].i = 0;
                        i++;
                 }
        }

        /***-------- MS Vector = C * eig_v --------***/
        zgemv_(&trans,&nc,&nc,alpha,eig_v,&nc,vec_c,&incx,beta,ms,&incy);

        /*Output*/
        vec.vec = ms;
        vec.nc = nc;

        free(beta);
        free(alpha);
        free(vec_c);
        free(eig_v);
        free_dvector(c_copy,1,nc);
        return vec;
}
```

```
/*-------------------------------------------------------------------------*/
/*                        FONCTION(static)          r_tmp                  */
/*-------------------------------------------------------------------------*/
static t_vec_comp r_tmp(t_vec_comp ma, t_vec_comp ms){

        /*Variables*/
        int i,l,k,kp,ncomp,nc,incx = 1,incy = 1;
        char trans = 'N';
        doublecomplex *ma_mat,*ma_vec,*ms_vec,*alpha,*beta,*r_vec;
        t_vec_comp vec;

        nc = ma.nc;
        ncomp = ms.nc;
        ma_vec = (doublecomplex*) malloc(nc*sizeof(doublecomplex));
        ma_mat = (doublecomplex*) malloc(nc*nc*sizeof(doublecomplex));
        ms_vec = (doublecomplex*) malloc(nc*sizeof(doublecomplex));
        alpha = (doublecomplex*) malloc(1*sizeof(doublecomplex));
        beta = (doublecomplex*) malloc(1*sizeof(doublecomplex));
        r_vec = (doublecomplex*) malloc(nc*sizeof(doublecomplex));

        /*Alpha and Beta set for zgemv*/
        alpha[0].r = 1;
        alpha[0].i = 0;
        beta[0].r = 0;
        beta[0].i = 0;

        /**Vector copy**/
        /* MA */
        for (i=0;i<=(nc-1);i++) {
                ma_vec[i].r = ma.vec[i].r;
                ma_vec[i].i = ma.vec[i].i;
        }

        /* MS */
        for (i=0;i<=(nc-1);i++) {
                ms_vec[i].r = ms.vec[i].r;
                ms_vec[i].i = ms.vec[i].i;
        }

        /*Matrix for Array product*/
        for(k=0;k<=(nc*nc)-1;k++){
                ma_mat[k].r = 0;
                ma_mat[k].i = 0;
        }

        for(k=0;k<nc;k++){
                ma_mat[k*(nc+1)].r = ma_vec[k].r;
                ma_mat[k*(nc+1)].i = ma_vec[k].i;
        }

        /***-------- Residue Vector = MA .* MS (Array product) --------***/
        zgemv_(&trans,&nc,&nc,alpha,ma_mat,&nc,ms_vec,&incx,beta,r_vec,&incy);

        /*Output*/
        vec.vec = r_vec;
        vec.nc = nc;

        free(beta);
        free(alpha);
        free(ms_vec);
        free(ma_mat);
        free(ma_vec);
        return vec;
}

/*-------------------------------------------------------------------------*/
/*                        FONCTION(static)          mag                    */
/*-------------------------------------------------------------------------*/
static t_vec mag(t_vec_comp r_tmp){

        /*Variables*/
        int i,l,nc;
        double *mag;
        doublecomplex *r_vec;
        t_vec vec;

        nc = r_tmp.nc;
        mag = dvector(1,nc);
        r_vec = (doublecomplex*) malloc(nc*sizeof(doublecomplex));
```

```
        /**Vector copy**/
        /* Residue */
        for (i=0;i<=(nc-1);i++) {
                r_vec[i].r = r_tmp.vec[i].r;
                r_vec[i].i = r_tmp.vec[i].i;
        }

        /*Magnitude*/
        for(l=1;l<=nc;l++){
                mag[l] = 2*sqrt(((r_vec[l-1].r)*(r_vec[l-1].r))+((r_vec[l-1].i)*(r_vec[l-1].i)));
        }

        /*Output*/
        vec.vec = mag;
        vec.nc = nc;

        free(r_vec);
        return vec;
}

/*-------------------------------------------------------------------------------*/
/*                        FONCTION(static)          damp                         */
/*-------------------------------------------------------------------------------*/
static t_vec damp(t_eig eig_val){

        /*Variables*/
        int l,nr;
        double *damp;
        t_vec vec;

        nr = eig_val.val_nr;
        damp = dvector(1,nr);

        /*Damping Factor*/
        for (l=1;l<=nr;l++) {
                damp[l] = eig_val.val[l-1].r;
                }

        /*Output*/
        vec.vec = damp;
        vec.nc = nr;

        return vec;
}

/*-------------------------------------------------------------------------------*/
/*                        FONCTION(static)          fn                           */
/*-------------------------------------------------------------------------------*/
static t_vec fn(t_eig eig_val){

        /*Variables*/
        int i,l,nr;
        double *fn;
        doublecomplex *val;
        t_vec vec;

        nr = eig_val.val_nr;
        fn = dvector(1,nr);
        val = (doublecomplex*) malloc(nr*sizeof(doublecomplex));

        /**Vector copy**/
        /* VAL */
        for (i=0;i<=(nr-1);i++) {
                val[i].r = eig_val.val[i].r;
                val[i].i = eig_val.val[i].i;
        }

        /*Oscillation Frequency(Hz)*/
        for(l=1;l<=nr;l++){
                fn[l] = (val[l-1].i)/(2*M_PI);
        }

        /*Output*/
        vec.vec = fn;
        vec.nc = nr;

        free(val);
        return vec;}
```

```
/*-------------------------------------------------------------------------*/
/*                        FONCTION(static)          theta_deg              */
/*-------------------------------------------------------------------------*/
static t_vec theta_deg(t_vec_comp r_tmp){

        /*Variables*/
        int i,l,nc;
        double *theta;
        doublecomplex *r_vec;
        t_vec vec;

        nc = r_tmp.nc;
        theta = dvector(l,nc);
        r_vec = (doublecomplex*) malloc(nc*sizeof(doublecomplex));

        /**Vector copy**/
        /* R */
        for (i=0;i<=(nc-l);i++) {
                r_vec[i].r = r_tmp.vec[i].r;
                r_vec[i].i = r_tmp.vec[i].i;
                printf("Residu = %f+j%f\n ", r_vec[i].r, r_vec[i].i);
        }

        /*Phase Angle (deg)*/
        for(l=l;l<=nc;l++){
                theta[l] = (atan2(r_vec[l-l].i,r_vec[l-l].r))*(180/M_PI);
        }

        /*Output*/
        vec.vec = theta;
        vec.nc = nc;

        free(r_vec);
        return vec;
}


/*-------------------------------------------------------------------------*/
/*                        FONCTION(static)          zeta                   */
/*-------------------------------------------------------------------------*/
static t_vec zeta(t_vec fn, t_vec damp){

        /*Variables*/
        int l,nc;
        double *wn, *dn, *zeta;
        t_vec vec;

        nc = damp.nc;
        wn = dvector(l,nc);
        dn = dvector(l,nc);
        zeta = dvector(l,nc);

        /*Frequency (wn)*/
        for(l=l;l<=nc;l++){
                wn[l] = fn.vec[l]*(2*M_PI);
        }

        /*Damping (dn)*/
        for(l=l;l<=nc;l++){
                dn[l] = damp.vec[l];
        }

        /*Damping ratio (zeta)*/
        for(l=l;l<=nc;l++){
                zeta[l] = (-(dn[l]))/(sqrt((dn[l]*dn[l])+(wn[l]*wn[l])));
        }

        /*Output*/
        vec.vec = zeta;
        vec.nc = nc;

        free_dvector(dn,l,nc);
        free_dvector(wn,l,nc);
        return vec;
}
```

```
/*-------------------------------------------------------------------------*/
/*                      FONCTION(static)          prony_cc                 */
/*-------------------------------------------------------------------------*/
static t_prony prony_cc(t_prony tmp){

        /*Variables*/
        int l,nbr,cpt=1, cpt2=1;
        t_prony prony;

        nbr = tmp.qty;

        /*Complex Conjugate count*/
        l=1;
        while(l<=nbr){
                if(l==nbr){
                        cpt++;
                        l++;
                }
                else if(degal(tmp.damp[l], tmp.damp[l+1]) == 1 ){
                        cpt++;
                        l=l+2;
                }
                else if(degal(tmp.damp[l], tmp.damp[l+1]) != 1 ){
                        cpt++;
                        l++;
                }
        }

        prony.mag = (double*) malloc((cpt+1)*sizeof(double));
        prony.damp = (double*) malloc((cpt+1)*sizeof(double));
        prony.fn = (double*) malloc((cpt+1)*sizeof(double));
        prony.theta_deg = (double*) malloc((cpt+1)*sizeof(double));
        prony.zeta = (double*) malloc((cpt+1)*sizeof(double));

        l=1;
        while(l<=nbr){
                if(l==nbr){
                        prony.mag[cpt2] = tmp.mag[l];
                        prony.damp[cpt2] = tmp.damp[l];
                        prony.fn[cpt2] = tmp.fn[l];
                        prony.theta_deg[cpt2] = tmp.theta_deg[l];
                        prony.zeta[cpt2] = tmp.zeta[l];
                        cpt2++;
                        l++;
                }
                else if(degal(tmp.damp[l], tmp.damp[l+1]) == 1){
                        prony.mag[cpt2] = tmp.mag[l];
                        prony.damp[cpt2] = tmp.damp[l];
                        prony.fn[cpt2] = tmp.fn[l];
                        prony.theta_deg[cpt2] = tmp.theta_deg[l];
                        prony.zeta[cpt2] = tmp.zeta[l];
                        cpt2++;
                        l=l+2;
                }
                else if(degal(tmp.damp[l], tmp.damp[l+1]) != 1){
                        prony.mag[cpt2] = tmp.mag[l];
                        prony.damp[cpt2] = tmp.damp[l];
                        prony.fn[cpt2] = tmp.fn[l];
                        prony.theta_deg[cpt2] = tmp.theta_deg[l];
                        prony.zeta[cpt2] = tmp.zeta[l];
                        cpt2++;
                        l++;
                }
        }

        prony.qty = cpt2-1;
        return prony;
}

/*-------------------------------------------------------------------------*/
/*                      FONCTION(static)          prony_bp                 */
/*-------------------------------------------------------------------------*/
static t_prony prony_bp(t_prony tmp, double f_low, double f_high){

        /*Variables*/
        int l,nbr,cpt=1,cpt2=1;
        t_prony prony;

        nbr = tmp.qty;
```

```
        /*Band limitation*/
        for(l=1;l<=nbr;l++){
                if( (tmp.fn[l] > f_low || degal(tmp.fn[l],f_low)) && (tmp.fn[l] < f_high ||
                degal(tmp.fn[l],f_high))){
                        cpt++;
                }
        }

        prony.mag = (double*) malloc((cpt+1)*sizeof(double));
        prony.damp = (double*) malloc((cpt+1)*sizeof(double));
        prony.fn = (double*) malloc((cpt+1)*sizeof(double));
        prony.theta_deg = (double*) malloc((cpt+1)*sizeof(double));
        prony.zeta = (double*) malloc((cpt+1)*sizeof(double));

        for(l=1;l<=nbr;l++){
                if((tmp.fn[l] > f_low || degal(tmp.fn[l],f_low)) && (tmp.fn[l] < f_high ||
                degal(tmp.fn[l],f_high))){
                        prony.mag[cpt2] = tmp.mag[l];
                        prony.damp[cpt2] = tmp.damp[l];
                        prony.fn[cpt2] = tmp.fn[l];
                        prony.theta_deg[cpt2] = tmp.theta_deg[l];
                        prony.zeta[cpt2] = tmp.zeta[l];
                        cpt2++;
                }
        }

        prony.qty = cpt2-1;
        return prony;
}

/*-------------------------------------------------------------------------*/
/*                      FONCTION(static)          prony_ins                */
/*-------------------------------------------------------------------------*/
static t_prony prony_ins(t_prony tmp){

        /*Variables*/
        int l,nbr,cpt=1,cpt2=1;
        t_prony prony;

        nbr = tmp.qty;

        /*Resonant mode limitation*/
        for(l=1;l<=nbr;l++){
                printf("Damp: %lf\n",tmp.damp[l] );
                if( tmp.damp[l] < 0 || degal(tmp.damp[l],0.0) == 1 || tmp.damp[l] > 0.001 ){
                        printf("Damp <=0.0\n");
                        cpt++;
                }
        }

        prony.mag = (double*) malloc((cpt+1)*sizeof(double));
        prony.damp = (double*) malloc((cpt+1)*sizeof(double));
        prony.fn = (double*) malloc((cpt+1)*sizeof(double));
        prony.theta_deg = (double*) malloc((cpt+1)*sizeof(double));
        prony.zeta = (double*) malloc((cpt+1)*sizeof(double));

        for(l=1;l<=nbr;l++){
                if( tmp.damp[l] < 0 || degal(tmp.damp[l],0.0) == 1 || tmp.damp[l] > 0.001 ){
                        prony.mag[cpt2] = tmp.mag[l];
                        prony.damp[cpt2] = tmp.damp[l];
                        prony.fn[cpt2] = tmp.fn[l];
                        prony.theta_deg[cpt2] = tmp.theta_deg[l];
                        prony.zeta[cpt2] = tmp.zeta[l];
                        cpt2++;
                }
        }

        prony.qty = cpt2-1;
        return prony;
}

/*-------------------------------------------------------------------------*/
/*                      FONCTION(static)          prony_zlim               */
/*-------------------------------------------------------------------------*/
static t_prony prony_zlim(t_prony tmp, double zeta_limit){

        /*Variables*/
        int l,nbr,cpt=1, cpt2=1;
        t_prony prony;
```

```
        nbr = tmp.qty;

        /*User input validation*/
        if(zeta_limit < 0 || zeta_limit > 0.707){
                zeta_limit = 0.707;
        }

        /*Zeta limitation*/
        for(l=1;l<=nbr;l++){
                if(tmp.zeta[l] < zeta_limit){
                        cpt++;
                }
        }

        prony.mag = (double*) malloc((cpt+1)*sizeof(double));
        prony.damp = (double*) malloc((cpt+1)*sizeof(double));
        prony.fn = (double*) malloc((cpt+1)*sizeof(double));
        prony.theta_deg = (double*) malloc((cpt+1)*sizeof(double));
        prony.zeta = (double*) malloc((cpt+1)*sizeof(double));

        for(l=1;l<=nbr;l++){
                if(tmp.zeta[l] < zeta_limit){
                        prony.mag[cpt2] = tmp.mag[l];
                        prony.damp[cpt2] = tmp.damp[l];
                        prony.fn[cpt2] = tmp.fn[l];
                        prony.theta_deg[cpt2] = tmp.theta_deg[l];
                        prony.zeta[cpt2] = tmp.zeta[l];
                        cpt2++;
                }
        }

        prony.qty = cpt2-1;
        return prony;
}

/*-------------------------------------------------------------------------*/
/*                      FONCTION(static)        prony_alim                 */
/*-------------------------------------------------------------------------*/
static t_prony prony_alim(t_prony tmp, double amp_limit){

        /*Variables*/
        int l,nbr,cpt=1, cpt2=1;
        double mode_ref;
        t_prony prony;

        nbr = tmp.qty;

        /*User input validation*/
        if(amp_limit < 1 || amp_limit > 100){
                amp_limit = 100;
        }

        /*Predominant mode*/
        mode_ref = tmp.mag[1];
        for(l=2;l<=nbr;l++){
                if(tmp.mag[l] > mode_ref/*tmp.mag[l-1]*/){
                        mode_ref = tmp.mag[l];
                }
        }

        /*Amplitude limitation*/
        for(l=1;l<=nbr;l++){
                if(tmp.mag[l] > (mode_ref/amp_limit) ||
                degal(tmp.mag[l], (mode_ref/amp_limit)) == 1 ){
                        cpt++;
                }
        }

        prony.mag = (double*) malloc((cpt+1)*sizeof(double));
        prony.damp = (double*) malloc((cpt+1)*sizeof(double));
        prony.fn = (double*) malloc((cpt+1)*sizeof(double));
        prony.theta_deg = (double*) malloc((cpt+1)*sizeof(double));
        prony.zeta = (double*) malloc((cpt+1)*sizeof(double));
```

```
        for(l=1;l<=nbr;l++){
                if(tmp.mag[l] >= (mode_ref/amp_limit)){
                        prony.mag[cpt2] = tmp.mag[l];
                        prony.damp[cpt2] = tmp.damp[l];
                        prony.fn[cpt2] = tmp.fn[l];
                        prony.theta_deg[cpt2] = tmp.theta_deg[l];
                        prony.zeta[cpt2] = tmp.zeta[l];
                        cpt2++;
                }
        }

        prony.qty = cpt2-1;
        return prony;
}

/*-------------------------------------------------------------------------*/
/*                      FONCTION(static)          prony_dc                 */
/*-------------------------------------------------------------------------*/
static t_prony prony_dc(t_prony tmp){

        /*Variables*/
        int l,nbr,cpt=1, cpt2=1;
        double mode_ref;
        t_prony prony;

        nbr = tmp.qty;

        /*DC limitation*/
        for(l=1;l<=nbr;l++){
                if( degal(tmp.fn[l],0.0) != 1 ){
                        cpt++;
                }
        }

        prony.mag = (double*) malloc((cpt+1)*sizeof(double));
        prony.damp = (double*) malloc((cpt+1)*sizeof(double));
        prony.fn = (double*) malloc((cpt+1)*sizeof(double));
        prony.theta_deg = (double*) malloc((cpt+1)*sizeof(double));
        prony.zeta = (double*) malloc((cpt+1)*sizeof(double));

        for(l=1;l<=nbr;l++){
                if( degal(tmp.fn[l],0.0) != 1 ){
                        prony.mag[cpt2] = tmp.mag[l];
                        prony.damp[cpt2] = tmp.damp[l];
                        prony.fn[cpt2] = tmp.fn[l];
                        prony.theta_deg[cpt2] = tmp.theta_deg[l];
                        prony.zeta[cpt2] = tmp.zeta[l];
                        cpt2++;
                }
        }

        prony.qty = cpt2-1;
        return prony;
}

/*-------------------------------------------------------------------------*/
/*                      FONCTION(static)          prony_sort               */
/*-------------------------------------------------------------------------*/
static t_prony prony_sort(t_prony tmp){

        /*Variables*/
        int l,nbr,k,p,ii,cpt=1, cpt2=1, *flag, *ind;
        double mode_pre, mode_ref;
        t_prony sort;

        nbr = tmp.qty;
        sort.mag = (double*) malloc((nbr+1)*sizeof(double));
        sort.damp = (double*) malloc((nbr+1)*sizeof(double));
        sort.fn = (double*) malloc((nbr+1)*sizeof(double));
        sort.theta_deg = (double*) malloc((nbr+1)*sizeof(double));
        sort.zeta = (double*) malloc((nbr+1)*sizeof(double));
        flag = ivector(1,nbr);
        ind = ivector(1,nbr);
```

```
        /*Flag*/
        for(l=1;l<=nbr;l++){
                flag[l] = 0;
        }

        /*Index*/
        mode_ref = 0;
        for(k=1;k<=nbr;k++){
                for(l=1;l<=nbr;l++){
                        if(tmp.mag[l] > mode_ref && flag[l]!=1){
                                mode_ref = tmp.mag[l];
                                ii=l;
                        }
                }
                ind[k] = ii;
                flag[ii] = 1;
                mode_ref = 0;
        }

        /*Sorting*/
        for(l=1;l<=nbr;l++){
                sort.mag[l] = tmp.mag[ind[l]];
                sort.damp[l] = tmp.damp[ind[l]];
                sort.fn[l] = tmp.fn[ind[l]];
                sort.theta_deg[l] = tmp.theta_deg[ind[l]];
                sort.zeta[l] = tmp.zeta[ind[l]];
        }

        sort.qty = nbr;

        free_ivector(ind,1,nbr);
        free_ivector(flag,1,nbr);
        return sort;
}
```

ANNEXE VII

Déclaration des fonctions du fichier DATA.H

```
/*****************************************************************************/
/* PAR     : Mathieu Perron        (ETS - GREPCI)                            */
/* FICHIER : data.h                                                          */
/* DATE    : 8 avril 2010                                                    */
/*****************************************************************************/
/*                                                                           */
/* Ce fichier inclut la declaration des fonctions qui valident               */
/* les arguments d'entree et optimisent le signal pour l'analyse             */
/* modale ERA/Prony.                                                         */
/*                                                                           */
/*===========================================================================*/
/*===========================================================================*/
#ifndef DATA_H
#define DATA_H
#include "util.h"

typedef struct{

        double *data;
        int count;

} t_data_dec;

typedef struct{

        int dec;
        int nr;
        int nc;
        double t_ech_dec;

} t_data_inf;

typedef struct{

        int nr;
        int nc;

} t_info_chk;

/*===========================================================================*/
/*---------------------------------------------------------------------------*/
/*                        DECLARATION DES FONCTIONS                          */
/*                                                                           */
/*===========================================================================*/
/*===========================================================================*/

/*---------------------------------------------------------------------------*/
/*                        FONCTION         data_chk                          */
/*---------------------------------------  ------------------------------------*/
/*                                                                           */
/*      La fonction data_chk permet de signaler des irregularites            */
/*      au niveau des valeurs des parametres trig, f_low et f_high.          */
/*      RETOUR : - Un signal d'arret dans le cas d'une irregularite.         */
/*      PARAMETRES : La taille de l'echantillon,                             */
/*                 - la periode d'echantillonnage,                           */
/*                 - la limite de frequence basse (>=0.05Hz),                */
/*                 - la limite de frequence haute,                           */
/*                 - le temps de debut de l'etude,                           */
/*                                                                           */
/*---------------------------------------------------------------------------*/
void data_chk(int ndata, double t_ech, double f_low, double f_high, double trig, t_amod_msg *msg);
```

```
/*-------------------------------------------------------------------------*/
/*                        FONCTION         data_info                       */
/*-------------------------------------------------------------------------*/
/*                                                                         */
/*       La fonction data_info genere les informations optimales sur la    */
/*       decimation et la taille de la matrice de Hankel en fonction       */
/*       des arguments trig, f_low et f_high et permet d'informer          */
/*       l'utilisateur sur des irregularites de sous-echantillonnage       */
/*       ou de donnees insuffisantes.                                      */
/*       RETOUR : - Le taux optimal de decimation,                         */
/*                - la taille optimale de la matrice de Hankel,            */
/*                - la periode d'echantillonnage optimale,                 */
/*                - Un signal d'avertissement pour irregularite mineure,   */
/*                - Un signal d'arret pour irregularite majeure.           */
/*       PARAMETRES : La taille de l'echantillon,                          */
/*                - la periode d'echantillonnage,                          */
/*                - la limite de frequence basse (>=0.05Hz),               */
/*                - la limite de frequence haute,                          */
/*                - le temps de debut de l'etude.                          */
/*                                                                         */
/*-------------------------------------------------------------------------*/
t_data_inf data_info(int ndata, double t_ech, double f_low, double f_high, double trig, t_amod_msg *msg);

/*-------------------------------------------------------------------------*/
/*                        FONCTION         data_dec                        */
/*-------------------------------------------------------------------------*/
/*                                                                         */
/*       La fonction data_dec permet de decimer le signal echantillonne    */
/*       selon un taux specifique de decimation.                           */
/*       RETOUR : - Le signal decime et sa taille.                         */
/*       PARAMETRES : La signal echantillonne,                             */
/*                - la taille de l'echantillon,                            */
/*                - le taux de decimation,                                 */
/*                - la periode d'echantillonnage,                          */
/*                - le temps de debut de l'etude.                          */
/*                                                                         */
/*-------------------------------------------------------------------------*/
t_data_dec data_dec(double *real_sel, int count, double DEC, double TECH, double TRIG);

/*-------------------------------------------------------------------------*/
/*                        FONCTION(static)          info_chk               */
/*-------------------------------------------------------------------------*/
/*                                                                         */
/*       La fonction info_chk permet de valider la taille de la matrice    */
/*       de Hankel et de l'ajuster en fonction des limites du signal.      */
/*       RETOUR : - La taille optimale de la matrice de Hankel.            */
/*       PARAMETRES : La taille de l'echantillon,                          */
/*                - la taille de l'echantillon,                            */
/*                - le taux de decimation,                                 */
/*                - la periode d'echantillonnage,                          */
/*                - le temps de debut de l'etude.                          */
/*                                                                         */
/*-------------------------------------------------------------------------*/
static t_info_chk info_chk(int ndata, double t_ech, double trig, int dec, int nr_max,
int nc_max, double t_ech_dec, t_amod_msg *msg);

#endif
```

ANNEXE VIII

Définition des fonctions du fichier DATA.C

```
/*****************************************************************************/
/* PAR     : Mathieu Perron        (ETS - GREPCI)                            */
/* FICHIER : data.c                                                          */
/* DATE    : 8 avril 2010                                                    */
/*****************************************************************************/
/*                                                                           */
/* Ce fichier inclut la definition des fonctions qui valident                */
/* les arguments d'entree et optimisent le signal pour l'analyse             */
/* modale ERA/Prony.                                                         */
/*                                                                           */
/*===========================================================================*/
/*===========================================================================*/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <malloc.h>
#include <string.h>
#include <assert.h>
#include <sys/param.h>
#include "nr.h"
#include "nrutil.h"
#include "data.h"

/*===========================================================================*/
/*---------------------------------------------------------------------------*/
/*                          DEFINITION DES FONCTIONS                         */
/*---------------------------------------------------------------------------*/
/*===========================================================================*/
/*===========================================================================*/

/*---------------------------------------------------------------------------*/
/*                          FONCTION         data_chk                        */
/*---------------------------------------------------------------------------*/
void data_chk(int ndata, double rate, double f_low, double f_high, double trig, t_amod_msg *msg){

        /*Variables*/
        int l,nr,nc,stop = 0, ndata_trig;

        ndata_trig = (int)floor(ndata-(trig*(1/rate)));

        if(ndata_trig <= 0 || trig < 0.0){
                appendMsg(&(msg->errmsg), "Arret: Valeur de trigger hors signal\n");
                msg->errcode = 1;
        }

        if(f_low > f_high || degal(f_low,f_high) == 1  || f_low < 0.05 || f_high < 0.0 ){
                appendMsg(&(msg->errmsg), "Arret: Valeurs de frequences non conformes\n");
                msg->errcode = 1;
        }
}

/*---------------------------------------------------------------------------*/
/*                          FONCTION         data_info                       */
/*---------------------------------------------------------------------------*/
t_data_inf data_info(int ndata, double rate, double f_low, double f_high, double trig, t_amod_msg *msg){

        /* Variables */
        int l, dec_max, dec, nr_max, nr, nc, warning = 0, stop = 0;
        double rate_dec;
        t_info_chk chk;
        t_data_inf info;
        char buff[256];

        printf("rate %lf\n", rate);
        fflush(stdout);

        /* Hankel and decimate sizing */
        dec_max = (int)floor(1/(4*f_high*rate));
        nr_max = (int)ceil(((1/f_low)*(1/(rate)))/(2.0));
```

```
        if(nr_max < 50){
                appendMsg(&(msg->errmsg), "Arret: Sous-echantillonnage\n");
                msg->errcode = 1;
        }

        else if(nr_max <= 1000){
                        if(dec_max == 0){
                                dec = 1;
                                rate_dec = rate;
                                nr = nc = nr_max;
                                sprintf(buff,"Avertissement: Etude limitee entre (%lfHz <-> %lfHz)\n",
                                                        1/(rate_dec*2.0*nr),1/(rate_dec*4.0));
                                appendMsg(&(msg->warnmsg), buff);
                                msg->warncode = 1;
                        }

                        else {

                                dec = 1;
                                rate_dec = rate;
                                nr = nc = nr_max;
                        }
        }

        else if(nr_max > 1000){
                if(dec_max == 0){
                        dec = 1;
                        rate_dec = rate;
                        nr = nc = 1000;
                        sprintf(buff,"Avertissement: Etude limitee entre (%lfHz <-> %lfHz)\n",
                                                        1/(rate_dec*2.0*nr),1/(rate_dec*4.0));
                        appendMsg(&(msg->warnmsg), buff);
                        msg->warncode = 1;
                }

                else {
                        for(dec=1; dec <= dec_max; dec++){
                                if( ((int)ceil(((1/f_low)*(1/(rate*dec)))/(2.0)) <= 1000) &&
                                        (dec <= 3) ){
                                        rate_dec = rate * dec;
                                        nr = nc = (int)ceil(((1/f_low)*(1/(rate_dec)))/(2.0));
                                        break;
                                }

                                else if( ((int)ceil(((1/f_low)*(1/(rate*dec)))/(2.0)) <= 1000) &&
                                       (dec > 3) ){
                                        rate_dec = rate * dec;
                                        nr = nc = (int)ceil(((1/f_low)*(1/(rate_dec)))/(2.0));
                                        appendMsg(&(msg->warnmsg),
                                                        "Avertissement: Sur-echantillonnage\n");
                                        msg->warncode = 1;

                                }
                                else if( dec == dec_max ){
                                        rate_dec = rate * dec;
                                        nr = nc = 1000;
                                        sprintf(buff,
                                                        "Avertissement: Etude limitee entre (%lfHz <->
                                                         %lfHz)\n",
                                                                1/(rate_dec*2.0*nr),1/(rate_dec*4.0));
                                        appendMsg(&(msg->warnmsg), buff);
                                        msg->warncode = 1;

                                }
                        }
                }
        }

        chk = info_chk(ndata, rate, trig, dec, nr, nc, rate_dec, msg);

        /*Output*/
        info.dec = dec;
        info.nr = chk.nr;
        info.nc = chk.nc;
        info.rate_dec = rate_dec;

        return info;
}
```

```
/*-------------------------------------------------------------------------*/
/*                        FONCTION(static)          info_chk                */
/*-------------------------------------------------------------------------*/
static t_info_chk info_chk(int ndata, double rate, double trig, int dec, int nr_max,
int nc_max, double rate_dec, t_amod_msg *msg){

        /*Variables*/
        int l,nr,nc, stop = 0, warning = 0, ndata_trig;
        t_info_chk chk;
        char buff[256];

        nr = nr_max;
        nc = nc_max;
        ndata_trig = (int)floor(ndata-(trig*(1/rate)));

        if(dec*(nr+nc) > ndata_trig){
                while(dec*(nr+nc) > ndata_trig){
                        nr--;
                        nc--;
                        if(nr+nc <= 98){
                                appendMsg(&(msg->errmsg), "Arret: Donnees insuffisantes\n");
                                msg->errcode = 1;
                        }
                }

                sprintf(buff,"Avertissement4: Etude limitee entre (%lfHz <-> %lfHz)\n",
                                1/(rate_dec*(nr+nc)),1/(rate_dec*4.0));
                appendMsg(&(msg->warnmsg), buff);
                msg->warncode = 1;

        }

        /*Output*/
        chk.nr = nr;
        chk.nc = nc;

        return chk;
}

/*-------------------------------------------------------------------------*/
/*                        FONCTION          data_dec                      */
/*-------------------------------------------------------------------------*/
t_data_dec data_dec(double *real_sel, int count, double DEC, double TECH, double TRIG){

        /*Variables*/
        int l,k,nbr,cut,ech_trig,ech_dec;
        double *data_trig,*data_dec;
        t_data_dec output;

        nbr = count;
        cut = (TRIG/TECH);
        ech_trig = (nbr-cut);
        ech_dec = (ech_trig/DEC);
        data_trig = dvector(1,ech_trig);
        data_dec = dvector(1,ech_dec);

        /*Trig data*/
        for(l=0;l<ech_trig;l++){
                data_trig[l+1] = real_sel[l+(cut)];
        }

        /*Decimate data*/
        k = 1;
        for(l=1;l<=ech_dec;l++){
                data_dec[l] = data_trig[k];
                k = k+DEC;
        }

        /*Output*/
        output.data = data_dec;
        output.count = ech_dec;

        free_dvector(data_trig,1,ech_trig);
        return output;
}
```

ANNEXE IX

Déclaration des fonctions du fichier YMODEL.H

```
/*****************************************************************************/
/* PAR     : Mathieu Perron        (ETS - GREPCI)                            */
/* FICHIER : ymodel.h                                                        */
/* DATE    : 8 avril 2010                                                    */
/*****************************************************************************/
/*                                                                           */
/* Ce fichier inclut la declaration de la fonction qui permet de             */
/* reconstruire le signal temporel analyse a partir des parametres           */
/* de Prony obtenus de apron_tot.                                            */
/*                                                                           */
/*===========================================================================*/
/*===========================================================================*/
#ifndef YMODEL_H
#define YMODEL_H
#include "util.h"

/*===========================================================================*/
/*---------------------------------------------------------------------------*/
/*                          DECLARATION DES FONCTIONS                        */
/*---------------------------------------------------------------------------*/
/*===========================================================================*/
/*===========================================================================*/

/*---------------------------------------------------------------------------*/
/*                          FONCTION         ymodel                          */
/*---------------------------------------------------------------------------*/
/*                                                                           */
/*       La fonction ymodel permet de reconstruire le signal temporel        */
/*       analyse a partir des parametres de Prony obtenus de apron_tot       */
/*       d'apres : ymodel(k) = a_k*exp(-sig_k*k*dt)*cos(2*pi*fn_k*k*dt+th_k). */
/*       RETOUR : - Le signal temporel reconstruit.                          */
/*       PARAMETRES : Les parametres de Prony retenus par amod_tot,          */
/*                - la nombre de modes,                                      */
/*                - la periode d'echantillonnage ajustee,                    */
/*                - le temps de debut de l'etude.                            */
/*                                                                           */
/*---------------------------------------------------------------------------*/
t_vec ymodel(t_prony prony, int count, double tech, double trig);

#endif
```

ANNEXE X

Définition des fonctions du fichier YMODEL.C

```
/*****************************************************************************/
/* PAR     : Mathieu Perron        (ETS - GREPCI)                            */
/* FICHIER : ymodel.c                                                        */
/* DATE    : 8 avril 2010                                                    */
/*****************************************************************************/
/*                                                                           */
/* Ce fichier inclut la definition de la fonction qui permet de              */
/* reconstruire le signal temporel analyse a partir des parametres           */
/* de Prony obtenus de apron_tot.                                            */
/*                                                                           */
/*===========================================================================*/
/*===========================================================================*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <malloc.h>
#include <assert.h>
#include "f2c.h"
#include "clapack.h"
#include "nr.h"
#include "nrutil.h"
#include "ymodel.h"
#define NRANSI

/*===========================================================================*/
/*---------------------------------------------------------------------------*/
/*                          DEFINITION DES FONCTIONS                         */
/*---------------------------------------------------------------------------*/
/*===========================================================================*/
/*===========================================================================*/

/*---------------------------------------------------------------------------*/
/*                          FONCTION          ymodel                         */
/*---------------------------------------------------------------------------*/
t_vec ymodel(t_prony prony, int count, double tech, double trig){

        /*Variables*/
        int i,l,nbr,nr,cut,qty;
        double *y, *t;
        t_vec ymodel;
        FILE *y_out;

        nbr = count;
        cut = (trig/tech);
        nr = (nbr-cut);
        qty = prony.qty;
        y = dvector(1,nr);
        t = dvector(1,nr);

        /*Sample vector*/
        t[1] = 0;
        for(l=2;l<=nr;l++){
                t[l] = t[l-1]+tech;
        }

        /*Sinusoidal summation*/
        for(i=1;i<=qty;i++){
                if((prony.fn[i]*2*M_PI) == 0.0){
                        prony.mag[i] = prony.mag[i]/2;
                }
        }
```

```
        for(l=1;l<=nr;l++){
                y[l] = 0;
                for(i=1;i<=qty;i++){
                        y[l] += (prony.mag[i] * exp(prony.damp[i]*t[l]) * cos( (prony.fn[i]*2*M_PI *
                                t[l]) + (prony.theta_deg[i]*(M_PI/180.0)) ) );
                }
        }

        /*Output*/
        ymodel.vec = y;
        ymodel.nc = nr;

        free_dvector(t,1,nr);
        return ymodel;
}
```

ANNEXE XI

Déclaration des fonctions du fichier UTIL.H

```
/*****************************************************************************/
/* PAR     : Mathieu Perron        (ETS - GREPCI)                            */
/* FICHIER : util.h                                                          */
/* DATE    : 8 avril 2010                                                    */
/*****************************************************************************/
/*                                                                           */
/* Ce fichier inclut la declaration des structures et fonctions              */
/* utilitaires.                                                              */
/*                                                                           */
/*===========================================================================*/
/*===========================================================================*/
#ifndef UTIL_H
#define UTIL_H
#include "f2c.h"

typedef struct{

        double *vec;
        int nc;

} t_vec;

typedef struct{

        double **matrix;
        int nr;
        int nc;
        int tmax;

} t_matrix;

typedef struct{

        double *mag;
        double *damp;
        double *fn;
        double *theta_deg;
        double *zeta;
        int qty;

} t_prony;

typedef struct{

        char* warnmsg;
        char* errmsg;
        int warncode;
        int errcode;

} t_amod_msg;

/*===========================================================================*/
/*---------------------------------------------------------------------------*/
/*                        DECLARATION DES FONCTIONS                          */
/*---------------------------------------------------------------------------*/
/*===========================================================================*/
/*===========================================================================*/

/*---------------------------------------------------------------------------*/
/*                        FONCTION         degal                             */
/*---------------------------------------------------------------------------*/
/*                                                                           */
/*       La fonction degal permet d'evaluer l'egalite de deux nombres        */
/*       reels a un degre de precision fixe a 1e-5.                          */
/*       RETOUR : 1 si egaux.                                                */
/*       PARAMETRES : Les deux nombres reels                                 */
/*                                                                           */
/*---------------------------------------------------------------------------*/
int degal(double v1, double v2);
```

```
/*-------------------------------------------------------------------------*/
/*                          FONCTION         appendMsg                     */
/*-------------------------------------------------------------------------*/
/*                                                                         */
/*       La fonction appendMsg permet de gerer le buffer contenant         */
/*       les messages d'avertissements et d'erreurs.                       */
/*       RETOUR : aucun                                                    */
/*       PARAMETRES : Un buffer de messages, un message                    */
/*                                                                         */
/*-------------------------------------------------------------------------*/
void appendMsg(char** msgBuf, char* msg);

#endif
```

ANNEXE XII

Définition des fonctions du fichier UTIL.C

```
/*****************************************************************************/
/* PAR     : Mathieu Perron        (ETS - GREPCI)                            */
/* FICHIER : util.C                                                          */
/* DATE    : 8 avril 2010                                                    */
/*****************************************************************************/
/*                                                                           */
/* Ce fichier inclut la definition des fonctions utilitaires.                */
/*                                                                           */
/*===========================================================================*/
/*===========================================================================*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <malloc.h>
#include <assert.h>
#include <sys/param.h>
#include "util.h"

/*===========================================================================*/
/*---------------------------------------------------------------------------*/
/*                        DEFINITION DES FONCTIONS                           */
/*---------------------------------------------------------------------------*/
/*===========================================================================*/
/*===========================================================================*/

/*---------------------------------------------------------------------------*/
/*                        FONCTION         degal                             */
/*---------------------------------------------------------------------------*/
int degal(double v1, double v2){

        /*Variables*/
        double eps = 1e-5;

        return (fabs(v2-v1) <= MAX(fabs(v2),fabs(v1))*eps);
}

/*---------------------------------------------------------------------------*/
/*                        FONCTION         appendMsg                         */
/*---------------------------------------------------------------------------*/
void appendMsg(char** msgBuf, char* msg){

        /*Variables*/
        int msglen;

        if ( msg == 0 || strlen(msg) == 0){
                return;
        }

        msglen = strlen(msg);

        if (*msgBuf == 0){
                *msgBuf = (char*)malloc((msglen + 1) * sizeof(char));
        }
        else{
                *msgBuf = (char*)realloc(*msgBuf, (strlen(*msgBuf) + msglen + 1) * sizeof(char));
       }

        strcpy(*msgBuf, msg);
}
```

ANNEXE XIII

Écoulement de puissance de l'étude 2

EMTP Harmonic Steady-State solution

Steady-State for Design file: F:\kundur_3_PI.ecf

Solution date: 09/12/07
Solution time: 17:35:30.085

Solution frequency: 60 : Show Node Voltages

Node Voltages

Node		Real (V)	Imaginary (V)	Module (V)	Phase (degrees)
YgYg_np1/xfmr_A/s36	1	+0.1556187928E+05	+0.3695435274E+04	+0.1599463437E+05	+0.1335843730E+02
YgYg_np1/xfmr_A/s31	2	+0.1789616117E+06	+0.4249750565E+05	+0.1839382952E+06	+0.1335843730E+02
YgYg_np1/xfmr_B/s36	3	-0.4580598814E+04	-0.1532470042E+05	+0.1599463437E+05	-0.1066415627E+03
YgYg_np1/xfmr_B/s31	4	-0.5267688636E+05	-0.1762340549E+06	+0.1839382952E+06	-0.1066415627E+03
YgYg_np1/xfmr_C/s36	5	-0.1098128046E+05	+0.1162926515E+05	+0.1599463437E+05	+0.1333584373E+03
YgYg_np1/xfmr_C/s31	6	-0.1262847253E+06	+0.1337365492E+06	+0.1839382952E+06	+0.1333584373E+03
YgYg_np2/xfmr_A/s36	7	+0.1562783923E+05	+0.7396027288E+03	+0.1564533065E+05	+0.2709556929E+01
YgYg_np2/xfmr_A/s31	8	+0.1797201511E+06	+0.8505431381E+04	+0.1799213024E+06	+0.2709556929E+01
YgYg_np2/xfmr_B/s36	9	-0.7173404861E+04	-0.1390390714E+05	+0.1564533065E+05	-0.1172904431E+03
YgYg_np2/xfmr_B/s31	10	-0.8249415590E+05	-0.1598949321E+06	+0.1799213024E+06	-0.1172904431E+03
YgYg_np2/xfmr_C/s36	11	-0.8454434365E+04	+0.1316430441E+05	+0.1564533065E+05	+0.1227095569E+03
YgYg_np2/xfmr_C/s31	12	-0.9722599520E+05	+0.1513895007E+06	+0.1799213024E+06	+0.1227095569E+03
YgYg_np3/xfmr_A/s36	13	+0.1444107114E+05	-0.8152521688E+04	+0.1658336955E+05	-0.2944634370E+02
YgYg_np3/xfmr_A/s31	14	+0.1660723182E+06	-0.9375399942E+05	+0.1907087498E+06	-0.2944634370E+02
YgYg_np3/xfmr_B/s36	15	-0.1428082646E+05	-0.8430073624E+04	+0.1658336955E+05	-0.1494463437E+03
YgYg_np3/xfmr_B/s31	16	-0.1642295043E+06	-0.9694584668E+05	+0.1907087498E+06	-0.1494463437E+03
YgYg_np3/xfmr_C/s36	17	-0.1602446848E+03	+0.1658259531E+05	+0.1658336955E+05	+0.9055365630E+02
YgYg_np3/xfmr_C/s31	18	-0.1842813875E+04	+0.1906998461E+06	+0.1907087498E+06	+0.9055365630E+02
YgYg_np4/xfmr_A/s36	19	+0.1600024216E+05	-0.5750357713E+04	+0.1700218700E+05	-0.1976802712E+02
YgYg_np4/xfmr_A/s31	20	+0.1840027848E+06	-0.6612911370E+05	+0.1955251505E+06	-0.1976802712E+02
YgYg_np4/xfmr_B/s36	21	-0.1298007694E+05	-0.1098143732E+05	+0.1700218700E+05	-0.1397680271E+03
YgYg_np4/xfmr_B/s31	22	-0.1492708848E+06	-0.1262865292E+06	+0.1955251505E+06	-0.1397680271E+03
YgYg_np4/xfmr_C/s36	23	-0.3020165218E+04	+0.1673179503E+05	+0.1700218700E+05	+0.1002319729E+03
YgYg_np4/xfmr_C/s31	24	-0.3473190000E+05	+0.1924156429E+06	+0.1955251505E+06	+0.1002319729E+03

7a	25	+0.1764859176E+06	-0.1874872395E+05	+0.1774789953E+06	-0.6063989936E+01
7b	26	-0.1044798300E+06	-0.1434669261E+06	+0.1774789953E+06	-0.1260639899E+03
7c	27	-0.7200608757E+05	+0.1622156500E+06	+0.1774789953E+06	+0.1139360101E+03
9a	28	+0.1495428897E+06	-0.1141627767E+06	+0.1881388196E+06	-0.3735853407E+02
9b	29	-0.1736393096E+06	-0.7242655307E+05	+0.1881388196E+06	-0.1573585341E+03
9c	30	+0.2409641995E+05	+0.1865893298E+06	+0.1881388196E+06	+0.8264146593E+02
1a	31	+0.1532552689E+05	+0.5638695978E+04	+0.1632993162E+05	+0.2020000000E+02
5a	32	+0.1789616117E+06	+0.4249750565E+05	+0.1839382952E+06	+0.1335843730E+02
1b	33	-0.2779509482E+04	-0.1609164360E+05	+0.1632993162E+05	-0.9980000000E+02
5b	34	-0.5267688636E+05	-0.1762340549E+06	+0.1839382952E+06	-0.1066415627E+03
1c	35	-0.1254601740E+05	+0.1045294762E+05	+0.1632993162E+05	+0.1402000000E+03
5c	36	-0.1262847253E+06	+0.1337365492E+06	+0.1839382952E+06	+0.1333584373E+03
2a	37	+0.1593776570E+05	+0.2745019672E+04	+0.1617243050E+05	+0.9772383649E+01
6a	38	+0.1797201511E+06	+0.8505431381E+04	+0.1799213024E+06	+0.2709556929E+01
2b	39	-0.5591626079E+04	-0.1517501981E+05	+0.1617243050E+05	-0.1102276164E+03
6b	40	-0.8249415590E+05	-0.1598949321E+06	+0.1799213024E+06	-0.1172904431E+03
2c	41	-0.1034613962E+05	+0.1243000014E+05	+0.1617243050E+05	+0.1297723836E+03
6c	42	-0.9722599520E+05	+0.1513895007E+06	+0.1799213024E+06	+0.1227095569E+03
4a	43	+0.1564449585E+05	-0.6677550415E+04	+0.1700999500E+05	-0.2311427520E+02
10a	44	+0.1660723182E+06	-0.9375399942E+05	+0.1907087498E+06	-0.2944634370E+02
4b	45	-0.1360517622E+05	-0.1020975563E+05	+0.1700999500E+05	-0.1431142752E+03
10b	46	-0.1642295043E+06	-0.9694584668E+05	+0.1907087498E+06	-0.1494463437E+03
4c	47	-0.2039319630E+04	+0.1688730604E+05	+0.1700999500E+05	+0.9688572480E+02
10c	48	-0.1842813875E+04	+0.1906998461E+06	+0.1907087498E+06	+0.9055365630E+02
3a	49	+0.1686992482E+05	-0.4065722639E+04	+0.1735293820E+05	-0.1355012338E+02
11a	50	+0.1840027848E+06	-0.6612911370E+05	+0.1955251505E+06	-0.1976802712E+02
3b	51	-0.1195598150E+05	-0.1257692214E+05	+0.1735293820E+05	-0.1335501234E+03
11b	52	-0.1492708848E+06	-0.1262865292E+06	+0.1955251505E+06	-0.1397680271E+03
3c	53	-0.4913943321E+04	+0.1664264477E+05	+0.1735293820E+05	+0.1064498766E+03
11c	54	-0.3473190000E+05	+0.1924156429E+06	+0.1955251505E+06	+0.1002319729E+03
8a	55	+0.1656615454E+06	-0.6786151970E+05	+0.1790221591E+06	-0.2227594635E+02
8b	56	-0.1416005727E+06	-0.1095363469E+06	+0.1790221591E+06	-0.1422759464E+03
8c	57	-0.2406097268E+05	+0.1773978666E+06	+0.1790221591E+06	+0.9772405365E+02

Show Generated Power

Generated Power

		Current		Power		
From k	To m	Module (A)	Phase (degrees)	P (W)	Q (VAR)	Model
1a	-	+0.2936371977E+05	-0.1730653642E+03	+0.2333566008E+09	+0.5501423534E+08	SM:G1
1b	-	+0.2936371977E+05	+0.6693463578E+02	+0.2333566008E+09	+0.5501423534E+08	SM:G1
1c	-	+0.2936371977E+05	-0.5306536422E+02	+0.2333566008E+09	+0.5501423534E+08	SM:G1
2a	-	+0.3043836552E+05	+0.1712147460E+03	+0.2333333333E+09	+0.7833333333E+08	SM:G2
2b	-	+0.3043836552E+05	+0.5121474598E+02	+0.2333333333E+09	+0.7833333333E+08	SM:G2
2c	-	+0.3043836552E+05	-0.6878525402E+02	+0.2333333333E+09	+0.7833333333E+08	SM:G2
3a	-	+0.2843812997E+05	+0.1526952288E+03	+0.2396666667E+09	+0.5866666667E+08	SM:G3
3b	-	+0.2843812997E+05	+0.3269522875E+02	+0.2396666667E+09	+0.5866666667E+08	SM:G3
3c	-	+0.2843812997E+05	-0.8730477125E+02	+0.2396666667E+09	+0.5866666667E+08	SM:G3
4a	-	+0.2855430857E+05	+0.1407890960E+03	+0.2333333333E+09	+0.6733333333E+08	SM:G4
4b	-	+0.2855430857E+05	+0.2078909604E+02	+0.2333333333E+09	+0.6733333333E+08	SM:G4
4c	-	+0.2855430857E+05	-0.9921090396E+02	+0.2333333333E+09	+0.6733333333E+08	SM:G4
			Total	+0.2819069802E+10	+0.7780427060E+09	

Power balance

P= 0.1907348633E-05 Q=-0.1192092896E-05

ANNEXE XIV

Données du SMDA de l'étude de cas 3

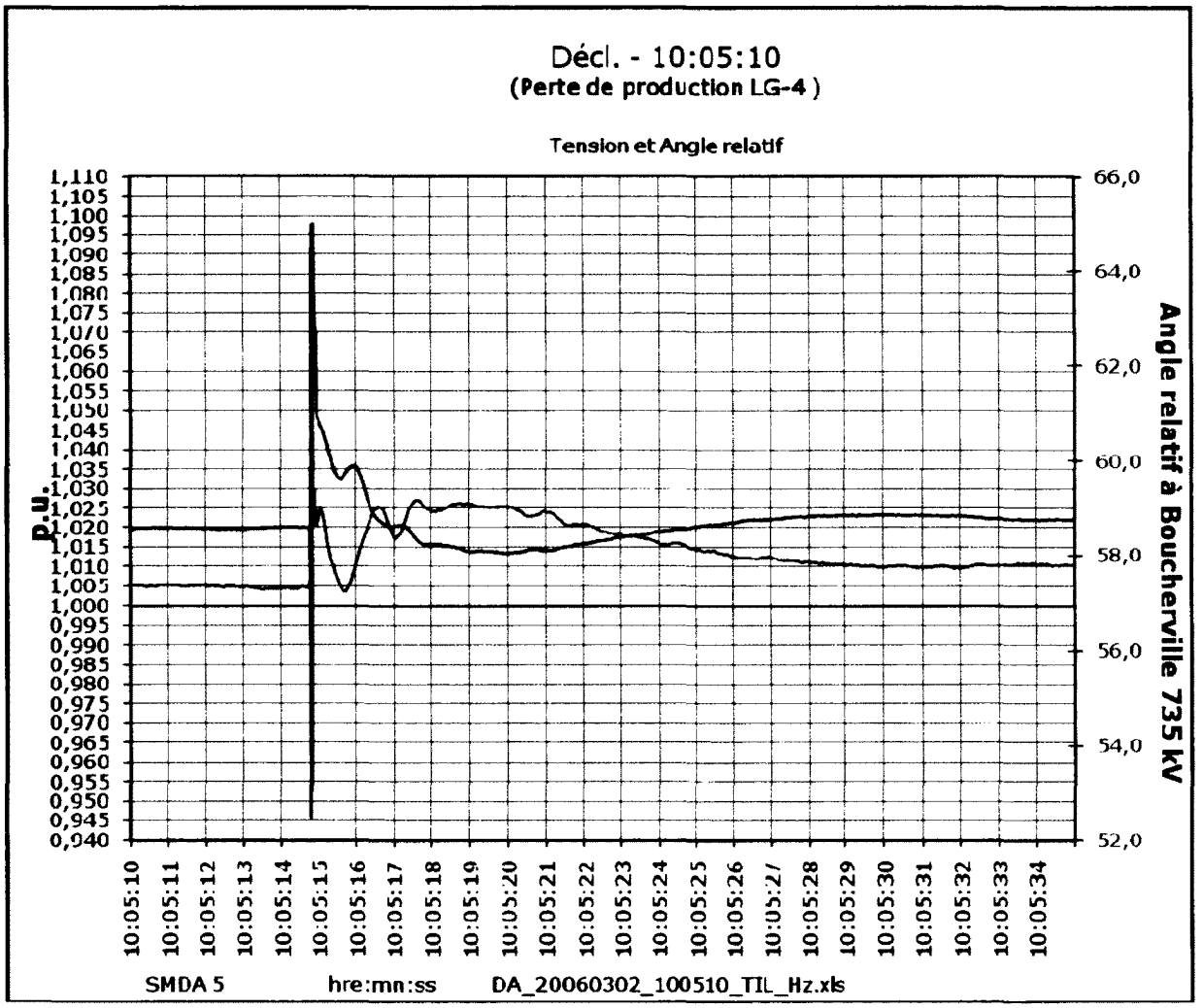


Figure-A XIV-1 Données du SMDA-5 au 2 mars 2006.

BIBLIOGRAPHIE

Bensoussan, David. 2008. *Commande moderne: Approche par modèles continus et discrets.* Montréal : Presses Internationales Polytechnique, 381 p.

Brown, T.J., V. Vittal, G.T. Heydt, A.R. Messina. 2008. « *A comparative assessment of two techniques for modal identification from power system measurement* ». *IEEE Transactions on power systems*, vol. 23, n° 3, p. 1408-1415.

Cheng, Sheung Hun, Nicolas J. Higham, Charles S. Kenney et Alan J. Laub. 2001. « *Approximating the Logarithm of a Matrix to Specified Accuracy* ». *SIAM J. Matrix Anal. Appl.*, 22(4), p. 1112-1125.

Ho, B. L. et R. E. Kalman. 1965. « *Effective Construction of Linear State-Variable Models from Input/Output Data* ». *Proc. of the 3rd Annual Allerton Conference on Circuit and System Theory*, p. 449-459.

Juang, J.-N. et R.S. Pappa. 1985. « *An Eigensystem Realization Algorithm (ERA) for Modal Parameter Identification and Model Reduction* ». *J. of Guidance, Control and Dynamics*, 8(5), p. 620-627.

Juang, Jer-Nan. 1994. *Applied System Identification.* New Jersey: PTR Prentice-Hall, 394 p.

Kamwa, Innocent. 1993. *Notes Pour un Courss d'Analyse Modale des Réseaux de Transport.* IREQ-93-292. Varennes (Qc) : Direction principale Recherche et Développement.

Kamwa, Innocent, D. Ndereyimana, D. Asber et R. Grondin. 1997. *Analyse Modale : Validation des Concepts et Application aux Systèmes de Commande.* IREQ-97-227. Varennes (Qc) : Direction principale Recherche et Développement, 231 p.

Kamwa, Innocent et Luc Gérin-Lajoie. 2000. « *State-Space System Identification – Towards MIMO Models for Modal Analysis and Optimization of Bulk Power Systems* ». *IEEE Transactions on power systems*, vol. 15, n° 2, p. 326-335.

Kamwa, I., R. Grondin, J. Béland, C. Lafond, G. Trudel et D. McNabb. 2006. « *Wide-Area Monitoring and Control at Hydro-Québec: Past, Present and Future* ». *IEEE*, 1-4244-0493-2/06, p. 1-12.

Kundur, P. 1994. *Power system stability and control*, United States of America: McGraw-Hill, 1176 p.

Lardies, J. 1998. « *State-space identification of vibrating systems from multi-ouput measurements* ». *Mechanical Systems and Signal Processing*, 12(4), p. 543-558.

Press, William H., Saul A. Teukolsky, William T. Vetterling et Brian P. Flannery. 1992. *Numerical Recipes in C : The art of Scientific Computing*, Second Edition. Cambridge: Press Syndicate of the University of Cambridge, 994 p.

Rao, Bhaskar D. et K. S. Arun. 1992. « *Model Based Processing of Signals : A State Space Approach* ». *Proceedings of the IEEE*, vol. 80, n° 2, p. 283-309.

Whaley, R. Clint, Antoine Petitet et Jack J. Dongarra. 2000. « *Automated Empirical Optimization of Software and the ATLAS project* ». En ligne. 33 p. <http://www.netlib.org/lapack/lawnspdf/lawn147.pdf>. Consulté le 12 janvier 2010.

Zhang S., X. Xie, J. Wu. 2008. « *WAMS-based detection and early-warning of low-frequency oscillations in large-scale power systems* ». *Electric Power Systems Research*, vol. 78, p. 897-906.