Algorithmes du type gradient à complexité de calcul réduite et à capacité de poursuite et vitesse de convergence accrues : application à l'annulation d'écho acoustique

Jacob Benesty

CENTRE NATIONAL D'ETUDES DES TELECOMMUNICATIONS

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

CENTRE DE RECHERCHES EN PHYSIQUE DE L'ENVIRONNEMENT TERRESTRE ET PLANETAIRE

CRPE

NOTE TECHNIQUE

CRPE/193

ALGORITHMES DU TYPE GRADIENT A COMPLEXITE DE CALCUL REDUITE ET A CAPACITE DE POURSUITE ET VITESSE DE CONVERGENCE ACCRUES. APPLICATION A L'ANNULATION D'ECHO ACOUSTIQUE.

Par

J. BENESTY

RPE/ETP
38-40, rue du Général Leclerc
92131 ISSY-LES-MOULINEAUX, FRANCE

CENTRE NATIONAL D'ETUDES
DES TELECOMMUNICATIONS
Centre Paris B

CENTRE NATIONAL DE LA
RECHERCHE SCIENTIFIQUE
Département TOAE

INGÉTEL

CENTRE DE RECHERCHES EN PHYSIQUE DE L'ENVIRONNEMENT TERRESTRE ET PLANETAIRE

NOTE TECHNIQUE CRPE/193
Juin 1991

ALGORITHMES DU TYPE GRADIENT A COMPLEXITE DE CALCUL REDUITE ET A CAPACITE DE POURSUITE ET VITESSE DE CONVERGENCE ACCRUES. APPLICATION A L'ANNULATION D'ECHO ACOUSTIQUE.

par

Jacob BENESTY
RPE/ETP
CRPE, 38-40 rue du Général Leclerc, 92131 ISSY LES MOULINEAUX

Le Directeur

G. SOMMERIA

Le Chef de Département

J.C. BIC

LISTE DE DIFFUSION SYSTEMATIQUE

C N E T

MM.	FENEYROL	Directeur du CNET
	THABARD	Directeur Adjoint du CNET
	FUERXER	Adjoint Militaire au Directeur du CNET
	MEREUR	Directeur des Programmes
	BLOCH	DICET
	THUE	DICET
MME	HENAFF	DICET
MM.	PIGNAL	PAB
	RAMAT	PAB
	ZYLBERSZTEJN	PAB-BAG
	MALOBERTI	PAB-SHM
	HOCQUET	PAB-STC
	THEBAULT	PAB-STS
	SOMMERIA	PAB-RPE
	BERTHELIER	PAB-RPE
	GENDRIN	PAB-RPE
MME	PARIS	PAB-RPE
MM.	BAUDIN	PAB-RPE
	BIC	PAB-RPE
	CERISIER	PAB-RPE
	LAVERGNAT	PAB-RPE
	ROBERT	PAB-RPE
	ROUX	PAB-RPE
	TESTUD	PAB-RPE
	VIDAL-MADJAR	PAB-RPE

C N R S

M.	BERROIR	TOAE
MME	SAHAL	TOAE
MM.	CHARPENTIER	SPI
	COUTURIER	INSU
	CADET	INSU
	MEGIE	MEN

C N E S

MMES	AMMAR
	DEBOUZY
MM.	BAUDOIN
	FELLOUS
	HERNANDEZ (Toulouse)

E E R M

M.	ANDRE

Bibliothèques

CNET-SDI (2)
CNET-EDB
CNET-RPE (Issy) (5)
CNET-RPE (St Maur) (2)
Observatoire de Meudon
CNRS-SA
CNRS-INIST
CNRS-LPCE

LISTE COMPLEMENTAIRE

PAA/TSA	MOUNIER
PAA/SRE	GIRARD
PAA/ATR	BOUSSER
LAB/SMR	LECLERT
LAB/SMR	VANDAMME
LAB/SMR	LOUSSOUARN
LAA/TSS	JULLIEN
LAA/TSS	MONFORT
LAA/TSS	GILLOIRE
LAA/TSS	COMBESCURE
LAA/TSS	PETIT
LAA/TSS	LE TOURNEUR
LAA/RSM	PAYS
LAB/OCM	BARDOUIL

CNS - GRENOBLE

CNS/CCI	LARDY
CNS/CCI	ROUQUIER
CNS/CCI	SENN

CCETT - RENNES

SRL/DCS	HELARD
SRL/DCS	EVAIN
SRL/DCS	DJOKO
SRL/DCS	LANOISELEE
SRL/DCS	PALICOT
SRL/DCS	THOMAS
SRL/DCS	VEILLARD
SRL/DRL	LE DAIN

DOCUMENTATION

EXTERIEUR

TRT	MARTIN

Cette note technique reprend le texte d'une thèse effectuée à PAB/RPE, dans le domaine du filtrage adaptatif appliqué à l'annulation d'échos acoustiques.

Plus précisément, cette thèse concerne l'amélioration de l'algorithme LMS sous ses aspects charge de calcul d'une part, mais aussi comportement adaptatif (vitesse de convergence et poursuite). Ces résultats sont dérivés à partir d'une nouvelle présentation des algorithmes en blocs: Les algorithmes obtenus sont comparés à des algorithmes classiques tels que le LMS et le FRLS dans une situation réelle: l'identification d'une réponse impulsionnelle acoustique de salle, d'une part en présence d'une non-stationnarité dans le système, et d'autre part avec des signaux non-stationnaires en entrée (parole).

Enfin, nous montrons que les techniques développées dans le cas du LMS s'appliquent à d'autres types d'algorithmes tel que le CMA, qui semble être un bon "candidat" pour l'égalisation aveugle.

TABLE DES MATIERES

CHAPITRE I

INTRODUCTION

Le filtrage adaptatif occupe une place très importante en traitement numérique du signal. Pour s'en convaincre, il suffit de voir ses applications [1,3,4,5,6]. En effet, nous le retrouvons dans les différents domaines suivants: estimation et identification, séismologie, traitement de la parole, radar, sonar, instrumentation biomédicale, etc.

Le principe d'un filtre adaptatif est donné à la figure 1.1. Nous avons à notre disposition un signal, que l'on appelle la référence, dont on désire se rapprocher le mieux possible par un autre signal représentant la sortie d'un filtre à coefficients variables. En retranchant la sortie du filtre à la référence, nous obtenons un signal d'erreur que l'on cherche ensuite à minimiser suivant un critère donné. Cette erreur servira, aussi, à la mise à jour des coefficients du filtre à l'aide d'un algorithme adaptatif [7].

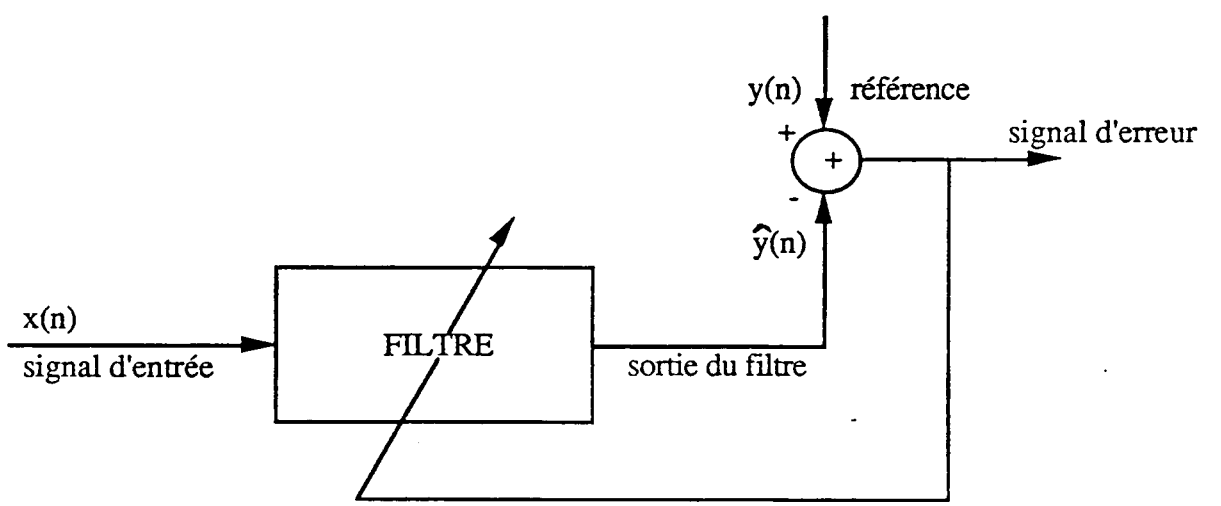


Figure 1.1: Principe d'un filtre adaptatif

De cette présentation, trois points importants se dégagent:

- le critère d'optimisation,
- l'algorithme de mise à jour des coefficients et
- la structure du filtre.

Le critère d'optimisation est en général du type des moindres carrés. Pour le filtre, nous avons pour notre part choisi la structure transverse RIF (à réponse impulsionnelle finie), car elle est de loin la plus simple et la plus utilisée.

Un exemple très intéressant de traitement adaptatif est l'annulation d'écho acoustique dans les salles de téléconference [1,9], car c'est un problème difficile et actuel où l'on cherche encore une solution efficace.

En effet, dans une communication bidirectionnelle, le haut-parleur et le microphone de chacune des salles créent un couplage acoustique (son du haut-parleur capté par le microphone) et la jonction des deux salles constitue une boucle fermée. Une première conséquence est que l'écho acoustique que l'on peut entendre dans les salles, détériore la qualité de l'écoute. Une autre conséquence est que la boucle peut être instable, un sifflement (effet Larsen) se produit alors et interdit la communication. Il est donc nécessaire de contrôler l'effet du couplage acoustique.

Le problème de l'annulation acoustique peut alors être ramené à celui de l'identification d'un canal avec entrée connue x et sortie y connue, suivie d'un traitement correctif approprié (voir figure 1.2). L'objectif pour l'utilisateur est de minimiser l'écho résiduel. En absence de parole, le signal émis est l'écho résiduel.

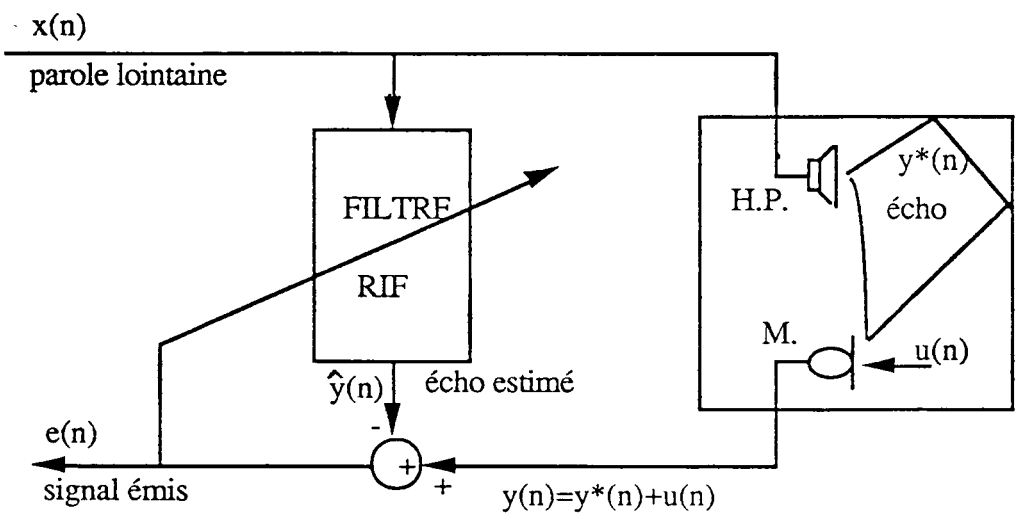


Figure 1.2: Schéma de l'annulation de l'écho acoustique

Le canal acoustique est caractérisé par:

- une réponse impulsionnelle à durée infinie dont la partie utile à identifier est de l'ordre de 500 ms, soit 8.000 points pour une fréquence d'échantillonnage de 16 kHz. Cette réponse a une structure temporelle complexe et n'admet pas de modèle simple (à nombre réduit de paramètres);
- une non-stationnarité due aux mouvements des personnes, aux déplacements d'objets, etc.

On doit donc utiliser des algorithmes adaptatifs puisque le canal acoustique est inconnu et évolue au cours du temps. De plus, le signal d'entrée est la parole, qui est fortement non-stationnaire, ce qui rend encore plus difficile l'adaptation.

La sortie du filtre RIF variable est une estimation de l'écho acoustique. La mise à jour des coefficients peut se faire de deux manières. La première, consiste à faire une adaptation à chaque échantillon de temps, c'est-à-dire que pour chaque nouvelle entrée, le filtre s'adaptera: ce sont les algorithmes "échantillon par échantillon". Le principe de la seconde méthode est d'attendre L entrées successives (L étant souvent la longueur du filtre dont le choix est lié à la longueur du canal acoustique) avant d'adapter les coefficients de ce filtre: ce sont les algorithmes par "bloc".

Dans la première catégorie d'algorithmes (échantillon par échantillon), on trouve essentiellement les algorithmes suivants: LMS, TDLMS et RLS.

Le premier ("Least Mean Square"-LMS) est obtenu par minimisation de l'erreur quadratique moyenne et l'adaptation se réalise avec le gradient stochastique qui est très simple à mettre en œuvre. Son inconvénient majeur est sa faible vitesse de convergence qui est très dépendante du spectre de la matrice de corrélation du signal d'entrée.

Dans l'algorithme TDLMS ("Transform-Domain LMS"), le signal d'entrée est multiplié par une transformée orthogonale connue et on se sert de la méthode du gradient pour l'adaptation des coefficients. Cette technique réduit, en général, le rapport de la plus grande valeur propre à la plus petite valeur propre de la matrice de corrélation, ce qui a pour avantage d'améliorer la vitesse de convergence de l'algorithme par rapport au LMS. Le TDLMS n'est efficace que pour un signal d'entrée stationnaire.

Enfin, le troisième algorithme ("Recursive Least Squares"-RLS) fait directement intervenir l'inversion de la matrice d'autocorrélation, ce qui le rend optimal au sens de la vitesse de convergence mais sa charge de calcul est élevée même dans sa version rapide.

Dans la seconde catégorie d'algorithmes (par bloc), on reprend les algorithmes séquentiels,

mais au lieu d'adapter le filtre à chaque instant, on l'adapte une fois par bloc (N), connaissant N entrées successives. Ainsi, le LMS devient le "Block-LMS", le RLS devient le "Block-RLS", etc. Néanmoins, le plus connu est le BLMS car on peut utiliser la FFT ("Fast Fourier Transform") comme intermédiaire de calcul pour diminuer sa complexité arithmétique, à condition que la taille du bloc soit égale à la longueur du filtre.

Malheureusement, aucune de ces deux méthodes (adapter le filtre échantillon par échantillon ou par bloc) ne convient vraiment lorsque l'on doit traiter de longues réponses impulsionnelles; ce qui est le cas des réponses impulsionnelles des salles de téléconférence.

En effet, la première technique nécessiterait un très grand nombre d'opérations, même pour les algorithmes les plus simples comme le LMS, qui a en plus l'inconvénient de converger lentement pour des entrées corrélées. Cette charge de calcul est un problème très sérieux quand on sait qu'un traitement adaptatif doit se faire en temps réel.

La seconde technique, quant à elle, réduit considérablement la complexité arithmétique grâce à l'apparition, dans le traitement par bloc, d'une matrice ayant une structure particulière. Mais la taille de ce bloc doit être égale à la longueur du filtre dans la plupart de ces algorithmes, ce qui pose essentiellement deux problèmes: des retards importants à la sortie du filtre (alors qu'en général le système tolère peu de retard) et l'utilisation d'un grand nombre de mémoires, qui font que ces algorithmes sont inutilisables en pratique. De plus, le fait que le filtre soit fixe pendant tout un bloc dégrade le comportement adaptatif de l'algorithme (comparé à un algorithme séquentiel) si le système ou le signal d'entrée sont non-stationnaires.

L'objectif de cette thèse est d'essayer d'obtenir une réduction de charge de calcul de certains algorithmes, certes, mais aussi une amélioration du comportement adaptatif (vitesse de convergence, capacité de poursuite). Pour parvenir à cette fin, nous cherchons d'abord à comprendre le mécanisme d'une autre formulation par bloc et pour ce faire, nous construirons des algorithmes intermédiaires entre les deux classes vues précédemment, c'est-à-dire d'algorithmes par bloc où la taille de bloc peut être choisie aussi petite que l'on veut. En d'autres termes, on cherchera à "déconnecter" la dimension du bloc de la longueur du filtre. Cette formulation par bloc est nouvelle, car elle est déduite directement d'un algorithme séquentiel, tout en conservant l'équivalence avec la version initiale. Elle permet, en outre, une réduction du nombre d'opérations, car elle fait apparaitre un filtrage fixe, mais ouvre aussi de nouvelles

possibilités d'algorithmes adaptatifs.

Le point de départ de notre travail a été la question suivante: comment réduire la complexité arithmétique de l'algorithme LMS, sans en changer le comportement, tout en travaillant par bloc, et ceci quelle que soit la taille de bloc?

Le chapitre II présente de façon unifiée les principaux algorithmes de filtrage adaptatif. Nous insistons sur le fait que la plupart des algorithmes découlent du RLS par approximations et simplifications successives, et que leur vitesse de convergence dépendra de ces approximations. Nous avons contribué à cette unification, puisque nous montrons le lien qu'il y a entre l'algorithme de Newton (que nous écrivons différemment pour aboutir au résultat désiré) et le TDLMS d'une part, et les algorithmes par bloc et le LMS (dont une formulation exacte par bloc est fournie) d'autre part.

Au chapitre III, nous donnons une approche nouvelle des algorithmes par bloc en prenant l'exemple du LMS. En effet, l'algorithme proposé ("Fast Exact LMS"-FELMS) est non seulement mathématiquement équivalent à l'algorithme initial (ce qui n'est, bien sûr, pas le cas du BLMS) mais aussi diminue la charge de calcul même pour une taille de bloc aussi petite que N=2. Cet exemple est généralisé dans l'article I de l'annexe, pour une taille de bloc quelconque. La possibilité de choisir la taille de ce bloc permet de contrôler le retard à la sortie du filtre, tout en obtenant la réduction du nombre d'opérations correspondant.

D'autre part, nous montrons (en annexe) que le BLMS est une approximation du FELMS, ce qui explique son comportement suivant la nature du signal d'entrée.

L'article I détaille différentes approximations du FELMS, fournit une analyse de son comportement vis-à-vis du bruit de calcul, et montre que les mêmes techniques s'appliquent à des variantes du LMS (signe, normalisé).

Nous prouvons, ensuite, que des DFT de petites longueurs peuvent être utilisées comme un intermédiaire de calcul dans le FELMS. L'algorithme obtenu est le FD-FELMS ("Frequency Domain-FELMS") et il est strictement équivalent au LMS. Cette version fréquentielle est très utile lorsqu'on doit travailler avec des tailles de blocs moyennes tandis que la version temporelle est intéressante pour des tailles de blocs plus petites.

Après avoir obtenu deux versions rapides (au sens de la complexité arithmétique) du LMS,

nous proposons au chapitre IV de développer certaines idées qui découlent du FELMS en vue d'améliorer ses performances (vitesse de convergence et poursuite).

Le premier algorithme que nous dérivons est une approximation du FD-FELMS. En effet, nous profitons des propriétés de la DFT pour accélérer la vitesse de convergence, en remplaçant le pas d'adaptation (scalaire unique) par une matrice diagonale. Cet algorithme converge beaucoup plus vite que le LMS mais il poursuit moins bien. Il a été obtenu en même temps que d'autres auteurs et indépendamment.

Nous cherchons ensuite à augmenter la capacité de poursuite de l'algorithme précédent, pour cela, nous proposons un nouveau critère par bloc, dépendant de l'erreur du FELMS, à partir duquel un algorithme est déduit ("New Block-FELMS"-NB-FELMS). Ce dernier converge plus vite, poursuit mieux les non-stationnarités du système et ceci avec une charge de calcul qui reste toujours inférieure à celle du LMS. Le NB-FELMS nécessite un peu plus d'opérations que le FD-FELMS. C'est le prix à payer pour avoir un algorithme qui converge plus vite et qui poursuit mieux grâce à un critère plus "fort" que celui utilisé dans le cas du LMS.

Il serait intéressant de trouver un algorithme séquentiel du type RLS, mais avec une matrice de corrélation plus petite que la taille du filtre. Un premier pas est fait dans ce sens, puisque nous proposons un algorithme plus général que le FELMS, faisant le lien entre le NLMS et le "Block-RLS". Pour l'instant, l'intérêt de celui-ci est plus théorique que pratique car il y a un système linéaire à résoudre une fois par bloc, ce qui augmente le nombre d'opérations. Néanmoins, on peut toujours arriver à une complexité arithmétique comparable à celle du LMS.

Au chapitre V nous faisons une synthèse des principaux algorithmes que nous avons obtenus. Ceci nous permettra de mettre en évidence les points communs, les différences ainsi que les avantages et inconvénients de chacun d'entre eux.

Le chapitre VI est entièrement consacré à des simulations, mais dans le cas où le signal d'entrée est la parole qui est fortement non-stationnaire. Le but est de voir comment se comportent nos algorithmes par rapport au NLMS et au RLS dans une situation réaliste d'annulation d'écho acoustique.

Enfin, le chapitre VII a pour but de montrer que ce qui a été fait dans le cas du LMS peut s'appliquer à d'autres types d'algorithmes. Il concerne essentiellement l'Algorithme à Module Constant (CMA). Nous nous sommes intéressés à ce dernier parce que l'application des

techniques classiques de diminution de la charge de calcul ont fourni un algorithme du type "Block-CMA" (avec utilisation de la FFT dont la taille est égale à deux fois la longueur du filtre) à vitesse de convergence très lente, à cause de la non-linéarité de l'erreur.

Nous montrons sur un exemple simple (N=2) que l'on peut diminuer le nombre d'opérations du CMA sans aucune approximation. L'article II de l'annexe fournit plusieurs possibilités, puisque nous proposons quatre versions rapides (aussi bien dans le domaine temporel que fréquentiel), qui sont toutes équivalentes au niveau du comportement mais pas au niveau de la complexité arithmétique.

CHAPITRE II

PRESENTATION UNIFIEE DU FILTRAGE RIF ADAPTATIF AUSSI BIEN DANS LE DOMAINE TEMPOREL QUE FREQUENTIEL

Les algorithmes de filtrage adaptatif associés à la structure transverse RIF sont très nombreux et ont largement été étudiés dans la littérature. Dans ce chapitre, nous en rappelons les principaux, et ceux qui nous semblent les plus intéressants. Les algorithmes qui ont certainement suscité le plus de travaux sont le LMS et le RLS. La raison de cet intérêt est que le premier est simple à mettre en œuvre tandis que le second a une vitesse de convergence optimale. La plupart des autres algorithmes découlent de ces deux derniers, comme nous allons essayer de le montrer.

Cette présentation est plus qu'un rappel puisque nous montrons que tous ces algorithmes adaptatifs sont liés entre eux, et qu'il est possible de passer de l'un à l'autre, en justifiant quand c'est possible, l'approximation faite. Ainsi, par exemple, l'algorithme TDLMS est un cas particulier de l'algorithme de Newton, que l'on propose d'écrire d'une autre façon, qui lui même est une approximation du RLS. Par conséquent, les algorithmes dans les domaines temporel et fréquentiel sont proches sinon comparables, et il n'y a plus à distinguer ces deux classes d'algorithmes quant à leur comportement adaptatif. D'autre part, nous montrons comment obtenir un algorithme par bloc à partir d'un algorithme séquentiel et ceci en conservant l'équivalence entre ces deux formes.

Nous commençons cette étude par l'algorithme RLS qui est construit à partir du critère des moindres carrés récursif, et progressivement nous aboutissons à des algorithmes moins complexes au niveau de la charge de calcul.

2.1. L'ALGORITHME DES MOINDRES CARRES RECURSIF (MCR OU RLS)

Il s'agit de trouver une équation de récurrence sur le temps, pour l'obtention des valeurs du filtre à l'instant n+1 à partir de ses valeurs à l'instant n, et en fonction des nouvelles entrées. Pour cela, il faut d'abord déduire la solution optimale au sens des moindres carrés.

Soit H_n le vecteur des L coefficients $h_i(n)$ du filtre, X_n le vecteur à l'instant n des L échantillons du signal d'entrée les plus récents et y(n) la sortie du système à l'instant n (voir figure 1.1):

$$H_n = [h_0(n) \quad h_1(n).....h_{L-1}(n)]^t$$

$$X_n = [x(n) \quad x(n-1).....x(n-L+1)]^t$$

La méthode d'optimisation consiste à minimiser, par rapport au vecteur H, une fonction coût $J_{LS}(n)$, qui est une somme pondérée des carrés des valeurs de l'erreur [7]:

(2.1)

$$J_{LS}(n) = \sum_{p=1}^{n} w^{n-p}[y(p) - X_p^t H_n]^2$$

où w est un facteur de pondération: 0<<w<1. Ce facteur est indispensable pour un signal d'entrée non-stationnaire, car il permet de favoriser les nouvelles données et d'atténuer les anciennes, d'où son nom de facteur d'oubli.

Le critère précédent étant quadratique, son minimum est fourni par le zéro de sa dérivée. On obtient alors l'équation normale:

(2.2)

$$H_n = R^{-1}(n) r_n$$

avec:

$$R(n) = \sum_{p=1}^{n} w^{n-p} X_p X_p^t \quad \text{et}$$

$$r_n = \sum_{p=1}^{n} w^{n-p} y(p) X_p$$

R est une estimation de la matrice d'autocorrélation du signal d'entrée et r une estimation de l'intercorrélation entre les signaux de référence et d'entrée.

La matrice R est symétrique, elle est de Toeplitz quand n tend vers l'infini (si w=1) [11].

L'équation normale (2.2) est donc un système linéaire qu'on peut résoudre à chaque instant pour trouver les valeurs du filtre, mais on lui préfére un algorithme récursif moins complexe.

Un algorithme récursif a pour objet de mettre à jour les coefficients H_n à partir de la

connaissance de H_{n-1} et des nouvelles entrées. L'algorithme peut être déduit, en écrivant les équations de récurrence suivantes:

(2.3)

$$R(n+1) = wR(n) + X_{n+1}X_{n+1}^t$$

et

(2.4)

$$r_{n+1} = wr_n + X_{n+1}y(n+1)$$

d'où, d'après (2.2):

$$H_{n+1} = R^{-1}(n+1)[wr_n + X_{n+1}y(n+1)]$$

comme:

$$r_n = R(n)H_n = [w^{-1}R(n+1) - w^{-1}X_{n+1}X_{n+1}^t]H_n$$

en remplaçant r_n dans l'expression (2.2), on trouve:

$$H_{n+1} = H_n + R^{-1}(n+1)X_{n+1}[y(n+1) - X_{n+1}^t H_n]$$

ou encore:

(2.5)

$$H_{n+1} = H_n + R^{-1}(n+1)X_{n+1}\varepsilon(n+1)$$

ε étant l'erreur d'estimation "*a priori* " (car elle se calcule avec le vecteur des coefficients de l'instant précédent) qui est donnée par:

(2.6)

$$\varepsilon(n+1) = y(n+1) - X_{n+1}^t H_n$$

On pose souvent:

(2.7)

$$G_{n+1} = R^{-1}(n+1)X_{n+1}$$

qui est appelé gain de Kalman "*a posteriori* ", alors que le gain de Kalman *"a priori"* est défini par:

$$G'_{n+1} = R^{-1}(n)X_{n+1}$$

Il faut bien noter que l'équation (2.5) d'adaptation du filtre découle directement de l'équation

normale (2.2) sans aucune approximation. Les coefficients H_n fournis par cet algorithme sont optimaux (solution exacte de la minimisation du critère (2.1)) à chaque instant. On aura donc une vitesse de convergence optimale.

Après avoir obtenu un algorithme adaptatif, il est important de pouvoir éviter l'inversion de la matrice R (ou la résolution du système linéaire (2.7)), qui nécessiterait un nombre d'opérations proportionnel à L^3, au profit d'une équation de récurrence. Ceci est possible grâce au lemme d'inversion de matrice (L1) suivant [2]:

Etant donné quatre matrices A, B, C et D de dimensions appropriées, on a la relation suivante:

$(A+BCD)^{-1}=A^{-1} - A^{-1}BC\,(C^{-1} + DA^{-1}B)^{-1}\,DA^{-1}$

pour autant que les inversions évoquées aient un sens.

Nous appliquons ce lemme à l'expression (2.3), en posant:

$A=wR(n)$, $B=X_{n+1}$, $C=I$ et $D=B^t$

pour obtenir:

(2.8)

$$R^{-1}(n+1) = w^{-1}R^{-1}(n) - w^{-2}\frac{R^{-1}(n)X_{n+1}X_{n+1}^t R^{-1}(n)}{1+w^{-1}X_{n+1}^t R^{-1}(n)X_{n+1}}$$

L'équation (2.8) calcule récursivement l'inverse de la matrice R(n+1) connaissant l'inverse de R(n). Cette procédure est moins complexe que d'inverser directement cette matrice de corrélation (ou de résoudre le système linéaire donné par l'équation (2.7)) à chaque instant.

Enfin, nous poserons:

$P(n+1)=R^{-1}(n+1)$

Résumons l'algorithme RLS qui fut, pour la première fois, proposé par Godard [10]:

(2.9)

a) $G_{n+1} = \dfrac{w^{-1}P(n)X_{n+1}}{1+w^{-1}X_{n+1}^{t}P(n)X_{n+1}}$

b) $\varepsilon(n+1) = y(n+1) - X_{n+1}^{t}H_n$

c) $H_{n+1} = H_n + G_{n+1}\varepsilon(n+1)$

d) $P(n+1) = w^{-1}P(n) - w^{-1}G_{n+1}X_{n+1}^{t}P(n)$

Initialisation de l'algorithme:

$P(0) = \delta^{-1}I$, δ = constante positive petite
$h_i(0) = 0, i = 0,1,.....,L-1$

L'utilisation du lemme (L1) a réduit la complexité arithmétique de l'algorithme RLS de L^3 à L^2. Cependant, il est encore possible de diminuer cette charge de façon très significative en transformant l'algorithme pour faire apparaître les paramètres des prédictions avant et arrière, comme nous allons le voir dans le prochain paragraphe.

2.2. L'ALGORITHME DES MOINDRES CARRES RECURSIF RAPIDE (MCRR OU FRLS)

L'algorithme RLS est construit à partir d'une équation de récurrence sur le temps pour l'inversion de la matrice d'autocorrélation. L'algorithme FRLS utilise une équation de récurrence à la fois sur le temps et sur l'ordre du gain de Kalman, qui est un vecteur de longueur L et qui est donc plus facile à manipuler qu'une matrice (LxL). On espère, ainsi, réduire la complexité arithmétique du RLS.

Nous allons d'abord donner quelques définitions, qui apparaitront dans l'algorithme recherché [5,7].

Soit la quantité scalaire:

$$\varphi(n+1) = \frac{w}{w + X_{n+1}^{t}R^{-1}(n)X_{n+1}} = 1 - X_{n+1}^{t}R^{-1}(n+1)X_{n+1} = 1 - \theta(n+1)$$

θ est la variable de vraisemblance. On vérifie que $\theta(n)$ est bornée par: $0 \leq \theta(n) \leq 1$,

ce qui implique que: $0 \leq \varphi(n) \leq 1$.

Avec la définition suivante:

$$\rho(n+1) = w / \varphi(n+1)$$

on a la relation entre le gain a posteriori et a priori:

(2.10)

$$G_{n+1} = G'_{n+1} / \rho(n+1)$$

Maintenant, on peut mettre en évidence une autre relation de récurrence sur les coefficients, qui nous servira par la suite:

(2.11)

$$H_{n+1} = H_n + G'_{n+1} \varepsilon(n+1) / \rho(n+1)$$
$$= H_n + w^{-1} G'_{n+1} e(n+1)$$

avec:

(2.12)

$$e(n+1) = y(n+1) - X^t_{n+1} H_{n+1}$$

e étant l'erreur d'estimation "*a posteriori* ".

L'algorithme rapide FRLS se sert des équations de la prédiction avant et arrière, qui apparaissent naturellement dans la matrice R et dans son inverse, comme nous le verrons plus loin. Commençons par en rappeler les principes.

La prédiction linéaire avant est basée sur la fonction coût:

(2.13)

$$J_a(n) = \sum_{p=1}^{n} w^{n-p} [x(p) - A^t_n X_{p-1}]^2$$

où A_n est le vecteur des coefficients du prédicteur dont l'optimum est donné par:

(2.14)

$$A_n = R^{-1}(n-1) r^a_n$$

avec:

$$r^a_n = \sum_{p=1}^{n} w^{n-p} x(p) X_{p-1}$$

et la mise à jour des coefficients se fait comme suit:

(2.15)

$$A_{n+1} = A_n + G_n \varepsilon_a(n+1)$$
$$= A_n + w^{-1} G'_n e_a(n+1)$$

avec:
$e_a(n+1) = \varepsilon_a(n+1) / \varphi(n)$
et:

$$\varepsilon_a(n+1) = x(n+1) - A_n^t X_n$$

ε_a et e_a sont respectivement les erreurs de prédiction avant a priori et a posteriori.

Enfin, l'énergie d'erreur de prédiction avant est:

(2.16)

$$E_a(n+1) = \sum_{p=1}^{n+1} w^{n+1-p} x^2(p) - A_{n+1}^t r_{n+1}^a$$
$$= w[E_a(n) + \varepsilon_a(n+1)\varepsilon_a(n+1) / \rho(n)]$$

On procède de la même façon pour la prédiction arrière. La fonction coût est:

(2.17)

$$J_b(n) = \sum_{p=1}^{n} w^{n-p}[x(p-L) - B_n^t X_p]^2$$

Le vecteur optimal des coefficients du prédicteur est donné par:

(2.18)

$$B_n = R^{-1}(n) r_n^b$$

avec:

$$r_n^b = \sum_{p=1}^{n} w^{n-p} x(p-L) X_p$$

La mise à jour du prédicteur arrière est faite avec l'expression (2.19):

(2.19)

$$B_{n+1} = B_n + G_{n+1} \varepsilon_b(n+1)$$
$$= B_n + w^{-1} G'_{n+1} e_b(n+1)$$

où:

$e_b(n+1) = \varepsilon_b(n+1) / \varphi(n+1)$

et:

$\varepsilon_b(n+1) = x(n+1-L) - B_n^t X_{n+1}$

ε_b et e_b sont respectivement les erreurs de prédiction arrière a priori et a posteriori.

L'énergie d'erreur de prédiction arrière est:

(2.20)

$$E_b(n+1) = \sum_{p=1}^{n+1} w^{n+1-p} x^2(p-L) - B_{n+1}^t r_{n+1}^b$$
$$= w[E_b(n) + \varepsilon_b(n+1)\varepsilon_b(n+1) / \rho(n+1)]$$

Nous allons, maintenant, montrer comment interviennent toutes ces notions dans le calcul récursif du gain de Kalman. Pour ce faire, nous rappelons d'abord le lemme d'inversion de matrices partitionnées (L2) [2]:

Etant donné R et son inverse $P=R^{-1}$, on peut écrire:

$$R = \begin{bmatrix} R_{11} & R_{12} \\ R_{21} & R_{22} \end{bmatrix}, \qquad P = R^{-1} = \begin{bmatrix} P_{11} & P_{12} \\ P_{21} & P_{22} \end{bmatrix}$$

avec:

$$P_{11} = \left(R_{11} - R_{12}R_{22}^{-1}R_{21}\right)^{-1} = R_{11}^{-1} + R_{11}^{-1}R_{12}P_{22}R_{21}R_{11}^{-1}$$
$$P_{12} = -R_{11}^{-1}R_{12}P_{22} = -P_{11}R_{12}R_{22}^{-1}$$
$$P_{21} = -R_{22}^{-1}R_{21}P_{11} = -P_{22}R_{21}R_{11}^{-1}$$
$$P_{22} = \left(R_{22} - R_{21}R_{11}^{-1}R_{12}\right)^{-1} = R_{22}^{-1} + R_{22}^{-1}R_{21}P_{11}R_{12}R_{22}^{-1}$$

Nous appliquons, à présent, ce lemme à la matrice de dimension (L+1)x(L+1) d'autocorrélation du signal d'entrée qu'on peut écrire de deux façons:

(2.21)

$$R_{L+1}(n+1) = \begin{bmatrix} \sum_{p=1}^{n+1} w^{n+1-p} x^2(p) & [r_{n+1,L}^a]^t \\ r_{n+1,L}^a & R_L(n) \end{bmatrix} = \begin{bmatrix} R_L(n+1) & r_{n+1,L}^b \\ [r_{n+1,L}^b]^t & \sum_{p=1}^{n+1} w^{n+1-p} x^2(p-L) \end{bmatrix}$$

pour aboutir -en omettant les détails- aux deux expressions suivantes:

(2.22)

$$R_{L+1}^{-1}(n+1)=\frac{1}{E_a(n+1)}\begin{bmatrix}1\\-A_{n+1}\end{bmatrix}\begin{bmatrix}1 & -A_{n+1}^t\end{bmatrix}+\begin{bmatrix}0 & \underline{O}^t\\ \underline{O} & R_L^{-1}(n)\end{bmatrix}$$

(2.23)

$$R_{L+1}^{-1}(n+1)=\frac{1}{E_b(n+1)}\begin{bmatrix}-B_{n+1}\\1\end{bmatrix}\begin{bmatrix}-B_{n+1}^t & 1\end{bmatrix}+\begin{bmatrix}R_L^{-1}(n+1) & \underline{O}\\ \underline{O}^t & 0\end{bmatrix}$$

En multipliant (2.22) et (2.23) par $\begin{bmatrix}x(n+2)\\X_{n+1}\end{bmatrix}$ et en égalant, on obtient:

(2.24)

$$G'_{n+1}=ZG'_n-m_a(n+1)ZA_n+m_b(n+1)B_n+m_a(n+1)u_1$$

où:

$u_1=[1\quad 0.....0]^t$

$m_a(n+1)=\varepsilon_a(n+1)/E_a(n)$

$m_b(n+1)=\varepsilon_b(n+1)/E_b(n)$

et Z est la matrice de décalage.

On a aussi:

(2.25)

$$m_b(n+1)=g'_L(n)-m_a(n+1)a_L(n)$$

On en déduit le calcul récursif de ρ:

(2.26)

$$\rho(n+1)=\rho(n)+m_a(n+1)\varepsilon_a(n+1)-m_b(n+1)\varepsilon_b(n+1)$$

Résumons l'algorithme FRLS, qui est en fait celui proposé dans [13]:

(2.27)

a) $\varepsilon_a(n+1)=x(n+1)-X_n^tA_n$

b) $m_a(n+1)=\varepsilon_a(n+1)/E_a(n)$

c) $m_b(n+1)=g'_L(n)-m_a(n+1)a_L(n)$

d) $E_a(n+1)=w[E_a(n)+\varepsilon_a(n+1)\varepsilon_a(n+1)/\rho(n)]$

e) $G'_{n+1}=ZG'_n-m_a(n+1)ZA_n+m_b(n+1)B_n+m_a(n+1)u_1$

f) $\varepsilon_b(n+1)=m_b(n+1)E_b(n)$

g) $\rho(n+1) = \rho(n) + \varepsilon_a(n+1)m_a(n+1) - \varepsilon_b(n+1)m_b(n+1)$

h) $E_b(n+1) = w[E_b(n) + \varepsilon_b(n+1)\varepsilon_b(n+1)/\rho(n+1)]$

i) $A_{n+1} = A_n + G'_n\varepsilon_a(n+1)/\rho(n)$

j) $B_{n+1} = B_n + G'_{n+1}\varepsilon_b(n+1)/\rho(n+1)$

k) $\varepsilon(n+1) = y(n+1) - X^t_{n+1}H_n$

l) $H_{n+1} = H_n + G'_{n+1}\varepsilon(n+1)/\rho(n+1)$

L'initialisation de l'algorithme est:

$A_0=B_0=G'_0=Q$; $E_a(0)=E_0$; $E_b(0)=w^{-L}E_0$

où E_0 est une constante positive faible.

Nous remarquons que le nombre de multiplications de cet algorithme est à peu près égal à 7L tandis qu'il est proportionnel à L^2 pour l'algorithme RLS. Le gain est donc considérable. La première version rapide du gain de Kalman a été proposée par Ljung et *al.* [14], elle nécessite 11L multiplications. Cioffi et *al.* ont, quant à eux, donné une approche géométrique de cet algorithme rapide [15].

Les algorithmes RLS et FRLS sont théoriquement équivalents. Cependant, en pratique (en précision finie), l'algorithme FRLS diverge à cause de l'accumulation des erreurs d'arrondi qui sont engendrées par les opérations de quantification, généralement effectuées après les multiplications et les divisions.

Il existe actuellement des techniques de stabilisation numérique de cet algorithme avec une faible augmentation du nombre de multiplications (de l'ordre de L) [24,25]. Notons, cependant, que l'effet de stabilisation des méthodes proposées n'est pas totalement garanti mathématiquement.

Pour certaines applications (l'annulation d'écho par exemple), l'algorithme FRLS est encore difficilement exploitable, à cause de sa charge de calcul, qui, malgré une réduction importante par rapport au RLS, demeure encore élevée. Des difficultés importantes se posent également au niveau de la structure de l'algorithme et de la précision des calculs.

2.3. L'ALGORITHME DE NEWTON

A partir de ce paragraphe, nous allons commencer à faire des approximations que nous justifierons au fur et à mesure, afin de retrouver des algorithmes plus simples à mettre en œuvre.

Si le signal d'entrée est stationnaire, on peut approcher la matrice d'autocorrélation R(n) par une matrice R_{xx} constante, telle que:

$$R_{xx} = E\left\{X_n X_n^t\right\}$$

Ainsi, on n'aura plus à inverser cette matrice à tout instant en fonction des nouvelles données.

L'algorithme RLS peut être, donc, sous cette condition, approché par l'algorithme suivant:

(2.28)

a) $H_{n+1} = H_n + \mu' R_{xx}^{-1} X_n e(n)$

b) $e(n) = y(n) - \hat{y}(n) = y(n) - X_n^t H_n$

où μ' est une constante strictement positive. Ce scalaire est égal à 1 si le signal est rigoureusement stationnaire, mais dans la pratique on le prend plus petit, pour ajuster la vitesse de convergence.

Les équations (2.28) forment l'algorithme de Newton [4] qui a, en général, sous l'hypothèse de faibles variations des statistiques du signal d'entrée, les mêmes performances que l'algorithme RLS, puisque les statistiques du signal ne changent presque pas au cours du temps.

Il est à remarquer que l'algorithme de Newton n'apporte aucune réduction significative de la charge de calcul par rapport au RLS.

Dans ce qui suit, nous formulons autrement l'algorithme de Newton, pour faire le lien avec d'autres algorithmes.

La matrice R_{xx} est supposée définie positive, elle peut donc être décomposée de façon unique en éléments propres:

(2.29)

$$R_{xx} = Q \Lambda Q^t$$

où Q est la matrice orthogonale contenant les vecteurs propres de R_{xx} et Λ la matrice diagonale contenant les valeurs propres correspondantes:

$QQ^t=Q^tQ=I$

L'inverse de la matrice R_{xx} est:

(2.30)

$$R_{xx}^{-1} = Q\Lambda^{-1}Q^t$$

En remplaçant (2.30) dans (2.28a) et en multipliant de chaque côté par Q^t, on a:

(2.31)

$$Q^t H_{n+1} = Q^t H_n + \mu' Q^t Q \Lambda^{-1} Q^t X_n e(n)$$

Posons:

$$H'_n = Q^t H_n = \left[h'_0(n) \quad h'_1(n).....h'_{L-1}(n)\right]^t$$

et:

$$X'_n = Q^t X_n = \left[x'_0(n) \quad x'_1(n).....x'_{L-1}(n)\right]^t$$

l'équation (2.31) devient:

(2.32)

$$H'_{n+1} = H'_n + \mu' \Lambda^{-1} X'_n e(n)$$

et:

(2.33)

$$\hat{y}(n) = X_n^t H_n = X'^t_n H'_n$$

D'autre part:

$$R_{x'x'} = E\left\{X'_n X'^t_n\right\} = Q^t R_{xx} Q = \Lambda$$

l'équation (2.32) peut encore s'écrire:

(2.34)

$$H'_{n+1} = H'_n + \mu' R_{x'x'}^{-1} X'_n e(n)$$

$R_{x'x'}$ est une matrice diagonale et ses éléments non nuls sont faciles à déterminer:

$$R_{x'x'} = \Lambda = \text{diag}\left\{\lambda_0 \quad \lambda_1.....\lambda_{L-1}\right\}$$

où:

$$\lambda_i = E\left\{x'^2_i(n)\right\}, \; i = 0,1,.....,L-1$$

Remarque: Le vecteur X' ne conserve pas la propriété de décalage, c'est-à-dire que les vecteurs X'_{n-1} et X'_n n'ont pas (L-1) composantes en commun.

Il est clair que les expressions (2.28a) et (2.34) sont rigoureusement équivalentes, donc si l'on connait Q, on en déduit immédiatement le vecteur X' puis les valeurs propres λ_i. Cette transformée idéale qui nous permet de passer d'une forme à l'autre, est la transformée de Karhunen-Lœve (KLT) qui dépend des statistiques du signal d'entrée.

L'algorithme de Newton sous les formes (2.28a) ou (2.34) est inutilisable en pratique car on ne dispose pas de la matrice R_{xx} et encore moins de la matrice Q ou de moyens rapides pour les calculer.

2.4. L'ALGORITHME "TRANSFORM-DOMAIN LMS" (TDLMS)

Nous avons vu que Q est une matrice orthogonale et que la transformée optimale est la KLT, pour conserver l'équivalence avec l'algorithme de Newton. Toutefois, en pratique on l'approche par une transformée orthogonale (W) connue telle que la DFT, la DCT ("Discrete Cosine Transform") ou encore la DHT ("Discrete Hartley Transform"), car ces dernières s'avérent être de bonnes approximations de la KLT pour des signaux fortement corrélés. L'algorithme TDLMS (voir figure 2.1) tel qu'il a été proposé dans [26] est obtenu en posant dans l'algorithme de Newton: $Q^t=W$.

En réalité, si l'on approxime Q^t par W, la matrice de corrélation R_{xx} sera approchée par une autre matrice R'_{xx} telle que:

$$R'_{xx}=W^t \, \mathrm{diag}\{W R_{xx} W^t\} \, W=W^t \, \mathrm{diag}\{\Lambda'\} \, W$$

où cette fois-ci, Λ' n'est plus forcément diagonale.

Avec l'approche du paragraphe précédent, nous constatons que l'algorithme TDLMS est très proche de l'algorithme de Newton. Le TDLMS peut être, aussi, obtenu à partir du critère suivant:

$$J_{TD} = E\left\{\left(y(n)-X'^t_n H'\right)^t\left(y(n)-X'^t_n H'\right)\right\}$$

Nous allons voir maintenant comment utiliser la DCT dans un tel algorithme.

Il existe en fait deux méthodes "rapides" différentes pour programmer la DCT. La première est fondée sur les techniques de FFT dont le nombre d'opérations est proportionnel à $(L\log_2 L)$, la seconde sur la récursivité mais avec une charge de calcul moindre. Nous avons opté pour la

seconde méthode.

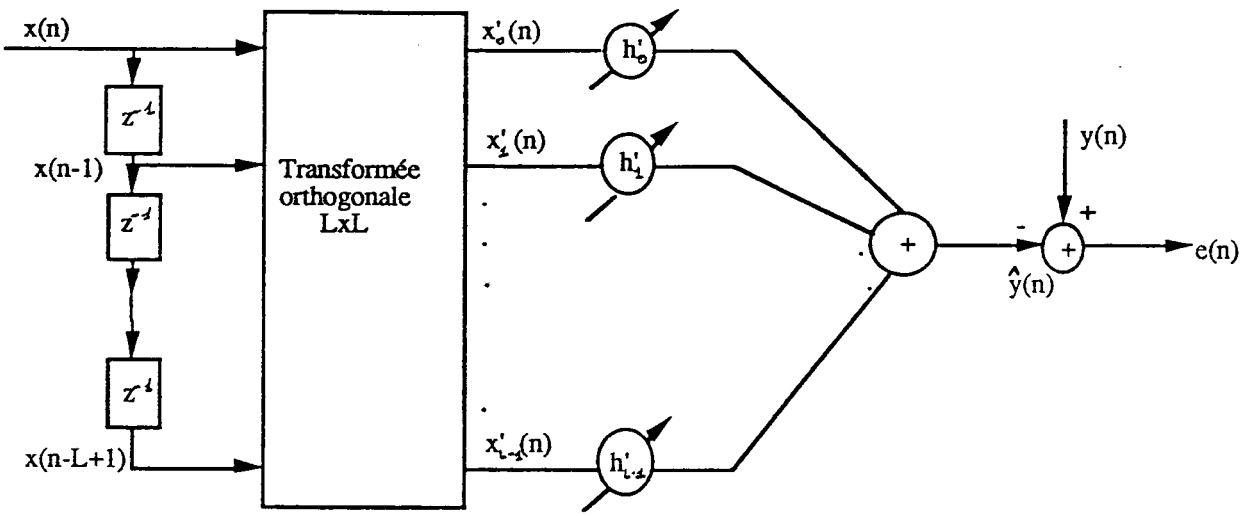


Figure 2.1: L'algorithme TDLMS

Les éléments x'_k du vecteur X' sont calculés à l'aide de la transformée en cosinus discrète et du vecteur X comme suit:

(2.35)

$$x'_k(n) = c_k \sum_{i=0}^{L-1} x(n-i)\cos\left[\frac{\pi(2i+1)}{2L}k\right], \quad k = 0,1,2,.....,L-1$$

avec:

$$c_0 = 1/\sqrt{L} \quad \text{et} \quad c_k = \sqrt{2/L} \text{ pour } k \neq 0$$

Posons:

$$h_k(n) = c_k \cos\left[\frac{\pi(2n+1)}{2L}k\right]$$

alors:

(2.36)

$$H_k(z) = \frac{X'_k(z)}{X(z)} = c_k \sum_{n=0}^{L-1} z^{-n}\cos\left[\frac{\pi(2n+1)}{2L}k\right]$$

$$= c_k \frac{1 - z^{-1} - (-1)^k z^{-L} + (-1)^k z^{-(L+1)}}{1 - 2\cos\left[\frac{\pi}{L}k\right]z^{-1} + z^{-2}} \cos\left[\frac{\pi}{2L}k\right]$$

D'où:

(2.37)

$$x'_k(n) = 2\cos(\pi k / L)x'_k(n-1) - x'_k(n-2)$$
$$+ c_k \cos(\pi k / 2L)\left[x(n) - x(n-1) - (-1)^k (x(n-L) - x(n-L-1)\right]$$

On a ainsi obtenu une équation de récurrence sur le temps, pour déterminer les éléments de X' sans avoir à faire à chaque instant le produit d'une matrice carrée par un vecteur.

Le calcul des valeurs propres est donné par la formule suivante (en faisant l'hypothèse que l'on peut remplacer une moyenne d'ensemble par une moyenne temporelle):

(2.38)

$$\lambda_k(n) = \beta\lambda_k(n-1) + (1-\beta)x'^2_k(n), \quad k = 0,1,.....,L-1$$

La principale difficulté de l'algorithme TDLMS est le calcul du vecteur X'. Une fois celui-ci trouvé, on en déduit facilement une approximation des valeurs propres à l'aide de la formule (2.38) puis ensuite l'adaptation du filtre H'.

Les figures 2.3 a, b, c donnent les courbes d'erreur des algorithmes FRLS, TDLMS (utilisant la DCT) et LMS, pour un signal d'entrée fortement corrélé (voir conditions de simulations au chap. 4). On peut voir que le TDLMS converge bien plus vite que le LMS et se rapproche assez bien du FRLS.

Si maintenant la transformée orthogonale est une DFT, on parle de filtrage adaptatif fréquentiel. Mais en général, les algorithmes dans le domaine fréquentiel sont des algorithmes par bloc comme nous le verrons plus loin.

L'algorithme TDLMS est encore compliqué. Dans le cas de la DCT il nécessite, à peu près, 5L multiplications et 2L divisions. Il est cependant possible de diminuer ce nombre d'opérations, mais en perdant au niveau de la convergence.

2.5. L'ALGORITHME LMS

L'objectif dans cette section est de retrouver un algorithme très connu (LMS) à partir du TDLMS par une autre approximation. En effet, prenons un gain constant dans l'algorithme TDLMS, c'est-à-dire posons:

$$\mu'\Lambda^{-1} = \mu I$$

cette approximation est assez brutale et elle n'est en réalité justifiée que pour un signal d'entrée

dont le spectre est plat (bruit blanc, par exemple); on a alors l'équation suivante pour l'adaptation des coefficients du filtre:

(2.39)

$$WH_{n+1} = WH_n + \mu W X_n e(n)$$

soit en multipliant par W^t des deux côtés:

(2.40)

$$H_{n+1} = H_n + \mu X_n e(n)$$

avec:

$$e(n) = y(n) - X_n^t H_n$$

C'est l'algorithme classique du gradient stochastique proposé par Widrow et *al.* [8], qui est donc strictement équivalent à l'algorithme TDLMS à pas constant. Son nombre d'opérations est relativement faible, 2L multiplications. A l'origine, il a été obtenu à partir du critère des moindres carrés moyens:

$$J_{MCM} = E\{e^2(n)\}$$

On montre [12] que cet algorithme converge bien vers le filtre optimal de Wiener-Hopf si le pas d'adaptation vérifie la condition suivante:

$$0 < \mu < \frac{2}{LE\{x^2(n)\}}$$

De tous les algorithmes adaptatifs, il est de loin le plus utilisé en pratique pour sa simplicité, sa robustesse, et ses capacités en poursuite.

Cependant, sa vitesse de convergence dépend fortement du conditionnement de la matrice d'autocorrélation du signal d'entrée, c'est-à-dire du rapport de la plus grande valeur propre à la plus petite valeur propre de cette matrice ($\lambda_{max}/\lambda_{min}$). Plus ce rapport est grand et plus la convergence de l'algorithme est lente. En réalité, la convergence du LMS est optimale si le signal d'entrée est un bruit blanc, car dans ce cas, la matrice de corrélation est nulle sauf sur la diagonale principale (($\lambda_{max}/\lambda_{min}$)=1), et le LMS est une bonne approximation du RLS.

D'autre part, on peut montrer [4,12] que le temps de convergence de l'algorithme LMS est inversement proportionnel au pas d'adaptation tandis que l'erreur résiduelle lui est proportionnelle. Donc si μ est petit, l'algorithme sera lent à converger tandis que l'erreur de

fluctuation sera faible. Et inversement, si μ est grand, l'algorithme convergera plus vite mais avec une erreur plus importante. Par conséquent, il y aura toujours un compromis à faire entre la vitesse de convergence et l'erreur résiduelle.

Nous remarquons que le pas d'adaptation dépend essentiellement de la puissance du signal d'entrée. Donc si ce signal varie au cours du temps, il est indispensable de prendre un pas variable où la puissance du signal est estimée à chaque instant. On aura, par exemple:

$$\mu(n) = \frac{\alpha}{X_n^t X_n}, \quad 0 < \alpha < 2$$

c'est l'algorithme du gradient normalisé (NLMS).

Jusqu'ici, nous avons décrit des algorithmes où l'adaptation du filtre se fait à chaque échantillon de temps. Depuis déjà quelques années, il existe une autre classe d'algorithmes où l'on adapte le filtre une fois par bloc d'échantillons, pour profiter de la redondance des calculs et réduire, ainsi, la complexité arithmétique. Ces derniers algorithmes ne sont pas sans liens avec les premiers.

2.6. FORMULATION EXACTE PAR BLOC DE L'ALGORITHME LMS (FELMS)

Dans ce paragraphe, nous ne suivrons pas l'approche classique dérivant le BLMS, mais nous montrons, tout d'abord, qu'il est possible de dériver à partir du LMS un algorithme par bloc qui lui est strictement équivalent. Il découlera de cette formulation tous les algorithmes par bloc connus, aussi bien dans le domaine temporel que fréquentiel. L'objectif final est la compréhension de la formulation par bloc.

Soit N la taille de ce bloc, l'algorithme FELMS (sa démonstration détaillée est fournie au chapitre III) est représenté par [31]:

(2.41)

$$H_{n+1} = H_{n-N+1} + \mu \underline{X}^t(n) E_n$$

où $\underline{X}^t(n)$ est une matrice de taille LxN:

$$\underline{X}^t(n) = [X_{n-N+1} \quad X_{n-N+2} \cdots\cdots X_{n-1} \quad X_n]$$

E_n, est un vecteur contenant les N dernières erreurs:

$$E_n = [e(n-N+1) \quad e(n-N+2).....e(n-1) \quad e(n)]^t$$

avec:

(2.42)

$$E_n = G(n)[Y_n - Y'_n] = G(n)\varepsilon_n$$

Y_n est formé à partir des N dernières sorties du système:

$$Y_n = [y(n-N+1) \quad y(n-N+2).....y(n-1) \quad y(n)]^t$$

et:

(2.43)

$$Y'_n = \underline{X}(n)H_{n-N+1}$$

La matrice G(n) de taille NxN est triangulaire:

(2.44)

$$G(n) = [\mu S(n) + I]^{-1}$$

avec:

(2.45)

$$S(n) = \begin{bmatrix} 0 & 0 & \cdots\cdots & 0 \\ s_1(n-N+2) & 0 & \cdots\cdots & 0 \\ s_2(n-N+3) & s_1(n-N+3) & & \vdots \\ \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ s_{N-1}(n) & s_{N-2}(n) & \cdots\cdots s_1(n) & 0 \end{bmatrix}$$

et:

$$s_i(n) = X_n^t X_{n-i}\,, \quad i = 1,2,.....,N-1$$

S est triangulaire, Gε s'obtient très facilement par substitution. D'autre part, les éléments de la matrice S peuvent être calculés récursivement.

Les algorithmes LMS et FELMS sont mathématiquement équivalents, il y a juste un

réarrangement des mêmes équations. La seule différence est que le FELMS adapte le filtre une fois tous les N instants et l'on n'a pas accès aux valeurs de H aux instants non multiples de N.

Nous proposons dans ce qui suit de prouver que tous les algorithmes fréquentiels ou par bloc classiques se déduisent facilement de cette nouvelle formulation par bloc.

2.7. L'ALGORITHME "BLOCK-LMS" (BLMS)

Cet algorithme tel qu'il a été présenté dans [18] peut être vu comme une approximation du FELMS. En effet, si dans ce dernier on prend G(n)=I et un pas d'adaptation éventuellement différent du LMS, on obtient les équations suivantes:

(2.48)

a) $Y'_n = \underline{X}(n)H_{n-N+1}$

b) $\varepsilon_n = Y_n - Y'_n$

c) $H_{n+1} = H_{n-N+1} + \frac{\mu_B}{N}\underline{X}^t(n)\varepsilon_n$

qui représentent l'algorithme BLMS (voir figure 2.2). Ses auteurs l'on déduit de la fonction coût suivante:

$$J_B = \frac{1}{N}E\left\{\varepsilon_n^t \varepsilon_n\right\}$$

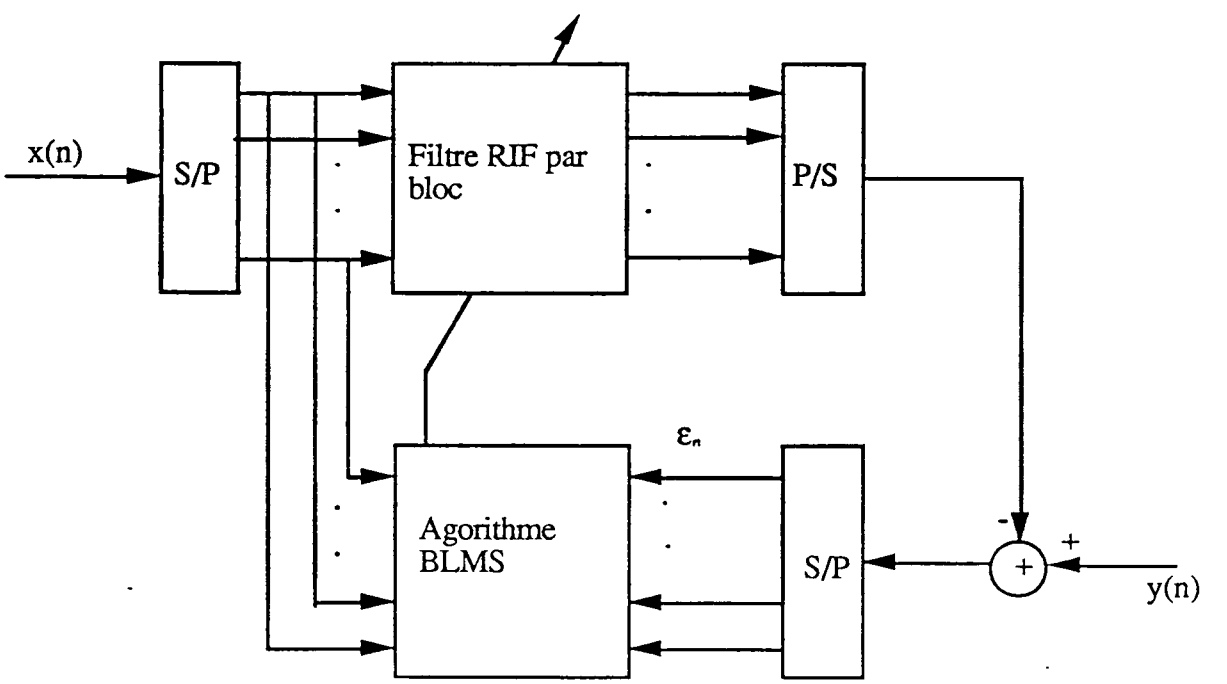


Figure 2.2: L'algorithme BLMS. S/P=convertisseur série/ parallèle. P/S=convertisseur parallèle/série

On montre [31] que cet algorithme a les mêmes performances que le FELMS (donc que le LMS) si le signal d'entrée est un bruit blanc, et ceci quelle que soit la taille du bloc. Car on remarque que la matrice S peut être approximée à la partie inférieure de la matrice d'autocorrélation du signal d'entrée. Et dans le cas d'un bruit blanc, S est nulle et G est bien égale à la matrice identité. Dans ce cas particulier, le pas d'adaptation du LMS (μ) est identique au pas du BLMS (μ_B/N).

Les choses sont différentes pour des entrées corrélées. En effet, on constate que le BLMS converge moins vite que le LMS, et plus la taille du bloc sera grande et plus l'algorithme mettra du temps à converger. L'erreur résiduelle sera aussi affectée.

On peut donc affirmer que le BLMS est une bonne approximation du LMS si le signal d'entrée est un bruit blanc, comme le LMS est une bonne approximation du TDLMS pour le même type de signal.

Le principal avantage de l'algorithme BLMS réside dans la possibilité d'utiliser la FFT (ou tout autre technique permettant le filtrage rapide) comme intermédiaire de calcul afin de réduire sa complexité arithmétique. C'est un premier exemple d'algorithme dans le domaine fréquentiel.

2.8. LES ALGORITHMES FREQUENTIELS

Comme nous l'avons déjà dit, l'utilisation de la FFT dans les algorithmes par bloc sert d'abord à diminuer leur nombre d'opérations, tout en conservant l'équivalence entre les deux versions (temporelle et fréquentielle). Ceci est possible, parce que la matrice intervenant dans l'adaptation du filtre et dans le calcul de l'erreur a une structure particulière.

L'algorithme fréquentiel le plus classique est certainement le "Fast-LMS" (FLMS) proposé par Ferrara [16]. Il a exactement le même comportement que le BLMS, mais il a l'avantage d'être plus rapide en temps de calcul.

Nous allons, à présent, expliciter l'algorithme FLMS.

Choisissons la taille de bloc égale à la longueur du filtre (L=N). Dans ce cas, la matrice $\underline{X}(n)$ est carrée (LxL) et a la structure de Toeplitz. Le fait que cette matrice soit carrée est important pour la suite (possibilité de diagonaliser). D'autre part, il est bien connu qu'une matrice circulante (C) peut être diagonalisée par une matrice de Fourier (F): $D=FCF^{-1}$, où D est une matrice diagonale dont les éléments sont la sortie de la DFT de la première colonne de la matrice C [2].

Il est possible d'appliquer ce résultat au BLMS en transformant la matrice de Toeplitz $\underline{X}(n)$ en une matrice circulante, juste en doublant sa taille.

En se servant de la technique précédente, l'équation (2.48a) de l'algorithme BLMS devient:

(2.49)

$$\begin{bmatrix} Y''_n \\ Y'_n \end{bmatrix} = \begin{bmatrix} \underline{X}'(n) & \underline{X}(n) \\ \underline{X}(n) & \underline{X}'(n) \end{bmatrix} \begin{bmatrix} H_{n-L+1} \\ \underline{O} \end{bmatrix} = C(n) \begin{bmatrix} H_{n-L+1} \\ \underline{O} \end{bmatrix}$$

2Lx1 2Lx2L 2Lx1

où $\underline{X}'(n)$ est une matrice de Toeplitz (LxL) choisie de telle sorte que la matrice C(n) (2Lx2L) soit circulante, Y''_n est un vecteur (Lx1) que l'on n'explicite pas et $\underline{O}$ le vecteur nul (Lx1).

Le même principe s'applique à l'équation d'adaptation de H du BLMS:

(2.50)

$$\begin{bmatrix} H_{n+1} \\ \underline{O} \end{bmatrix} = \begin{bmatrix} H_{n-L+1} \\ \underline{O} \end{bmatrix} + \frac{\mu_B}{L} W_F C^t(n) \begin{bmatrix} \underline{O} \\ \varepsilon_n \end{bmatrix}$$

où:

$W_F = \text{diag}\{1,1,\ldots 1,0,0,\ldots 0\}$

Maintenant, en explicitant la forme de la matrice circulante C et en appliquant la FFT aux expressions (2.49) et (2.50), on a:

(2.51)

$$\begin{bmatrix} Y''_n \\ Y'_n \end{bmatrix} = F^{-1} D(n) F \begin{bmatrix} H_{n-L+1} \\ \underline{O} \end{bmatrix}$$

et:

(2.52)

$$F \begin{bmatrix} H_{n+1} \\ \underline{O} \end{bmatrix} = F \begin{bmatrix} H_{n-L+1} \\ \underline{O} \end{bmatrix} + \frac{\mu_B}{L} F W_F F^{-1} D^*(n) F \begin{bmatrix} \underline{O} \\ \varepsilon_n \end{bmatrix}$$

où D(n)=diag{FFT[x(n-2L+1),....., x(n-L), x(n-L+1),....., x(n)]}

Les équations (2.51) et (2.52) résument l'algorithme FLMS qui est, rappelons le, mathématiquement équivalent au BLMS (dans le cas où L=N). Si L est une puissance de 2, le nombre de multiplications par point de sortie du premier est à peu près égal à $5\log_2 L$, alors qu'il est de 2L pour le LMS. On voit donc l'intérêt de travailler avec la FFT.

Il existe un autre avantage de la FFT: c'est la possibilité de remplacer la matrice $(\mu_B/L)I$ par une matrice diagonale (Λ^{-1}) où chacun des coefficients du filtre est adapté à l'aide d'un pas différent et variable au cours du temps (même principe entre le LMS et le TDLMS):

(2.53)

$$\Lambda^{-1}(n) = \alpha \mathrm{diag}\{P_0(n) \quad P_1(n).....P_{2L-1}(n)\}^{-1}$$

et, par exemple:

$$P_k(n) = \beta P_k(n-L) + (1-\beta)d_k(n)d_k^*(n), \quad k = 0,1,.....,2L-1$$

où P_k est la puissance du signal à la fréquence k et β est un facteur d'oubli ($0<\beta<1$).

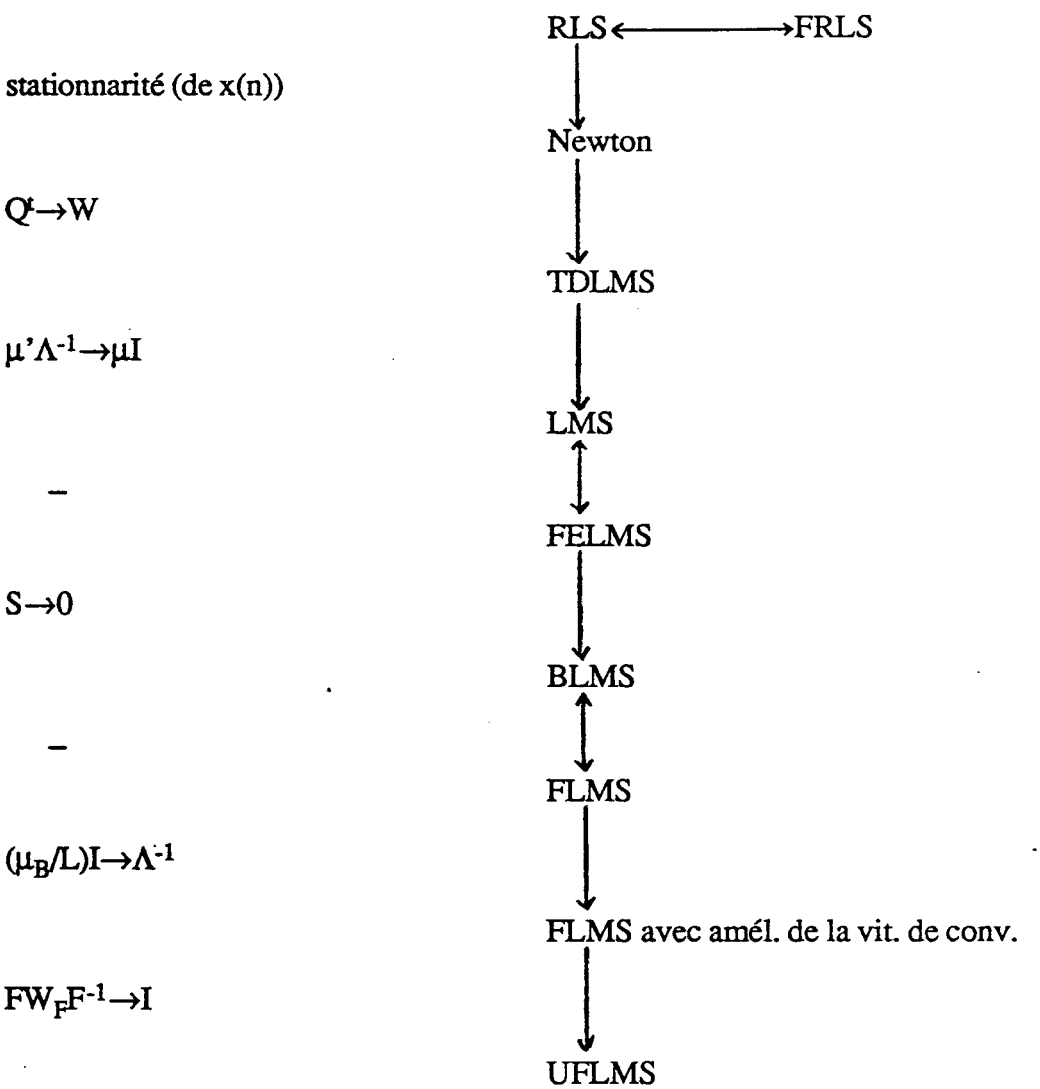
C'est la seule différence, au niveau du comportement adaptatif de l'algorithme, entre le BLMS et le FLMS. Alors que dans le premier cas, l'énergie intervenant dans le pas d'adaptation est la même pour tous les coefficients du filtre, dans le second cas elle est indépendante pour chacun d'entre eux grâce aux propriétés de la DFT.

Il est intéressant de faire l'analogie entre le BLMS et le FLMS avec amélioration de la vitesse de convergence d'une part, et le LMS et le TDLMS d'autre part. Les premiers sont des algorithmes par bloc avec dans un cas, un pas d'adaptation identique pour tous les coefficients et dans l'autre cas, un pas différent pour chacun d'entre eux. Les seconds sont des algorithmes échantillon par échantillon où le même principe se retrouve.

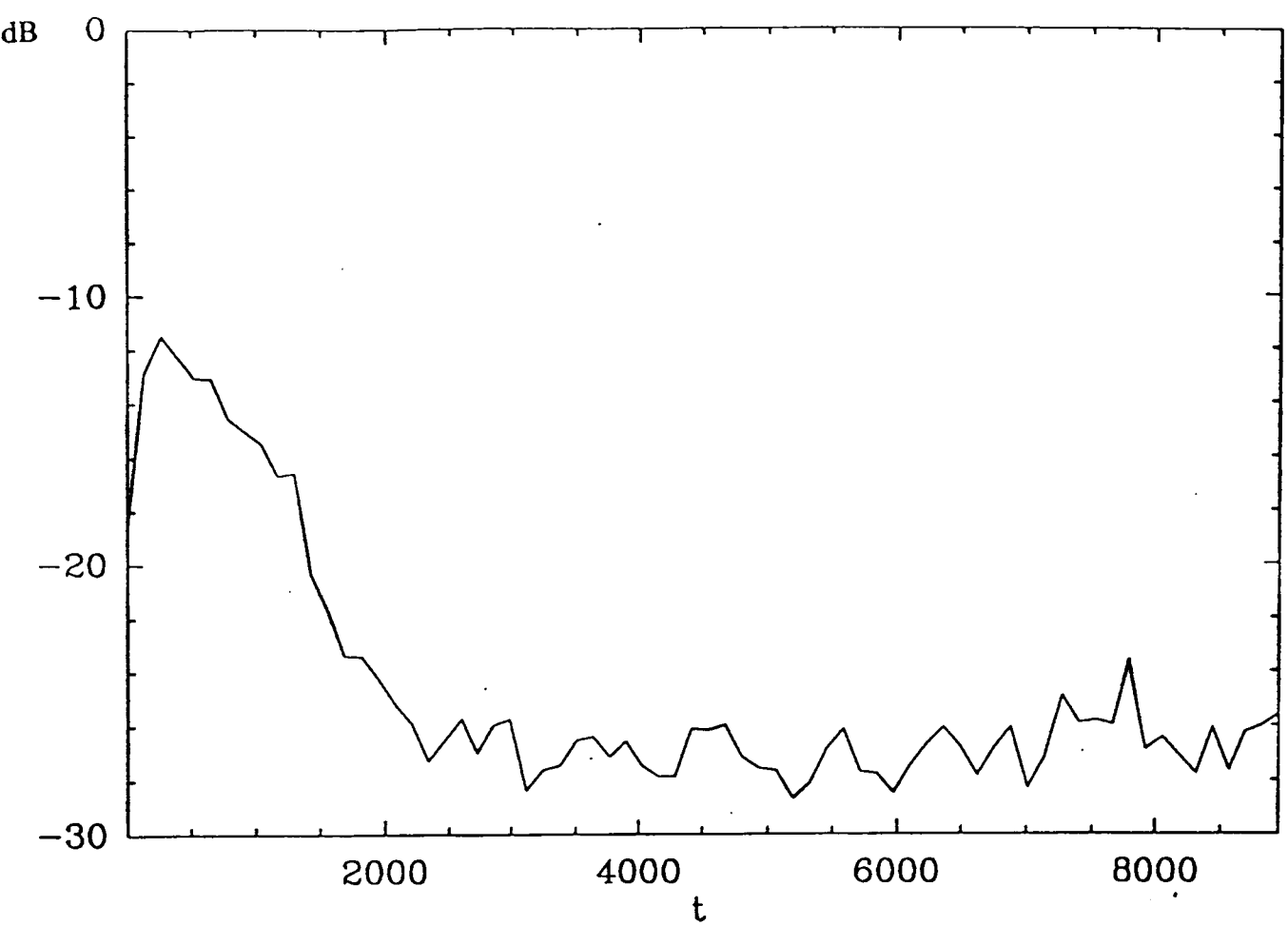
Il est encore possible de simplifier le FLMS. En effet, en approximant la matrice FW_FF^{-1} de l'expression (2.52) par la matrice identité, on obtient l'algorithme UFLMS (Unconstrained Frequency-domain LMS) [17]. On aura ainsi 2 FFT en moins à calculer par bloc, au prix d'une erreur résiduelle plus grande et une vitesse de convergence moins rapide que le FLMS, car on

n'a plus une véritable opération de filtrage. On peut, cependant, démontrer la convergence.

Nous avons montré dans ce chapitre, qu'à partir du RLS, les principaux algorithmes connus pouvaient être déduits par approximations successives. Le passage de l'algorithme RLS à l'algorithme TDLMS s'est fait par l'intermédiaire de l'algorithme de Newton, tandis que le passage aux algorithmes par bloc s'est fait à partir du LMS qui lui même peut être vu comme un cas particulier du TDLMS. D'autre part, nous avons obtenu les algorithmes fréquentiels connaissant les algorithmes par bloc grâce aux structures des matrices circulantes et de Toeplitz qui apparaissent dans ces algorithmes. Aucun des algorithmes présentés n'a été dissocié des autres. Tous ont été dérivés avec un objectif précis: réduction de la complexité arithmétique. Le nombre d'opérations par point de sortie du RLS sans utiliser le lemme (L1) est proportionnel à L^3, alors qu'il est proportionnel à $\log_2 L$ pour les algorithmes FLMS et UFLMS. En traitement numérique du signal, ce gain en complexité est très important (surtout pour un filtre long).



-Lien entre les différents algorithmes-



(b)

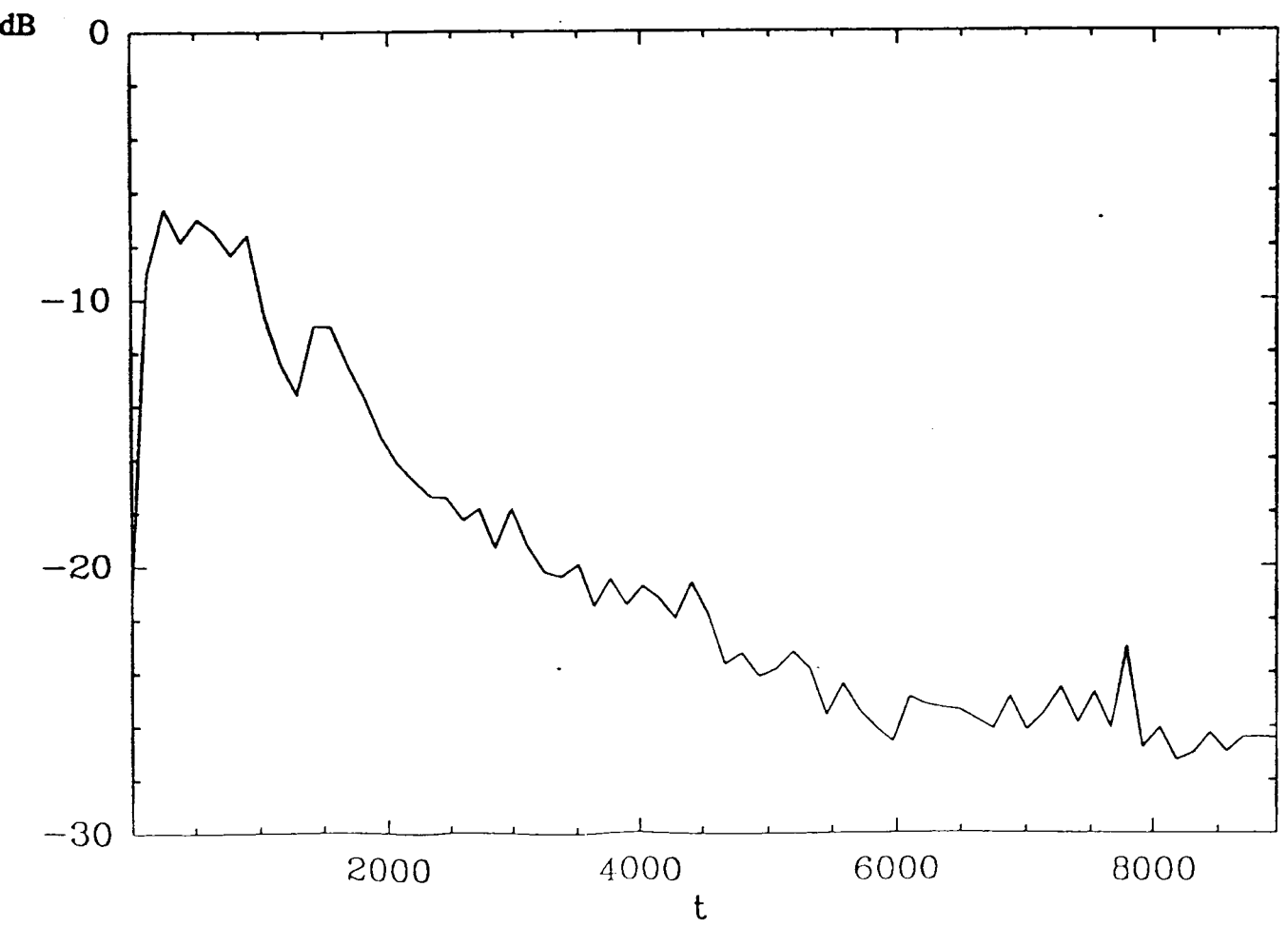


Fig. 2.3: Courbes d'erreur des algorithmes FRLS (a) et TDLMS (b)

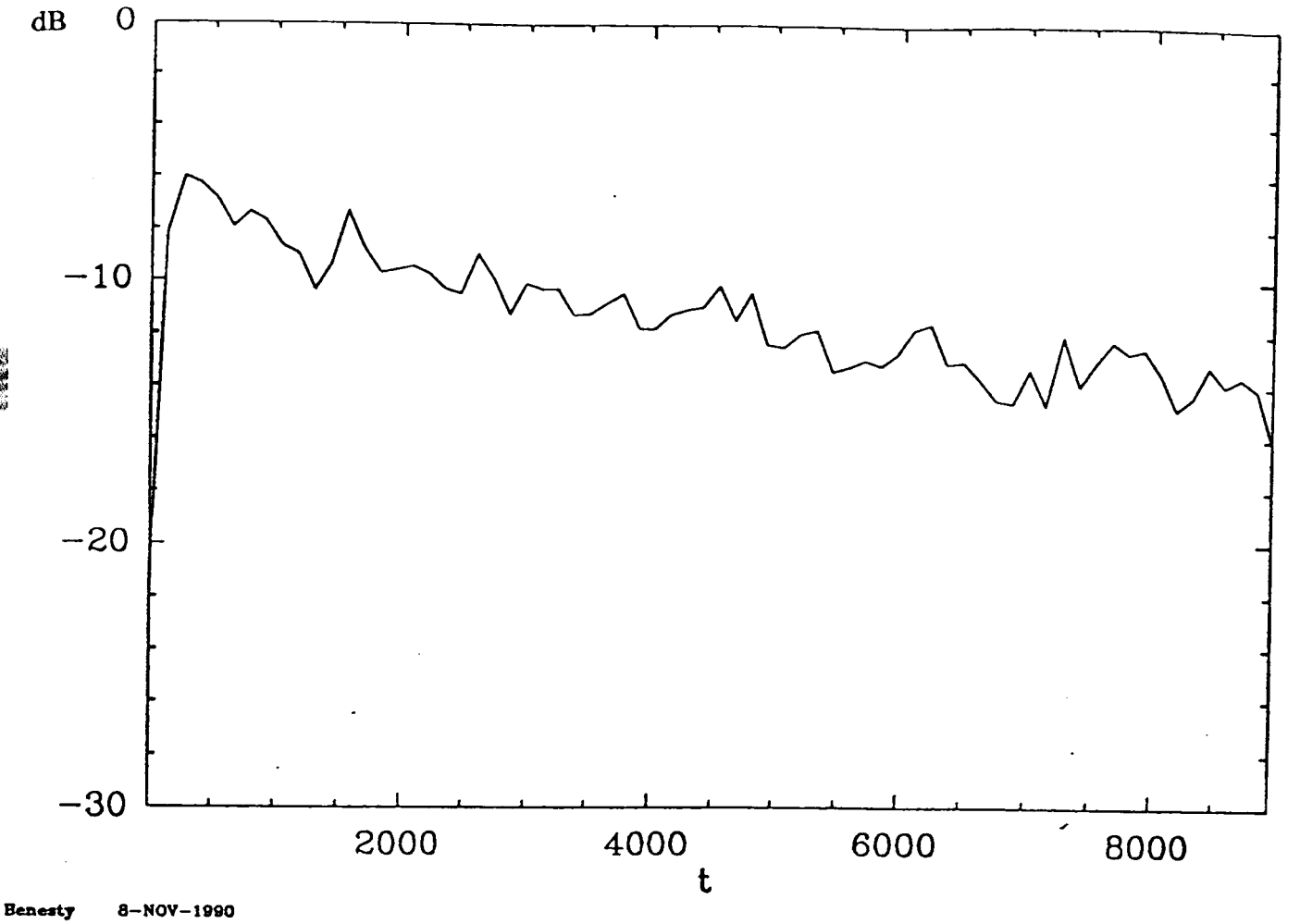


Fig. 2.3c: Courbe d'erreur de l'algorithme LMS

CHAPITRE III

UN ALGORITHME LMS RAPIDE

(au sens de la complexité arithmétique)

3.1. INTRODUCTION

Il est certainement impossible de diminuer le nombre d'opérations de l'algorithme LMS sous forme séquentielle habituelle sans faire d'approximations. Un moyen de s'en sortir est de travailler par bloc, pour profiter de la redondance des calculs due aux propriétés de décalage du signal d'entrée, en essayant de ne pas perturber le comportement adaptatif du filtre. C'est pourquoi nous nous sommes attachés à rechercher un algorithme par bloc équivalent à un algorithme séquentiel. L'idée même de cette possibilité peut sembler, parfois, difficile à admettre, car dans un cas on a accès aux valeurs du filtre à chaque instant, tandis que dans l'autre cas, seulement une fois par bloc. Cependant, nous avons déjà indiqué une telle formulation au chapitre II (FELMS) répondant à cette spécification. Nous la reprenons en la détaillant pour montrer comment procéder afin de diminuer la charge de calcul.

Comme nous l'avons déjà signalé, non seulement les algorithmes par bloc connus jusqu'alors avaient un comportement différent du LMS mais de plus la taille de ce bloc (N) devait être égale à la longueur du filtre (L) pour pouvoir utiliser les techniques de calculs rapides (la FFT le plus souvent), afin de réduire leur complexité arithmétique. Cette contrainte (L=N) est difficilement supportable pour certaines applications nécessitant des filtres longs, à cause du nombre de mémoires nécessaires et du retard trop important apporté au traitement du signal. De plus, ces techniques ne sont pas intéressantes pour des filtres de petites longueurs.

Au prochain paragraphe, nous appliquons les techniques "Fast FIR" [19], élaborées pour le filtrage fixe, à l'algorithme FELMS, mais cette fois portant sur un couple "filtrage - mise à jour des coefficients".

Ainsi, la réduction de la complexité arithmétique est toujours possible, même pour des tailles de blocs très petites (N=2, par exemple) et pour des filtres à peu de coefficients. De plus, dans notre approche, la taille du bloc est indépendante de la longueur du filtre et l'algorithme résultant

(FELMS) se comporte rigoureusement comme le LMS.

Les techniques proposées au paragraphe 3.2 ne sont efficaces que pour des petites tailles de blocs. Pour des tailles de blocs moyennes, il est plus intéressant d'utiliser la FFT comme intermédiaire de calcul, car elle réduit davantage la complexité arithmétique et la capacité mémoire requise.

Dans la section 3.3, nous montrons qu'il est possible d'utiliser la FFT comme intermédiaire de calcul dans l'algorithme FELMS. Notre approche est originale puisque la taille de la FFT est liée à la dimension du bloc et non à la longueur du filtre comme c'est la cas des algorithmes adaptatifs fréquentiels classiques [16,17]. Ce point est important, car pour un certain retard toléré et donné, on pourra toujours trouver la dimension de la FFT adéquate.

3.2. PRINCIPE DE L'ALGORITHME LMS RAPIDE (FELMS)

3.2.1. Etude d'un exemple simple (N=2)

Notre objectif dans cette section est de fournir, à partir d'un exemple simple (N=2), un algorithme strictement équivalent au LMS avec un nombre d'opérations réduit. Cet exemple nous permettra de bien cerner ce nouvel algorithme sachant que le cas général qui est un peu plus compliqué se déduit sans trop de difficultés.

Ecrivons l'algorithme LMS à l'instant n-1:

(3.1)

a) $e(n-1) = y(n-1) - X_{n-1}^t H_{n-1}$

b) $H_n = H_{n-1} + \mu e(n-1) X_{n-1}$

et à l'instant n:

(3.2)

a) $e(n) = y(n) - X_n^t H_n$

b) $H_{n+1} = H_n + \mu e(n) X_n$

En remplaçant (3.1b) dans (3.2a), nous obtenons:

(3.3)

$$e(n) = y(n) - X_n^t H_{n-1} - \mu e(n-1) X_n^t X_{n-1}$$
$$= y(n) - X_n^t H_{n-1} - \mu e(n-1) s(n)$$

avec:

$$s(n) = X_n^t X_{n-1}$$

Nous pouvons mettre sous forme matricielle les équations (3.1a) et (3.3), pour mieux faire ressortir la structure de l'algorithme:

(3.4)

$$\begin{bmatrix} e(n-1) \\ e(n) \end{bmatrix} = \begin{bmatrix} y(n-1) \\ y(n) \end{bmatrix} - \begin{bmatrix} X_{n-1}^t \\ X_n^t \end{bmatrix} H_{n-1} - \begin{bmatrix} 0 & 0 \\ \mu s(n) & 0 \end{bmatrix} \begin{bmatrix} e(n-1) \\ e(n) \end{bmatrix}$$

ou bien:

$$\begin{bmatrix} 1 & 0 \\ \mu s(n) & 1 \end{bmatrix} \begin{bmatrix} e(n-1) \\ e(n) \end{bmatrix} = \begin{bmatrix} y(n-1) \\ y(n) \end{bmatrix} - \begin{bmatrix} X_{n-1}^t \\ X_n^t \end{bmatrix} H_{n-1}$$

d'où:

(3.5)

$$\begin{bmatrix} e(n-1) \\ e(n) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -\mu s(n) & 1 \end{bmatrix} \left[\begin{bmatrix} y(n-1) \\ y(n) \end{bmatrix} - \begin{bmatrix} X_{n-1}^t \\ X_n^t \end{bmatrix} H_{n-1} \right]$$

Le second terme de cette équation apparait comme étant le calcul de deux sorties successives d'un filtre à coefficients fixes. On peut ainsi appliquer les mêmes techniques que celles expliquées dans [19,20,21,22]:

(3.6)

$$\begin{bmatrix} X_{n-1}^t \\ X_n^t \end{bmatrix} H_{n-1} = \begin{bmatrix} x(n-1) & x(n-2).....x(n-L) \\ x(n) & x(n-1).....x(n-L+1) \end{bmatrix} \begin{bmatrix} h_0(n-1) \\ h_1(n-1) \\ \vdots \\ h_{L-1}(n-1) \end{bmatrix}$$

$$= \begin{bmatrix} x(n-1) & x(n-3)\ldots x(n-L+1) & x(n-2) & x(n-4)\ldots x(n-L) \\ x(n) & x(n-2)\ldots x(n-L+2) & x(n-1) & x(n-3)\ldots x(n-L+1) \end{bmatrix} \begin{bmatrix} h_0(n-1) \\ h_2(n-1) \\ \vdots \\ h_{L-2}(n-1) \\ h_1(n-1) \\ h_3(n-1) \\ \vdots \\ h_{L-1}(n-1) \end{bmatrix}$$

où chacun des vecteurs a été divisé en deux parties correspondant à ses versions sous-échantillonnées. On suppose que L est un entier pair.

Définissons:

$$A_0 = [x(n) \quad x(n-2)\ldots x(n-L+2)]^t$$

$$A_1 = [x(n-1) \quad x(n-3)\ldots x(n-L+1)]^t$$

$$A_2 = [x(n-2) \quad x(n-4)\ldots x(n-L)]^t$$

$$H^0_{n-1} = [h_0(n-1) \quad h_2(n-1)\ldots h_{L-2}(n-1)]^t$$

$$H^1_{n-1} = [h_1(n-1) \quad h_3(n-1)\ldots h_{L-1}(n-1)]^t$$

l'expression (3.6) devient:

(3.7)

$$\begin{bmatrix} X^t_{n-1} \\ X^t_n \end{bmatrix} H_{n-1} = \begin{bmatrix} A^t_1 & A^t_2 \\ A^t_0 & A^t_1 \end{bmatrix} \begin{bmatrix} H^0_{n-1} \\ H^1_{n-1} \end{bmatrix}$$

On peut appliquer les mêmes règles pour l'équation d'adaptation des coefficients du filtre. En remplaçant (3.1b) dans (3.2b), on a:

(3.8)

$$H_{n+1} = H_{n-1} + \mu e(n) X_n + \mu e(n-1) X_{n-1}$$

soit, avec les notations précédentes:

(3.9)

$$\begin{bmatrix} H^0_{n+1} \\ H^1_{n+1} \end{bmatrix} = \begin{bmatrix} H^0_{n-1} \\ H^1_{n-1} \end{bmatrix} + \mu e(n) \begin{bmatrix} A_0 \\ A_1 \end{bmatrix} + \mu e(n-1) \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}$$

L'ensemble suivant des deux équations est équivalent à l'algorithme LMS pour un bloc de

deux sorties:

(3.10)

$$\text{a)}\quad \begin{bmatrix} e(n-1) \\ e(n) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -\mu s(n) & 1 \end{bmatrix} \left[\begin{bmatrix} y(n-1) \\ y(n) \end{bmatrix} - \begin{bmatrix} A_1^t & A_2^t \\ A_0^t & A_1^t \end{bmatrix} \begin{bmatrix} H_{n-1}^0 \\ H_{n-1}^1 \end{bmatrix} \right]$$

$$\text{b)}\quad \begin{bmatrix} H_{n+1}^0 \\ H_{n+1}^1 \end{bmatrix} = \begin{bmatrix} H_{n-1}^0 \\ H_{n-1}^1 \end{bmatrix} + \mu \begin{bmatrix} A_1 & A_0 \\ A_2 & A_1 \end{bmatrix} \begin{bmatrix} e(n-1) \\ e(n) \end{bmatrix}$$

Maintenant, on réduit la complexité arithmétique en écrivant (3.10) comme suit:

(3.11)

$$\text{a)}\quad \begin{bmatrix} e(n-1) \\ e(n) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -\mu s(n) & 1 \end{bmatrix} \left[\begin{bmatrix} y(n-1) \\ y(n) \end{bmatrix} - \begin{bmatrix} A_1^t(H_{n-1}^0 + H_{n-1}^1) + (A_2 - A_1)^t H_{n-1}^1 \\ A_1^t(H_{n-1}^0 + H_{n-1}^1) - (A_1 - A_0)^t H_{n-1}^0 \end{bmatrix} \right]$$

$$\text{b)}\quad \begin{bmatrix} H_{n+1}^0 \\ H_{n+1}^1 \end{bmatrix} = \begin{bmatrix} H_{n-1}^0 \\ H_{n-1}^1 \end{bmatrix} + \mu \begin{bmatrix} A_1(e(n-1) + e(n)) - (A_1 - A_0)e(n) \\ A_1(e(n-1) + e(n)) + (A_2 - A_1)e(n-1) \end{bmatrix}$$

L'opération de filtrage $A_1{}^t(H^0{}_{n-1}+H^1{}_{n-1})$ est commune aux deux termes de (3.11a); on a ainsi 3 filtres de longueur L/2 au lieu de 4 comme c'est le cas dans (3.10a). Deux d'entre eux dépendent des expressions suivantes:

(3.12)

$$A_2 - A_1 = [x(n-2) - x(n-1) \ldots\ldots x(n-L) - x(n-L+1)]^t$$

$$A_1 - A_0 = [x(n-1) - x(n) \ldots\ldots x(n-L+1) - x(n-L+2)]^t$$

Le nombre apparent d'additions dans (3.12) peut être diminué en notant que l'ensemble des produits scalaires du bloc précédent $\{X_{n-2}{}^tH_{n-2}, X_{n-3}{}^tH_{n-3}\}$ demande presque le même nombre d'opérations; il y a seulement deux additions en plus à calculer:(x(n-2)-x(n-1)) et (x(n-1)-x(n)).

On remarquera aussi que dans l'équation d'adaptation du filtre (3.11b), on aura 3 produits d'un vecteur de longueur L/2 par un scalaire à réaliser, au lieu de 4.

Le produit scalaire s(n) de longueur L se calcule récursivement comme suit:

(3.13)

$$s(n) = s(n-2) + \{x(n-1)(x(n) + x(n-2)) - x(n-L-1)(x(n-L) + x(n-L-2))\}$$

La seconde expression de l'accolade a été calculée à l'instant n-L, ainsi (3.13) nécessite une multiplication et trois additions.

Nous allons, à présent, comparer la complexité arithmétique des deux algorithmes. Le LMS nécessite pour deux sorties successives:

4L muliplications

4L additions

l'algorithme proposé (éq.(3.11)) demande:

3L+2 multiplications

4L+8 additions

pour le même calcul. Cela signifie que le nombre de multiplications a été réduit d'à peu près 25% avec une légère augmentation du nombre d'additions et ceci sans aucune approximation sur le LMS.

3.2.2. Discussion

L'exemple du paragaphe 3.2.1 nous permet de comprendre le principe de fonctionnement de l'algorithme FELMS où la réduction de la charge de calcul a été possible pour trois raisons. D'abord, on profite de la redondance qui apparait clairement dans la matrice pour l'opération de filtrage, ensuite la matrice qui intervient dans la mise à jour de l'erreur est triangulaire inférieure donc facilement inversible et enfin la possibilité de calculer récursivement le scalaire s(n).

Le cas N=2 peut se généraliser pour une taille de bloc quelconque. Plus cette taille sera grande et plus le nombre d'opérations (aussi bien les multiplications que les additions) diminuera.

Cependant, une analyse détaillée de la complexité arithmétique fournie en annexe (article I), montre qu'elle fait intervenir deux termes: l'un décroit avec N (filtrage et mise à jour de H), l'autre augmente avec N (inversion de S et calcul récursif de ses éléments). Il existe donc une taille de bloc optimale (N_{opt}) pour un filtre de longueur donnée et on a toujours $N_{opt}<L$. On se rend compte que l'ensemble des techniques expliquées dans ce chapitre sont liées à l'existence d'algorithmes rapides et efficaces avec N<L.

L'un des points importants de notre algorithme est donc de pouvoir gérer le retard à la sortie du filtre par le choix de N.

Le cas général de l'algorithme FELMS peut se résumer à ceci:

- l'obtention d'un filtre RIF fixe pour le calcul de N sorties successives et des erreurs correspondantes;

- la correction de ces erreurs pour obtenir celles qui auraient été fournies par un algorithme LMS; et

- l'obtention d'un algorithme pour adapter ce filtre tous les N instants.

La figure 3.1 donne le principe de fonctionnement de l'algorithme FELMS.

Le tableau 3.1 donne la complexité arithmétique par point de sortie des deux algorithmes LMS et FELMS pour différentes longueurs de filtres et pour des tailles de blocs optimales. On peut voir que pour un filtre de 1024 coefficients, l'algorithme FELMS nécessite 3 fois moins d'opérations que le LMS.

		Algorithme LMS		Algorithme FELMS	
Longueur du filtre L	Taille du bloc N	Nombre d'adds	Nombre de mults	Nombre d'adds	Nombre de mults
32	8	64	64	70	37
64	8	128	128	116	64
128	16	256	256	192	103
256	32	512	512	315	167
512	32	1024	1024	542	289
1024	64	2048	2048	865	458

Tableau 3.1: Comparaison du nombre d'opérations des algorithmes LMS et FELMS.

Dans notre approche, la diminution de la charge de calcul a été obtenue par un réarrangement des équations initiales et il y a équivalence mathématique avec l'algorithme LMS. Néanmoins, plusieurs approximations sont possibles au niveau de la matrice triangulaire qui intervient dans le calcul des erreurs, car elle a la propriété caractéristique de ne dépendre que des statistiques du signal d'entrée. Ces approximations impliquent encore la réduction de la complexité arithmétique et suivant les cas, un comportement plus ou moins proche du LMS (voir l'article I en annexe).

Elles s'appliquent, bien sûr, quelle que soit la technique de calcul rapide utilisée, et particulièrement à celle expliquée au paragraphe 3.2.

L'ensemble des techniques de réduction de calcul exposées précédemment s'appliquent à des variantes du LMS comme l'algorithme du signe ou encore l'algorithme du gradient normalisé (voir annexe).

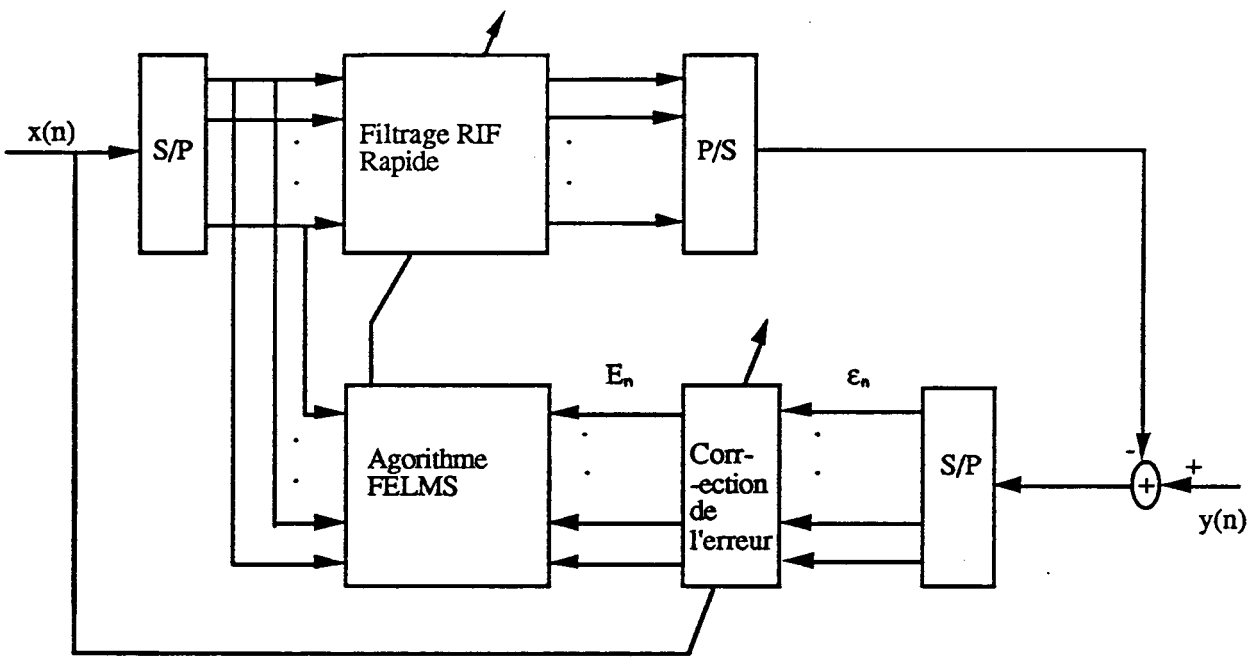


Figure 3.1: Principe de l'algorithme FELMS. S/P=convertisseur série/parallèle P/S=convertisseur parallèle/série

3.3. L'ALGORITHME FELMS UTILISANT DES FFT DE PETITES LONGUEURS (FD-FELMS)

Cette section est consacrée à l'utilisation de FFT de petites longueurs dans l'algorithme FELMS. L'objectif étant de réduire davantage la complexité arithmétique de l'algorithme par rapport à l'algorithme précédent, quand on a affaire à des filtres très longs.

Bien qu'il soit décrit de manière très succincte dans l'article I de l'annexe, nous donnons ici les détails mathématiques nécessaires pour l'obtention du FD-FELMS.

3.3.1. Une autre manière d'écrire le FELMS

Dans l'expression (2.43) du chapitre II, le terme $\underline{X}(n)$ est une matrice de Toeplitz

rectangulaire (NxL). Cette matrice est carrée et de Toeplitz si L=N. Le but de ce paragraphe est de faire apparaitre dans l'algorithme, des matrices de Toeplitz carrées (NxN) plus petites que dans le cas L=N. On pourra alors introduire la FFT.

Supposons que L=MN où M est un entier positif, alors l'équation (2.43) peut s'écrire comme une somme de L/N produits d'une matrice de Toeplitz carrée (NxN) par un vecteur (Nx1):

(3.14)

$$Y'_n = \sum_{i=0}^{(L/N)-1} T_i(n)\tilde{H}^i_{n-N+1}$$

avec:

$$\tilde{H}^i_{n-N+1} = \left[h_{Ni}(n-N+1)\ \ h_{Ni+1}(n-N+1)\ldots\ldots h_{Ni+N-2}(n-N+1)\ \ h_{Ni+N-1}(n-N+1)\right]^t$$
$$i = 0,1,\ldots\ldots,(L/N)-1$$

et:

$$T_i(n) = \begin{bmatrix} x(n-Ni-(N-1)) & x(n-Ni-N) & \ldots\ldots\ldots\ldots & x(n-Ni-(2N-2)) \\ x(n-Ni-(N-2)) & x(n-Ni-(N-1))\ldots\ldots\ldots & & \vdots \\ \vdots & & & \\ \vdots & & & \\ x(n-Ni-1)\ldots & & & \vdots \\ x(n-Ni) & \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots & & x(n-Ni-(N-1)) \end{bmatrix}$$

Le vecteur H est ainsi "coupé" en L/N sous-filtres de longueur N et $T_i(n)$ est une matrice de Toeplitz carrée (NxN).

Maintenant, l'adaptation des coefficients du filtre se fait selon les équations suivantes:

(3.15)

$$\tilde{H}^i_{n+1} = \tilde{H}^i_{n-N+1} + \mu T_i^t(n)E_n$$
$$i = 0,1,\ldots\ldots,(L/N)-1$$

Puisque nous avons introduit des matrices de Toeplitz carrées (NxN) dans les expressions (3.14) et (3.15), il est facile de faire intervenir la FFT, suivant les techniques de traitements classiques.

3.3.2. Utilisation de la FFT

Soit F une matrice symétrique (2Nx2N) dont les éléments sont: $f_{kj}=\exp(-i(2\pi/2N)kj)$, k,j=0,1,....,2N-1. Le produit de F par un vecteur (2Nx1) est un vecteur représentant la DFT du vecteur initial.

Soit F^{-1} l'inverse de F, on peut montrer que $F^{-1}=(1/2N)F^*$ où F* désigne le conjugué de F.

Nous savons que si C est une matrice circulante: $D=FCF^{-1}$ où D est une matrice diagonale dont les éléments sont la DFT de la première colonne de C.

Une matrice de Toeplitz peut être rendue circulante en doublant sa taille. En appliquant cette transformation aux équations (3.14) et (3.15), nous obtenons:

(3.16)

$$\begin{bmatrix} Y''_n \\ Y'_n \end{bmatrix} = \sum_{i=0}^{(L/N)-1} \begin{bmatrix} T'_i(n) & T_i(n) \\ T_i(n) & T'_i(n) \end{bmatrix} \begin{bmatrix} \tilde{H}^i_{n-N+1} \\ \underline{O} \end{bmatrix}$$

$$= \sum_{i=0}^{(L/N)-1} C_i(n) \begin{bmatrix} \tilde{H}^i_{n-N+1} \\ \underline{O} \end{bmatrix}$$

et:

(3.17)

$$\begin{bmatrix} \tilde{H}^i_{n+1} \\ \underline{O} \end{bmatrix} = \begin{bmatrix} \tilde{H}^i_{n-N+1} \\ \underline{O} \end{bmatrix} + \mu W_F C^t_i(n) \begin{bmatrix} \underline{O} \\ E_n \end{bmatrix}$$

$i = 0,1,.....,(L/N)-1$

où $T'_i(n)$ est une matrice de Toeplitz (NxN):

$$T'_i(n) = \begin{bmatrix} x(n-Ni-(2N-1)) & x(n-Ni) & \cdots & x(n-Ni-(N-2)) \\ x(n-Ni-(2N-2)) & x(n-Ni-(2N-1)) & \cdots & \vdots \\ \vdots & & & \\ \vdots & & & \\ \vdots & & & \vdots \\ x(n-Ni-N) & \cdots & & x(n-Ni-(2N-1)) \end{bmatrix}$$

$C_i(n)$ est une matrice circulante de dimension (2Nx2N), $\underline{O}$ un vecteur nul (Nx1), Y''_n un vecteur (Nx1) non explicité car non utilisé et W_F une matrice diagonale:

$W_F=\text{diag}\{1,1,....,1,0,0,....,0\}$

La matrice W_F est une fenêtre indispensable pour maintenir l'équivalence (avec l'algorithme

initial -FELMS).

L'augmentation de la dimension des matrices et vecteurs des équations précédentes n'est qu'un artifice qui permet d'utiliser les propriétés des matrices circulantes, et ne change absolument rien au résultat.

En introduisant la DFT dans les équations (3.16) et (3.17), nous obtenons:

(3.18)

$$\begin{bmatrix} Y''_n \\ Y'_n \end{bmatrix} = F^{-1} \sum_{i=0}^{(L/N)-1} \left[FC_i(n)F^{-1} \right] F \begin{bmatrix} \tilde{H}^i_{n-N+1} \\ \underline{O} \end{bmatrix}$$

$$= F^{-1} \sum_{i=0}^{(L/N)-1} D_i(n) F \begin{bmatrix} \tilde{H}^i_{n-N+1} \\ \underline{O} \end{bmatrix}$$

et:

(3.19)

$$F \begin{bmatrix} \tilde{H}^i_{n+1} \\ \underline{O} \end{bmatrix} = F \begin{bmatrix} \tilde{H}^i_{n-N+1} \\ \underline{O} \end{bmatrix} + \mu F W_F F^{-1} D_i^*(n) F \begin{bmatrix} \underline{O} \\ E_n \end{bmatrix}$$

$$i = 0,1,.....,(L/N)-1$$

où $D_i(n)$ est une matrice diagonale dont les éléments sont la sortie de la DFT de la première colonne de $C_i(n)$.

Or, certains éléments sont communs entre les matrices T_i successives, on peut alors obtenir une relation de décalage temporel:

(3.20)

$$D_i(n) = D_{i-1}(n-N)$$
$$i = 1,2,....,(L/N)-1$$

et l'expression (3.18) peut encore s'écrire:

(3.21)

$$\begin{bmatrix} Y''_n \\ Y'_n \end{bmatrix} = F^{-1} \left(D_0(n) F \begin{bmatrix} \tilde{H}^0_{n-N+1} \\ \underline{O} \end{bmatrix} + \sum_{i=1}^{(L/N)-1} D_{i-1}(n-N) F \begin{bmatrix} \tilde{H}^i_{n-N+1} \\ \underline{O} \end{bmatrix} \right)$$

$$= F^{-1} \left(\sum_{i=0}^{(L/N)-1} D_0(n-Ni) F \begin{bmatrix} \tilde{H}^i_{n-N+1} \\ \underline{O} \end{bmatrix} \right)$$

qui montre bien que l'ensemble des opérations à effectuer dans le domaine fréquentiel est

constitué de 2N filtres de longueur L/N. On a alors le schéma de filtrage rapide suivant [20,23]:

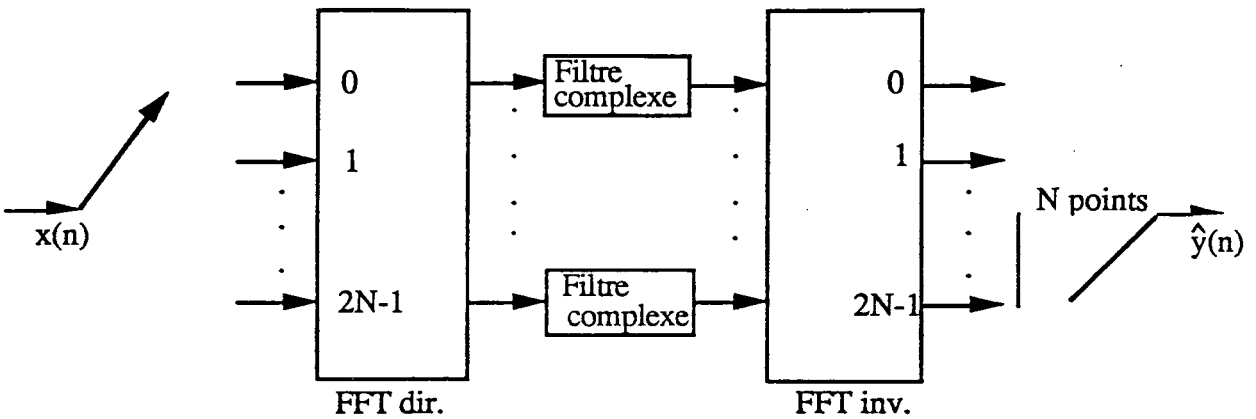


Figure 3.2: Filtrage rapide fréquentiel

Les expressions (3.18) et (3.19) forment l'algorithme FD-FELMS qui est mathématiquement équivalent au LMS. Nous avons, ainsi, complétement dissocié la taille de la matrice F de la longueur du filtre, parce que nous avons réussi à trouver la formulation adéquate. On peut montrer que cet algorithme (pour la partie filtrage fixe) est équivalent à d'autres méthodes de convolution rapide, par exemple la double convolution [29].

3.3.3. Complexité arithmétique de l'algorithme FD-FELMS pour $N=2^n$

Dans tout ce chapitre, nous utilisons l'algorithme "Split-Radix" [34] pour calculer la transformée de Fourier d'un vecteur. Pour un vecteur (2Nx1) de données réelles, le coût de la FFT est de:

$2^n(n-2)+2$ multiplications réelles

$2^n(3n-2)+4$ additions réelles

L'expression (3.18) nécessite le calcul d'une FFT directe:

$D_0(n)=\text{diag}\{\text{FFT}\{x(n-(2N-1)),...,x(n-N),x(n-(N-1)),...,x(n)\}\}$,

d'une FFT inverse d'un vecteur hermitien, du calcul de 2 filtres RIF réels de longueur L/N et de (N-1) filtres RIF complexes de longueur L/N.

Le calcul des éléments de la matrice S nécessite:

N(N-2) multiplications

(3/2)N(N-1) additions

La complexité arithmétique pour déterminer le vecteur erreur (E_n) est de:

N(N-1)/2 multiplications

N(N+1)/2 additions

Enfin, la dernière étape est l'adaptation des coefficients du filtre. Ainsi, l'expression (3.18) demande:

- (L/N)+1 FFT directes de vecteurs réels (2Nx1),
- L/N FFT inverses de vecteurs hermitiens (2Nx1),
- 2L/N multiplications réelles,
- (N-1)L/N multiplications complexes,
- 2L/N additions réelles,
- (N-1)L/N additions complexes.

On en déduit, à présent, le nombre total d'opérations réelles par point de sortie de l'algorithme FD-FELMS:

(3.22)

$2(L/N)(n+2)+(3/2)N+3n-17/2+6/N$ multiplications

(3.23)

$2(L/N)(3n+2)+2N+9n-9+4L/N^2+12/N$ additions

On a utilisé le schéma classique de la multiplication complexe qui nécessite 4 multiplications et 2 additions réelles.

Comme pour le FELMS, le FD-FELMS a, pour une longueur de filtre donnée, une taille de bloc optimale (N_{opt}) pour laquelle le coût en charge de calcul de l'algorithme est minimal. Les figures 3.3a et 3.3b donnent les courbes du nombre d'additions et de multiplications en fonction de $\log_2 N=n$, pour un filtre de longueur L=1024. On peut voir sur cet exemple, que le bloc optimal est: $N_{opt}=2^7$.

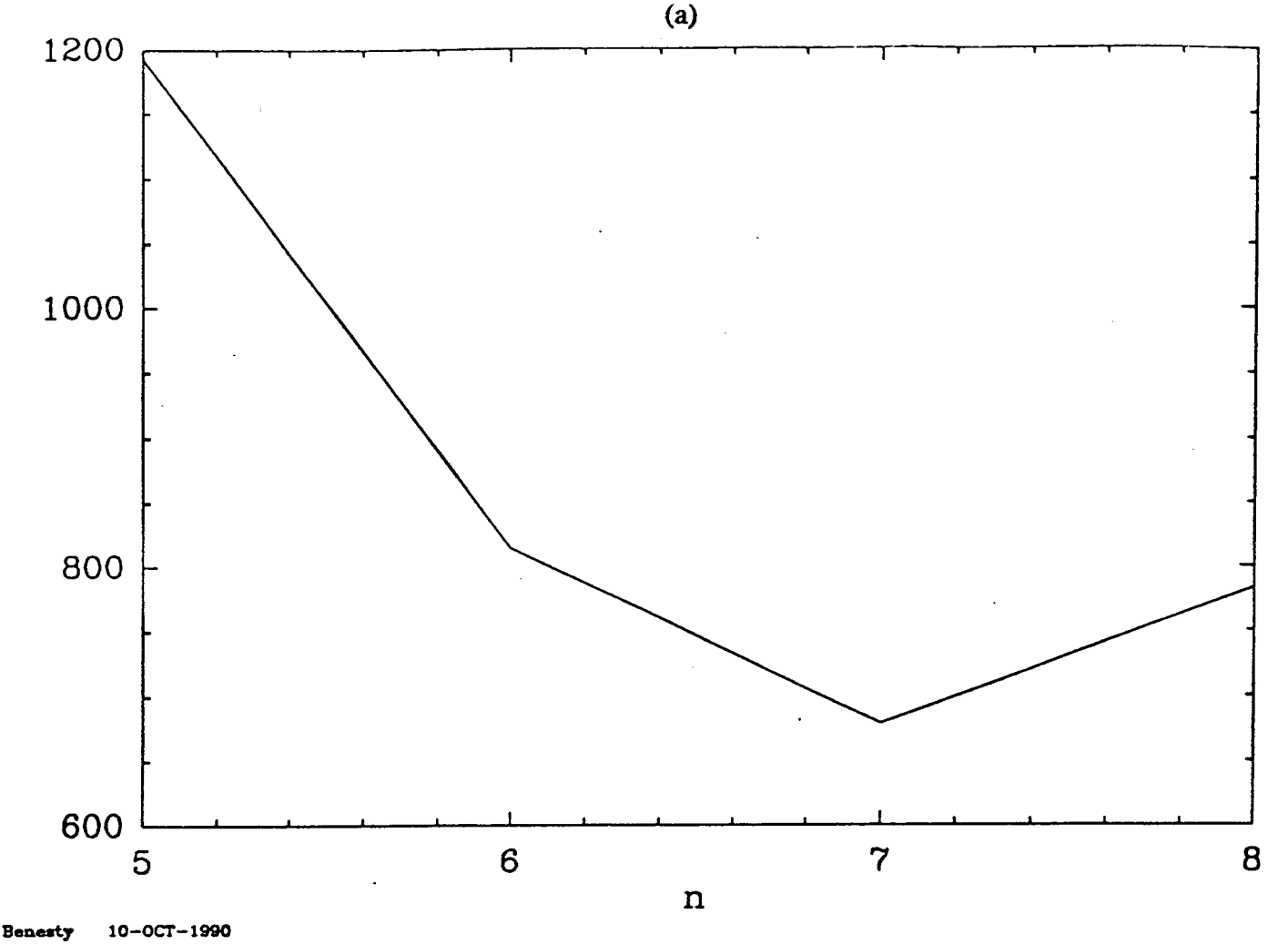
Le tableau 3.2 fournit le nombre d'opérations par point de sortie des algorithmes LMS et FD-FELMS, pour différentes longueurs de filtres. Nous remarquons que pour un filtre L=1024 et un bloc N=128, le nombre total d'opérations a été réduit d'un facteur 4 par rapport au LMS.

3.4. CONCLUSION

Dans ce chapitre, nous avons proposé deux versions rapides de l'algorithme LMS. Le premier obtenu (FELMS) est une version temporelle du LMS où la complexité arithmétique est diminuée quelle que soit la taille du bloc. Nous avons vu sur un exemple simple (N=2) que cela était possible. Cet exemple est généralisé pour une taille de bloc quelconque en annexe (voir article I). Le FELMS semble être intéressant quand on travaille avec des filtres pas trop longs et des blocs pas très grands. La seconde version (fréquentielle) est utile lorsqu'on a de longues réponses impulsionnelles à traiter puisqu'elle réduit davantage la charge de calcul. Il est important de retenir que les algorithmes LMS, FELMS et FD-FELMS sont mathématiquement équivalents et que la taille de bloc n'est plus liée à la longueur du filtre. Cette formulation par bloc sera utilisée dans le chapitre suivant pour "travailler" le comportement adaptatif du filtre.

		LMS		FD-FELMS	
L	N	Nbre d'adds	Nbre de mults	Nbre d'adds	Nbre de mults
128	32	256	256	237	111
256	64	512	512	333	170
512	64	1024	1024	494	234
1024	128	2048	2048	678	349

Tableau 3.2: Comparaison du nombre d'opérations du LMS et du FD-FELMS



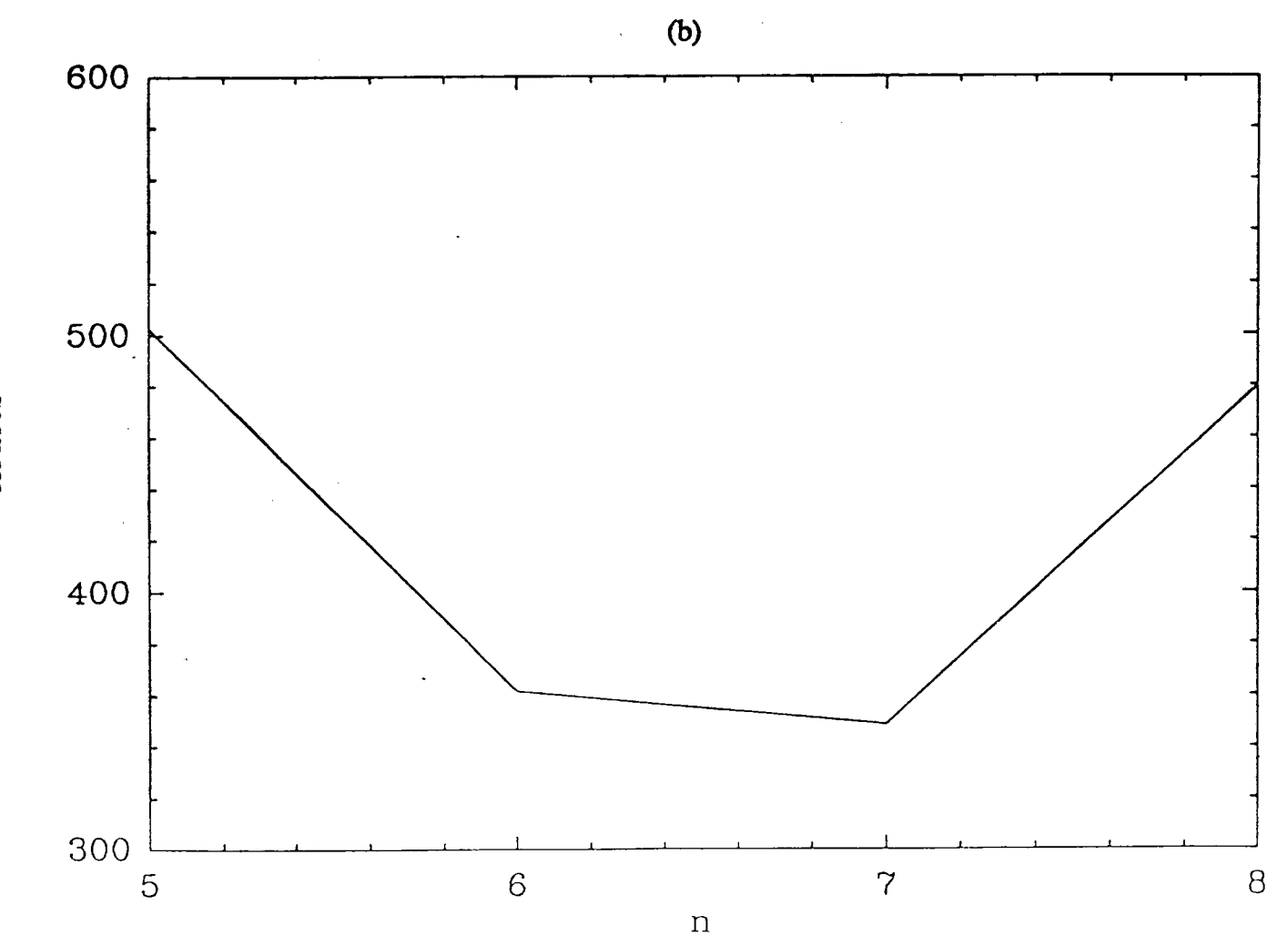


Fig. 3.3: Courbes donnant le nombre d'additions (a) et de multiplications (b) en fonction de $\log_2 N$, pour un filtre L=1024

CHAPITRE IV

AUTRES ALGORITHMES PLUS RAPIDES QUE LE LMS

(au sens de la vitesse de convergence)

4.1. INTRODUCTION

Au chapitre précédent, nous avons vu comment diminuer la charge de calcul de l'algorithme LMS, aussi bien dans le domaine temporel que fréquentiel. Nous désirons, à présent, profiter de la formulation proposée, pour dériver des algorithmes plus performants, mais cette fois-ci au sens du comportement adaptatif (vitesse de convergence et capacité de poursuite), que le LMS, tout en recherchant la réduction de la complexité arithmétique. L'objectif étant de trouver un algorithme qui converge plus vite, qui poursuit mieux et qui nécessite moins d'opérations que le LMS.

Nous proposons, en fait, trois nouveaux algorithmes. Le premier d'entre eux se sert directement des propriétés de la transformée de Fourier pour améliorer la vitesse de convergence. Il suffit, pour cela, de remplacer le pas de convergence du FD-FELMS par une matrice diagonale. De plus, en utilisant cette technique, il apparait dans les simulations que la matrice S n'a plus aucun effet; on pourra donc la négliger, et l'algorithme obtenu est le MDF ("Multidelay Block Frequency Domain") que nous avons obtenu indépendamment et en même temps que d'autres auteurs [27].

Le MDF converge mieux et nécessite moins d'opérations que le FD-FELMS, mais il poursuit moins bien les variations du système du fait que les coefficients du filtre restent fixes pendant tout un bloc.

Dans la section 4.3, nous dérivons un algorithme qui a une meilleure capacité de poursuite que les précédents. Pour ce faire, nous proposons un nouveau critère qui découle directement de l'erreur du FELMS. L'algorithme obtenu (NB-FELMS) converge plus vite et poursuit mieux que le LMS.

Comme cet algorithme est dérivé à partir de la formulation par bloc du chap. 2 qui a permis une réduction de calcul au chap. 3, toutes les techniques de réduction de complexité vont

s'appliquer, que ce soient les techniques temporelles ou fréquentielles. Il est toujours possible d'obtenir un algorithme à charge de calcul inférieure à celle du LMS, bien que présentant un comportement adaptatif amélioré.

Une vision plus globale de l'algorithme FELMS montre une analogie avec l'algorithme RLS (inversion d'une matrice de corrélation). Ceci nous a incité à rechercher s'il était possible de trouver des algorithmes qui font le lien entre le FELMS et le RLS.

Au paragraphe 4.4, nous établissons un lien entre le FELMS modifié, de manière justifiée au niveau de l'inverse de la matrice de corrélation, et le "Block-RLS". L'algorithme résultant, introduit l'inversion de la matrice de corrélation du signal d'entrée dont la dimension est indépendante de la longueur du filtre. Ce point est attirant, car une matrice de corrélation liée à la longueur du filtre ne semble pas nécessaire pour avoir le même comportement adaptatif: dans beaucoup de cas, en effet, et spécialement pour des réponses impulsionnelles longues, il est illusoire de vouloir attribuer une signification physique aux coefficients de corrélation d'ordre élevé. Dériver des algorithmes où les tailles de corrélation sont plus faibles est donc un objectif souhaitable.

Enfin, dans la section 4.5, nous donnons des résultats de simulations pour comparer les différents algorithmes proposés.

4.2. L'ALGORITHME MDF

Reprenons l'algorithme FD-FELMS. Pour accélérer sa vitesse de convergence, nous utilisons le fait que l'énergie du signal d'entrée dans le domaine temporel se retrouve dans le domaine fréquentiel (voir chapitre II). Nous ne faisons là qu'étendre les résultats du TDLMS où, comme nous l'avons vu, l'adaptation se fait dans le domaine de la transformation pour réduire le rapport $(\lambda_{max}/\lambda_{min})$ de la matrice de corrélation, et donc augmenter la vitesse de convergence. Ainsi, on peut remplacer le scalaire μ (qui ne dépend que de l'énergie du signal d'entrée) par une matrice diagonale:

(4.1)

$$\Lambda^{-1}(n) = \alpha\left\{\frac{L}{N}\text{diag}(P_0(n), P_1(n), \ldots\ldots\ldots, P_{2N-1}(n))\right\}^{-1}$$

où P_k est une estimation de la puissance du signal à la fréquence k, obtenue, par exemple, à l'aide de la formule suivante:

$$P_k(n) = \beta P_k(n-N) + (1-\beta)\left(\sum_{i=0}^{(L/N)-1} d_{k,i}(n) d_{k,i}^*(n)\right)\frac{N}{L}$$

$$k = 0,1,....,2N-1$$

Si:

$$z_k(n) = \sum_{i=0}^{(L/N)-1} d_{k,i}(n) d_{k,i}^*(n)$$

et puisque:

$$d_{k,i}(n) = d_{k,i-1}(n-N)$$

$$i = 1,2,....,(L/N)-1$$

alors:

$$z_k(n) = z_k(n-N) + d_{k,0}(n) d_{k,0}^*(n) - d_{k,(L/N)-1}(n-N) d_{k,(L/N)-1}^*(n-N)$$

α est le facteur de normalisation pour ajuster la vitesse de convergence et β le facteur d'oubli.

D'autre part, des simulations prouvent que la partie inférieure de la matrice de corrélation qui intervient dans le calcul de l'erreur (où le pas μ demeure constant) n'a plus aucun effet sur l'algorithme si l'on utilise la technique précédente, on pourra donc la négliger et l'algorithme obtenu sera encore beaucoup plus simple.

Cet algorithme est le MDF avec la simplification précédente et sa charge de calcul par point de sortie est:

(4.2)

$2(L/N)(n+3)+3n+2+6/N$ multiplications

(4.3)

$2(L/N)(3n+2)+9n-1+4L/N^2+12/N$ additions

Un algorithme similaire a été proposé dans [27] et d'autres algorithmes fondés sur les mêmes principes peuvent être trouvés dans [28,29,30].

Le tableau 4.1 donne le nombre d'opérations de l'algorithme MDF pour plusieurs longueurs

de filtres. On voit que la charge de calcul de celui-ci est bien inférieure à celle du FD-FELMS, puisqu'on n'a plus à calculer les éléments de la matrice S dont le coût est élevé.

L'algorithme MDF nécessite beaucoup moins d'opérations que le LMS (pour un filtre de 1024 coefficients et un bloc de 128, le gain est d'un facteur 7) et converge plus vite. Ceci a été possible, parce que nous avons profité des avantages de la FFT (accélération des calculs et décorrélation partielle). Cependant, le MDF poursuit moins bien, du fait que le filtre n'est adapté qu'une seule fois par bloc.

Nous pouvons nous poser la question suivante: existe-t-il un algorithme par bloc qui puisse mieux poursuivre qu'un algorithme séquentiel?

4.3. UN NOUVEL ALGORITHME PAR BLOC AYANT DE BONNES PERFORMANCES EN POURSUITE (NB-FELMS)

Il est tout à fait possible de répondre affirmativement à la question précédente. La raison est que le FELMS (qui est un algorithme par bloc) poursuit aussi bien que le LMS (qui est un algorithme séquentiel), puisqu'ils sont mathématiquement équivalents. D'où l'idée d'utiliser le vecteur erreur obtenu dans l'algorithme FELMS dans un critère par bloc.

4.3.1. Critère

Le vecteur erreur (E_n) tient compte de l'information utilisée pour l'adaptation du LMS à chacun des échantillons, contrairement à l'erreur par bloc classique (ε_n) [18]. Il serait donc intéressant de déduire une nouvelle fonction coût à partir de E_n, puisque dans celle-ci, même si H apparait fixe pendant tout le bloc, il est en réalité fictivement mis à jour grâce à la matrice S.

Soit le critère suivant:

(4.4)

$$J_{G'} = \frac{1}{N} E\left\{ E'^{t}_{n} E'_{n} \right\} > 0$$

où (par analogie avec 2.42):

(4.5)

$$E'_n = G'(n)\varepsilon_n = G'(n)\left[Y_n - \underline{X}(n)H\right]$$

avec:

$$G'(n) = \left[\mu_1 S(n) + I\right]^{-1} \quad \text{et} \quad \mu_1 = \frac{1}{LE\left\{x^2(n)\right\}}$$

En développant (4.4), on a:

(4.6)

$$J_{G'} = \frac{1}{N}\left\{E\left\{\left[G'(n)Y_n\right]^t\left[G'(n)Y_n\right]\right\} - 2E\left\{\left[G'(n)Y_n\right]^t\left[G'(n)\underline{X}(n)\right]\right\}H + H^tE\left\{\left[G'(n)\underline{X}(n)\right]^t\left[G'(n)\underline{X}(n)\right]\right\}H\right\}$$

Posons:

$$R_{G'} = E\left\{\left[G'(n)\underline{X}(n)\right]^t\left[G'(n)\underline{X}(n)\right]\right\}$$

$$r_{G'} = E\left\{\left[G'(n)\underline{X}(n)\right]^t\left[G'(n)Y_n\right]\right\}$$

En utilisant les définitions précédentes, $J_{G'}$ peut s'écrire de manière plus compacte:

$$J_{G'} = \frac{1}{N}\left\{E\left\{\left[G'(n)Y_n\right]^t\left[G'(n)Y_n\right]\right\} - 2r_{G'}^t H + H^t R_{G'} H\right\}$$

Cas particulier: Si l'on prend μ_1=0, alors G'(n)=I et:

$$J_{G'} = J_B = \frac{1}{N}E\left\{\varepsilon_n^t \varepsilon_n\right\}$$

qui est le critère par bloc classique [18].

Le critère que nous avons proposé tient compte des variations de l'erreur à l'intérieur d'un bloc. Ainsi, le dernier élément du vecteur E'_n dépend de toutes les erreurs précédentes de ce même bloc. Ceci aura un effet positif dans un algorithme adaptatif par bloc.

Il est maintenant possible de dériver un algorithme pour l'adaptation du filtre H.

4.3.2. Algorithme

Le vecteur gradient est:

(4.7)

$$\nabla = \frac{1}{N}\frac{\partial E\left\{E'^{t}_{n}E'_{n}\right\}}{\partial H} = \frac{2}{N}\left[-r_{G'} + R_{G'}H\right]$$

En égalant l'expression (4.7) à zéro, on obtient le filtre optimal:

(4.8)

$$H_{opt} = R_{G'}^{-1}r_{G'}$$

On suppose, bien sûr, que $R_{G'}$ est non-singulière.

Le minimum de la fonction coût (4.4) est obtenu à partir de (4.6) et de (4.8):

$$(J_{G'})_{min} = \frac{1}{N}\left\{E\left\{\left[G'(n)Y_n\right]^t\left[G'(n)Y_n\right]\right\} - r_{G'}^{t}H_{opt}\right\}$$

D'où:

(4.9)

$$J_{G'} = (J_{G'})_{min} + \frac{1}{N}(H - H_{opt})^t R_{G'}(H - H_{opt})$$

qui est une autre façon d'écrire l'équation (4.6).

Comme pour le filtrage adaptatif LMS, un algorithme peut être déduit par la technique du gradient, mais le vecteur poids n'est adapté qu'une seule fois par bloc. L'algorithme est donc:

(4.10)

$$H_{n+1} = H_{n-N+1} - \frac{\mu_{1B}}{2}\nabla$$

où μ_{1B} est la constante de convergence.

Parce que le calcul d'une moyenne d'ensemble est difficile, on ne prendra qu'une estimation du vecteur gradient sachant que cette moyenne se fera au cours du temps (du type gradient stochastique):

$$\hat{\nabla} = \frac{1}{N}\frac{\partial(E'^{t}_{n}E'_{n})}{\partial H} = -\frac{2}{N}\left[G'(n)\underline{X}(n)\right]^t E'_n = -\frac{2}{N}\underline{X}^t(n)G'^{t}(n)G'(n)\varepsilon_n$$

A présent, les équations suivantes résument l'algorithme NB-FELMS:

(4.11)

a) $G'(n) = [\mu_1 S(n) + I]^{-1}$

b) $E'_n = G'(n)[Y_n - \underline{X}(n)H_{n-N+1}]$

c) $H_{n+1} = H_{n-N+1} + \frac{\mu_{1B}}{N}\underline{X}^t(n)G'^t(n)E'_n$

Des simulations montrent que μ_{1B} est choisi comme suit:

$$\mu_{1B} \leq 3N\mu_1 = \frac{3N}{LE\{x^2(n)\}}$$

Convergence de l'algorithme:

En prenant l'espérance mathématique de l'expression (4.11c) et en supposant que les entrées sont stationnaires, on a:

$$E\{H_{n+1}\} = E\{H_{n-N+1}\} + \frac{\mu_{1B}}{N}[r_{G'} - R_{G'}E\{H_{n-N+1}\}]$$

où l'hypothèse classique que $[G'(n)\underline{X}(n)]$ et H_{n-N+1} sont décorrélés, a été faite. Si μ_{1B} est suffisamment petit, alors:

$$\lim_{n \to \infty} E\{H_{n-N+1}\} = R_{G'}^{-1} r_{G'} = H_{opt}$$

Ainsi, la moyenne d'ensemble converge bien vers le filtre optimal.

4.3.3. Complexité arithmétique de l'algorithme NB-FELMS pour $N=2^n$

L'algorithme proposé plus haut peut être implanté aussi bien dans le domaine temporel que fréquentiel. Dans le domaine temporel, les techniques du chapitre III peuvent être utilisées sans aucune difficulté. Nous choisissons ici, de l'implanter dans le domaine fréquentiel et tout ce qui a été fait pour le FD-FELMS s'étend facilement.

Le nombre d'opérations par point de sortie du NB-FELMS, pour un filtre égal à $L=NM=2^nM$, est:

(4.12)

$2(L/N)(n+2)+2N+3n-9+6/N$ multiplications

(4.13)

$2(L/N)(3n+2)+(5/2)N+9n-19/2+4L/N^2+12/N$ additions

Le tableau 4.1 donne la complexité arithmétique de l'algorithme NB-FELMS pour plusieurs exemples. Cette charge de calcul est supérieure à celle du FD-FELMS, mais elle demeure largement inférieure à celle du LMS.

Rappelons que la matrice G'(n) qui apparait doublement dans l'équation (4.11c) est triangulaire inférieure; par conséquent, la résolution des deux systèmes linéaires (4.11b et 4.11c) est immédiate (par substitution).

Comme nous le verrons dans les simulations, le NB-FELMS converge aussi vite que le MDF et poursuit mieux du fait que son vecteur erreur est de la même forme que celui du FELMS. On remarquera également l'allure très "lissée" des courbes d'erreur du NB-FELMS.

Ainsi, nous avons présenté un algorithme qui, à la fois, converge plus vite, poursuit mieux et nécessite moins d'opérations que le LMS.

4.4. UN AUTRE ALGORITHME PAR BLOC OU LA DIMENSION DE LA MATRICE DE CORRELATION EST INDEPENDANTE DE LA LONGUEUR DU FILTRE (SB-RLS)

Dans l'algorithme RLS, intervient directement l'inverse de la matrice d'autocorrélation du signal d'entrée dont la taille est liée à la longueur du filtre. Souvent dans la pratique, seuls un certain nombre d'éléments de corrélation de cette matrice sont significatifs tandis que les autres peuvent être approximés à zéro. Pouvons-nous trouver un algorithme où l'on peut jouer sur la taille de cette matrice?

Il est possible de répondre positivement à cette question, en modifiant l'algorithme FELMS. En effet, réécrivons tout d'abord l'équation d'adaptation du filtre de cet algorithme:

$$
\begin{aligned}
H_{n+1} &= H_{n-N+1} + \mu \underline{X}^t(n) E_n \\
&= H_{n-N+1} + \mu \underline{X}^t(n) [\mu S(n) + I]^{-1} \varepsilon_n \\
&= H_{n-N+1} + \underline{X}^t(n) \left[S(n) + \frac{1}{\mu} I \right]^{-1} \varepsilon_n
\end{aligned}
$$

Le terme S(n) est une estimation de la partie triangulaire de la matrice d'autocorrélation du signal d'entrée, de dimension (NxN). L'idée est de remplacer [S(n)+(1/μ)I] par une matrice plus

complète:

$$R_N(n) = S(n) + S^t(n) + S_0(n) = \underline{X}(n)\underline{X}^t(n)$$

avec:

$$S_0(n) = \text{diag}\{s_0(n-N+1), s_0(n-N+2), \ldots\ldots, s_0(n)\}$$

et:

$$s_0(n) = X_n^t X_n$$

$R_N(n)$ est une matrice (NxN) qui est une autre façon d'estimer la matrice de corrélation, qui n'est plus liée à la longueur du filtre mais à la taille du bloc.

On en déduit l'algorithme SB-RLS:

(4.14)

a) $\varepsilon_n = Y_n - \underline{X}(n)H_{n-N+1}$

b) $H_{n+1} = H_{n-N+1} + \underline{X}^t(n)\left[\underline{X}(n)\underline{X}^t(n)\right]^{-1}\varepsilon_n$

En effet, multiplions l'expression (4.14b) par $\underline{X}(n)$, on obtient exactement:

$$Y_n = \underline{X}(n)H_{n+1}$$

ce qui veut dire que les N sorties successives du système sont exactement données par les N sorties du filtre du bloc d'après.

Du SB-RLS, on tire deux cas importants:

- Premier cas: N=1

On obtient les équations suivantes:

a) $e(n) = y(n) - X_n^t H_n$

b) $H_{n+1} = H_n + \dfrac{1}{X_n^t X_n} X_n e(n)$

qui ne sont pas autre chose que l'algorithme du gradient normalisé (NLMS).

- Second cas: N=L

Multiplions l'expression (4.14b) par $\underline{X}(n)$:

$$\underline{X}(n)H_{n+1} = \underline{X}(n)H_{n-N+1} + \underline{X}(n)\underline{X}^t(n)\left[\underline{X}(n)\underline{X}^t(n)\right]^{-1}\varepsilon_n$$

et si maintenant on remultiplie l'expression précédente par $\underline{X}^t(n)$:

$$\underline{X}^t(n)\underline{X}(n)H_{n+1} = \underline{X}^t(n)\underline{X}(n)H_{n-N+1} + \underline{X}^t(n)\varepsilon_n$$

Si N=L, la matrice $\underline{X}^t(n)\underline{X}(n)$ est en principe inversible, on a donc:

$$H_{n+1} = H_{n-L+1} + \left[\underline{X}^t(n)\underline{X}(n)\right]^{-1}\underline{X}^t(n)\varepsilon_n$$

$$= H_{n-L+1} + \left[\sum_{i=0}^{L-1} X_{n-i}X_{n-i}^t\right]^{-1}\underline{X}^t(n)\varepsilon_n$$

C'est un algorithme RLS par bloc.

On peut toujours utiliser la FFT comme intermédiaire de calcul dans l'algorithme SB-RLS. La principale difficulté de ce dernier est la résolution du système linéaire:

$$R_N^{-1}(n)\varepsilon_n = v(n)$$

Dans le cas stationnaire il est possible toutefois d'approcher $R_N(n)$ à une matrice de Toeplitz, et la résolution de ce système sera moins complexe en appliquant, par exemple, l'algorithme de Levinson (charge de calcul en $O(N^2)$ au lieu de $O(N^3)$ par bloc) [40], et dans ce cas on aura une charge de calcul comparable à celle du FELMS. Il est à remarquer que ce coût n'est pas trop pénalisant car, encore une fois, N est déconnecté de la longueur du filtre et peut être pris petit.

4.5. SIMULATIONS

Tous les algorithmes précédents ont été programmés pour l'identification d'une réponse impulsionnelle acoustique (voir figure 3 de l'article I).

Le système à identifier est décrit par une réponse impulsionnelle mesurée dans une salle réelle (durée de la réponse: 128 ms, fréquence d'échantillonnage: 16 kHz) avec un mouvement d'un objet durant l'expérience après environ 13.000 itérations (≈ 0,8 s). Ce mouvement rend le système non-stationnaire.

Le signal d'entrée est un bruit USASI (le signal USASI est un bruit corrélé ayant le même spectre que celui de la parole).

On ajoute un bruit blanc à la sortie du système et le rapport signal à bruit (S/B) est égal à 40

dB.

Tous les algorithmes ont été implantés en virgule flottante et en simple précision (32 bits).

Le système est modélisé par un filtre RIF de longueur L=1024 et la taille de la FFT a été prise égale à 2N=2x128 pour tous les algorithmes par bloc.

Pour les courbes d'erreur, l'abscisse représente le temps qui est donné en nombre d'itérations tandis que l'ordonnée représente la puissance de l'erreur (on fait une moyenne sur 128 points) normalisée par la puissance du signal d'entrée, donnée en dB.

Nous allons comparer le comportement adaptatif (vitesse de convergence et poursuite) des cinq algorithmes suivants: LMS, FD-FELMS, MDF, NB-FELMS et SB-RLS.

- Premier cas: vitesse de convergence:

Nous avons montré que les algorithmes LMS et FD-FELMS sont strictement équivalents. Les courbes d'erreur des figures 4.1a et 4.1b, correspondant aux deux algorithmes précédents, confirment ce résultat, puisqu'on peut voir qu'elles sont identiques. Cependant, dans nos simulations, le temps de calcul du FD-FELMS est 4 fois plus court que celui du LMS. D'autre part, la figure 4.1c montre que l'algorithme MDF converge nettement plus vite que les deux précédents et avec un temps de calcul inférieur. Pour atteindre la valeur de -16 dB, il faut environ 9.000 itérations pour le LMS alors qu'il ne faut que 3.000 itérations pour le MDF. En d'autres mots, le MDF converge trois fois plus vite que le LMS. Le défaut du MDF est sa convergence initiale, plus mauvaise que celle du LMS, qui est due à l'effet du bloc.

La figure 4.1d représente la courbe d'erreur du NB-FELMS. Il est clair que ce dernier converge aussi vite que le MDF, sauf pour la convergence initiale qui est meilleure, où du moins qui fait apparaitre des erreurs d'amplitudes plus faibles.

Sur la figure 4.1e, on peut voir la courbe d'erreur du SB-RLS. Cet algorithme converge plus vite que le LMS. Pour atteindre -16 dB, il lui faut environ 2.000 itérations. Il converge donc à peu près quatre fois plus vite que le LMS et plus vite donc que le MDF et que le NB-FELMS. On peut, cependant, voir l'effet des blocs sur l'amplitude de l'erreur, qui a tendance à se comporter comme celle du MDF.

- Second cas: poursuite:

On peut distinguer trois phases sur les courbes d'erreur de ce paragraphe:

- convergence initiale de l'algorithme (de 0 à 13.000 itérations),

- poursuite de l'algorithme à cause d'une non-stationnarité dans le système (de 13.000 à 70.000 itérations), et

- reconvergence de l'algorithme après la poursuite.

Dans la première phase on laisse l'algorithme converger, puis après on déplace un objet qui a pour effet de rendre le canal acoustique non-stationnaire et ainsi l'amplitude des erreurs devient plus importante. Quand cette perturbation est stoppée, l'algorithme reconverge.

Dans le cas non-stationnaire (mouvement de l'objet dans la salle), les choses ne sont pas aussi claires que dans le cas de la convergence, mais on peut voir que le LMS (donc aussi le FD-FELMS) poursuit légèrement mieux que l'algorithme MDF (voir figures 4.2a et 4.2b). Ce résultat ne doit pas paraître étonnant, car dans les algorithmes par bloc classiques, le filtre H reste fixe pendant tout ce bloc et l'on ne tient pas compte de ses variations échantillon par échantillon. Il y a une perte d'informations et il poursuit donc moins bien.

On pourra remarquer, par contre, la supériorité du NB-FELMS (figure 4.2c) sur les deux algorithmes précédents, tandis que le SB-RLS se comporte à peu près comme le LMS.

La conclusion de ces résultats de simulations est que l'algorithme MDF converge mieux que le LMS et avec un temps de calcul nettement inférieur (de l'ordre de 7), mais avec une capacité de poursuite plus mauvaise à cause de l'effet du bloc, tandis que le NB-FELMS converge aussi vite que le MDF et poursuit mieux que le LMS.

4.6. CONCLUSION

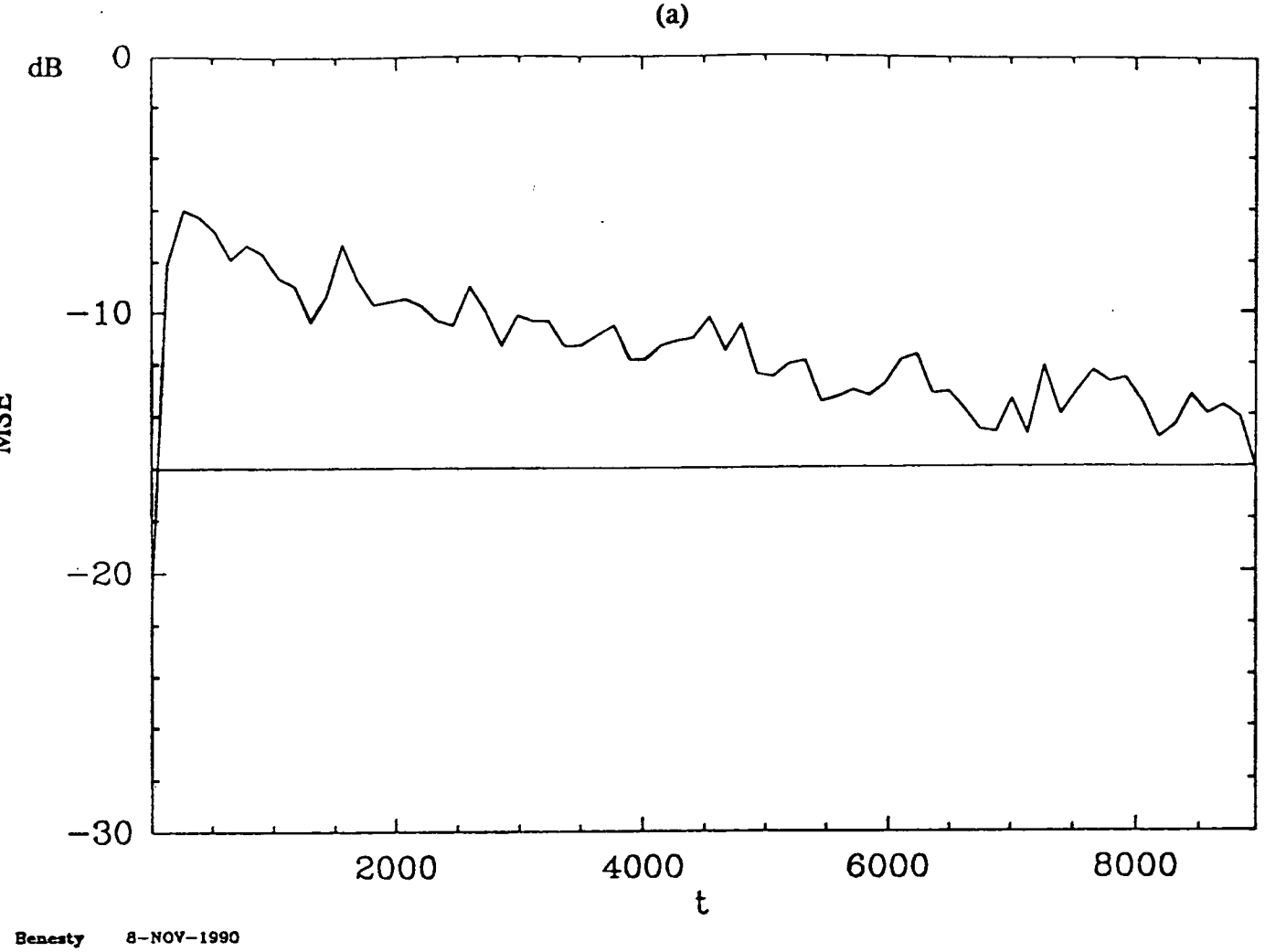
Dans ce chapitre, nous avons essayé de dériver des algorithmes plus efficaces que le LMS. Nous en avons proposé trois, qui ont tous l'avantage de converger plus vite que le LMS. Le premier algorithme (MDF) est une approximation du FD-FELMS, où l'on diminue encore le nombre d'opérations tout en accélérant sa vitesse de convergence (par rapport au LMS) grâce aux propriétés de la DFT. Toutefois, ses capacités en poursuite peuvent être inférieures à celles du LMS. Nous avons ensuite proposé un nouvel algorithme par bloc (NB-FELMS) dont la vitesse de convergence est comparable à celle du MDF, mais poursuivant nettement mieux.

Enfin, le dernier algorithme trouvé est le SB-RLS qui est un intermédiaire entre l'algorithme du gradient normalisé (cas où N=1) et le RLS par bloc (cas où N=L).

Il serait intéressant de voir que tout ce qui a été fait pour le LMS peut s'étendre à d'autres types d'algorithmes.

		LMS		FD-FELMS		MDF		NB-FELMS	
L	N	Nbre d'adds	Nbre de mults	Nbre d'adds	Nbre de mults	Nbre d'adds	Nbre de mults	Nbre d'adds	Nbre de mults
128	32	256	256	237	111	180	81	252	126
256	64	512	512	333	170	213	92	365	202
512	64	1024	1024	494	234	373	164	526	266
1024	128	2048	2048	678	349	430	183	712	413

Tableau 4.1: Comparaison du nombre d'opérations par point de sortie des différents algorithmes



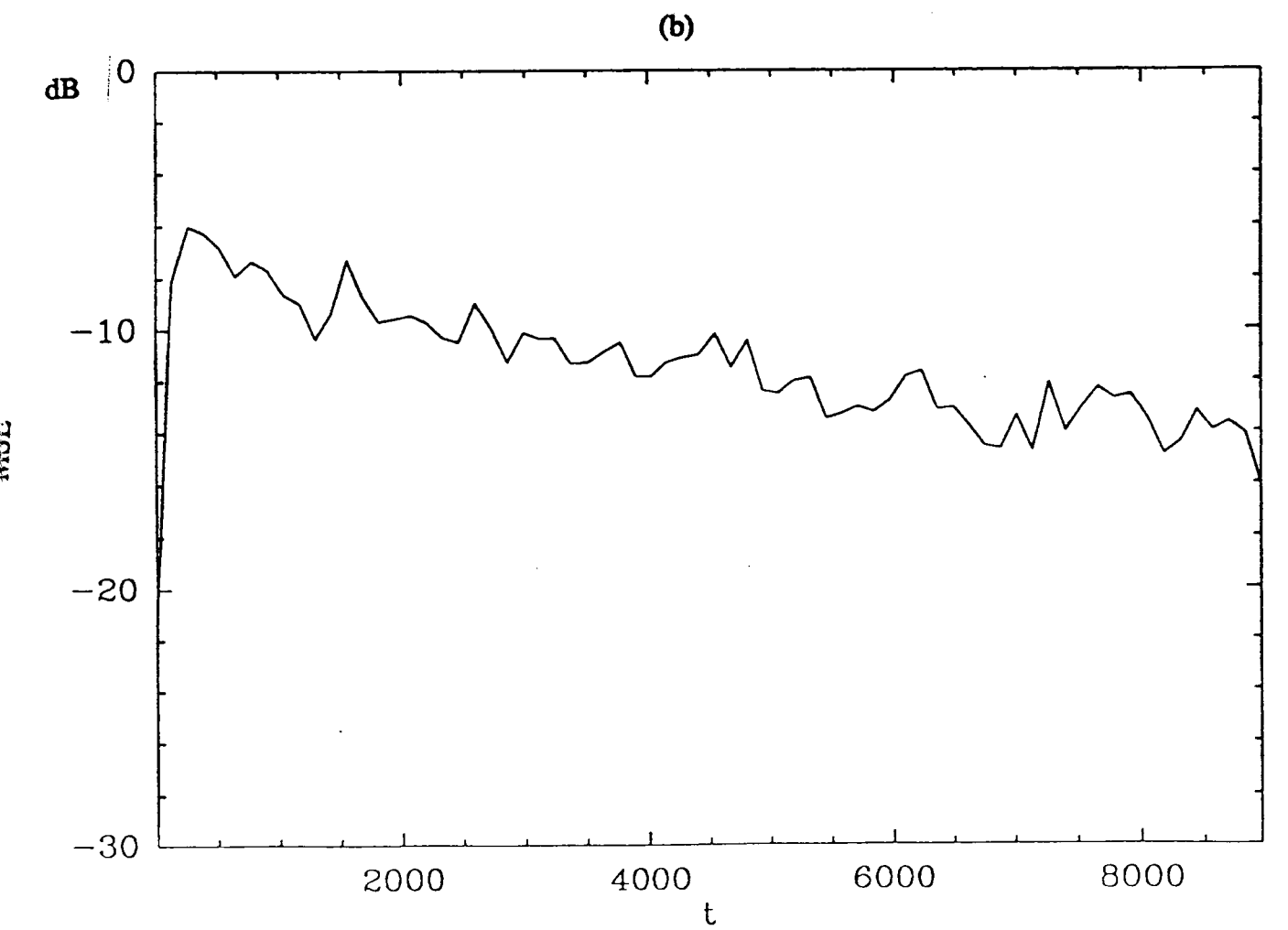
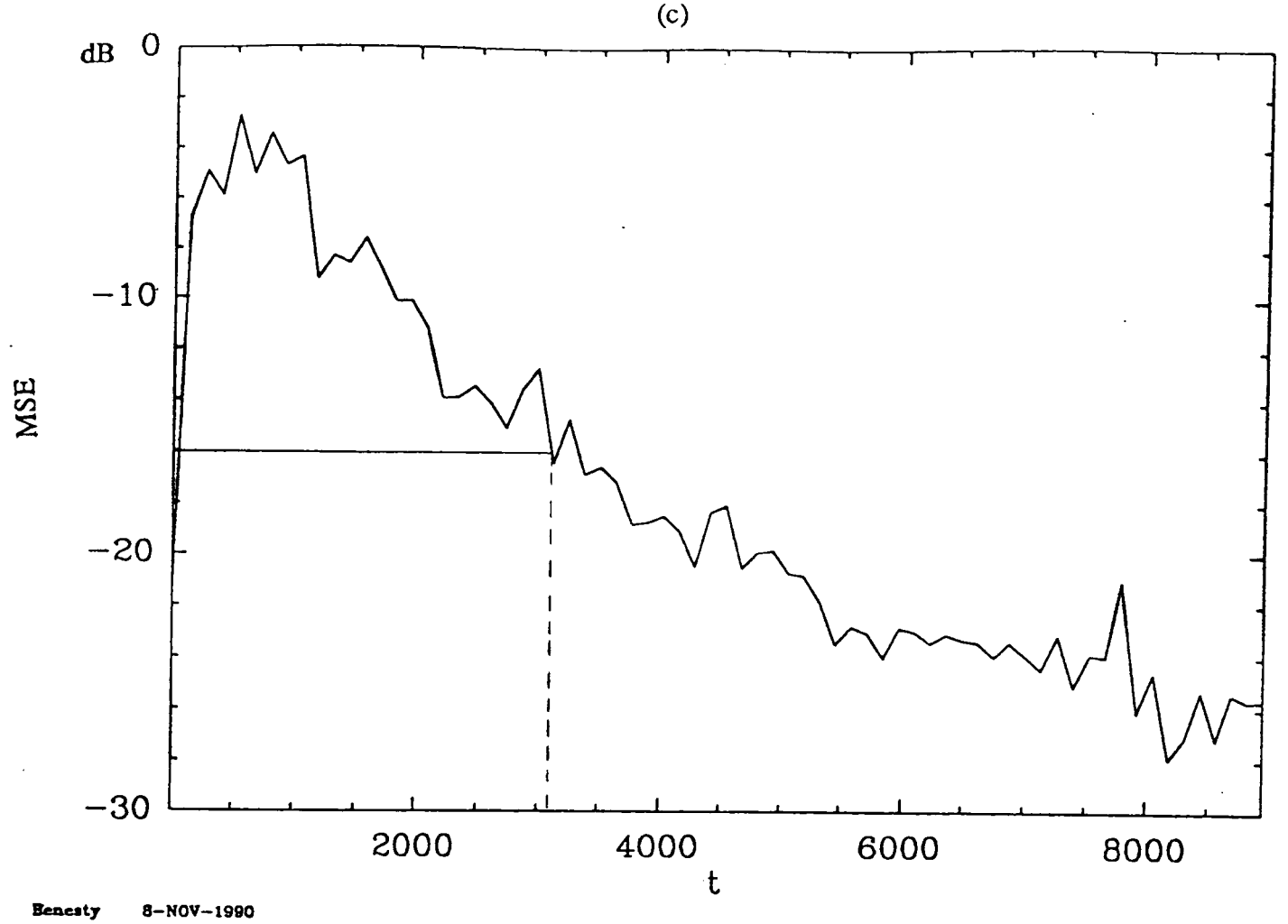


Fig. 4.1: Courbes d'erreur des algorithmes LMS (a) et FD-FELMS (b)



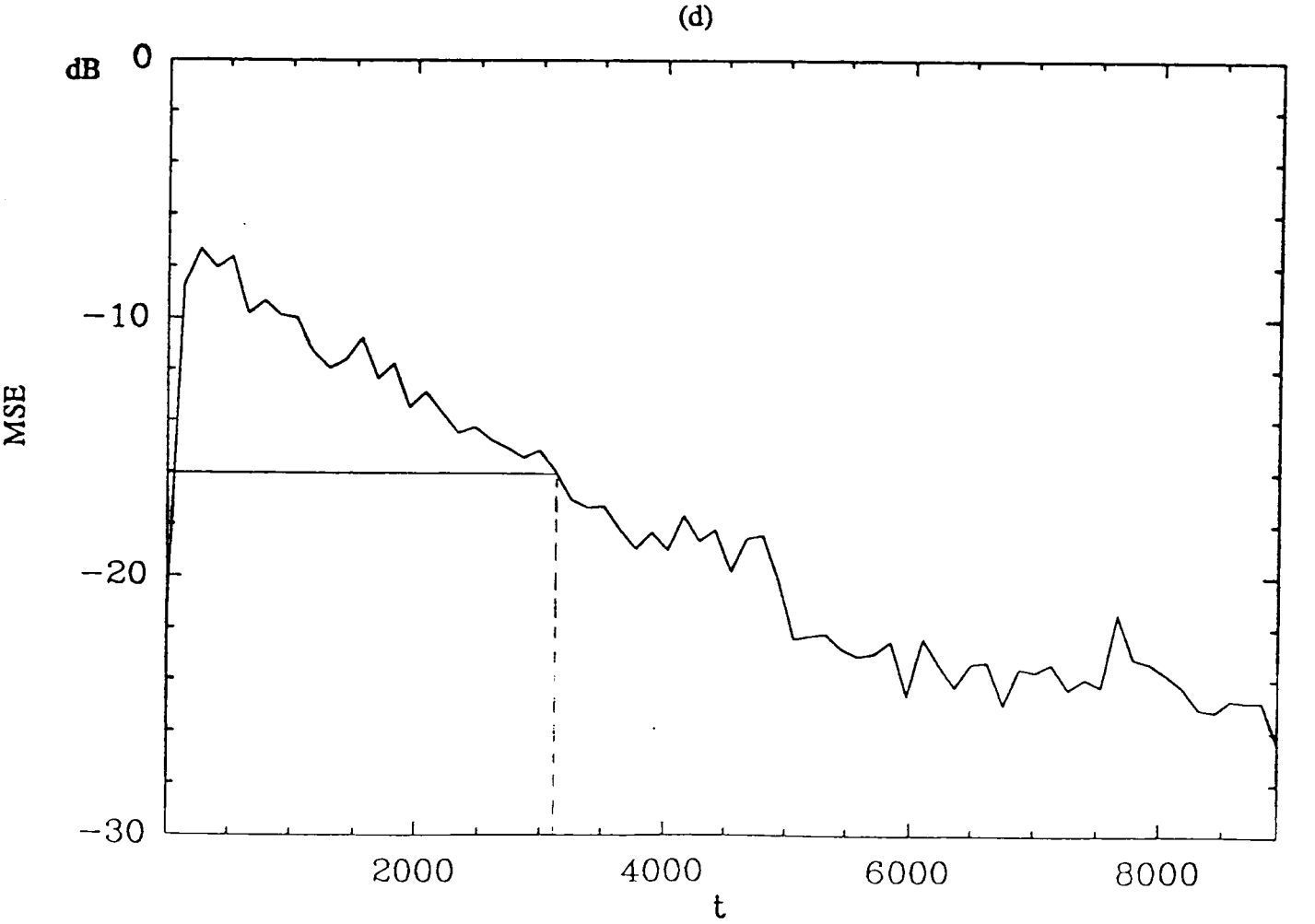


Fig. 4.1: Courbes d'erreur des algorithmes MDF (c) et NB-FELMS (d)

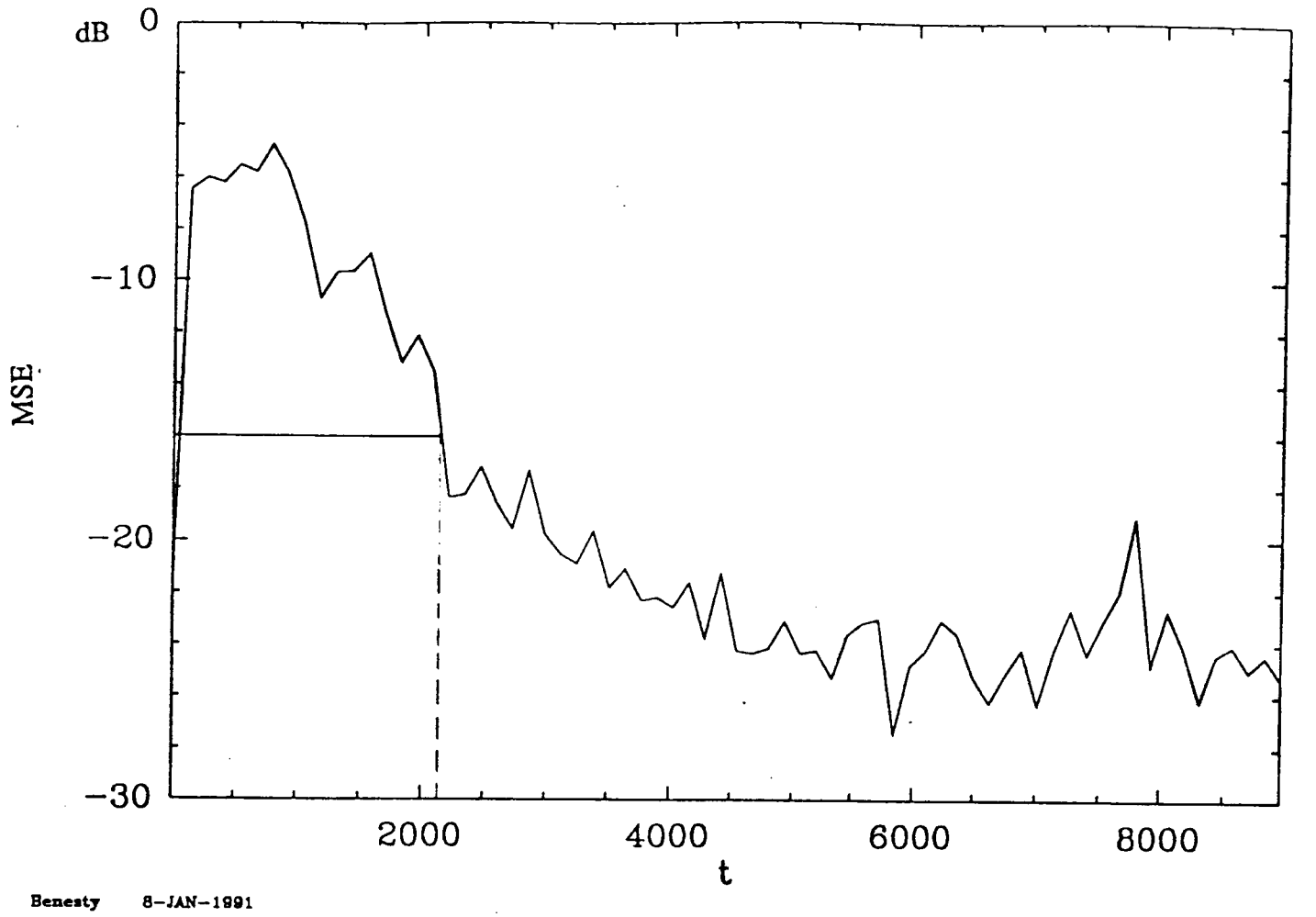
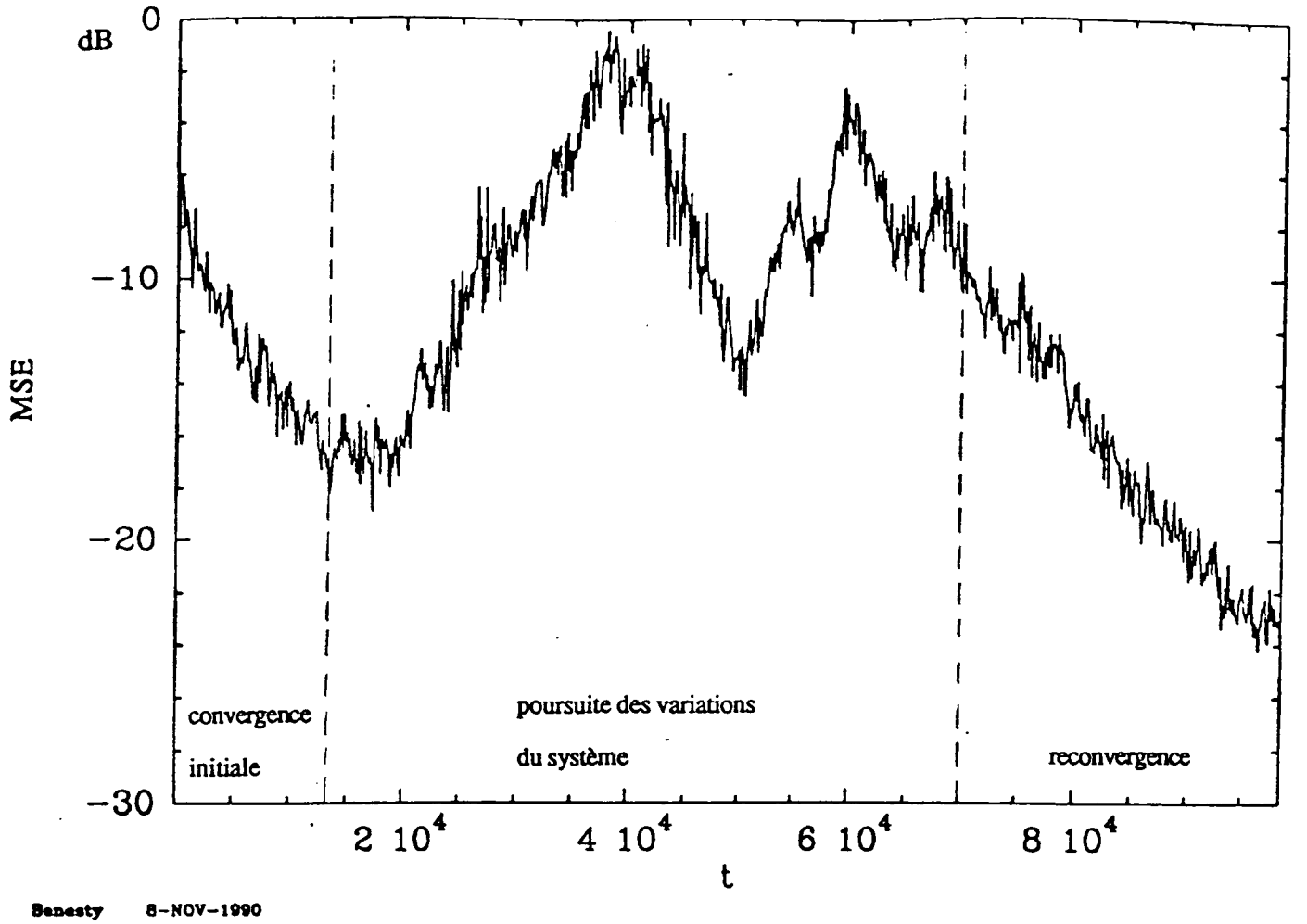


Fig. 4.1e: Courbe d'erreur de l'algorithme SB-RLS



(b)

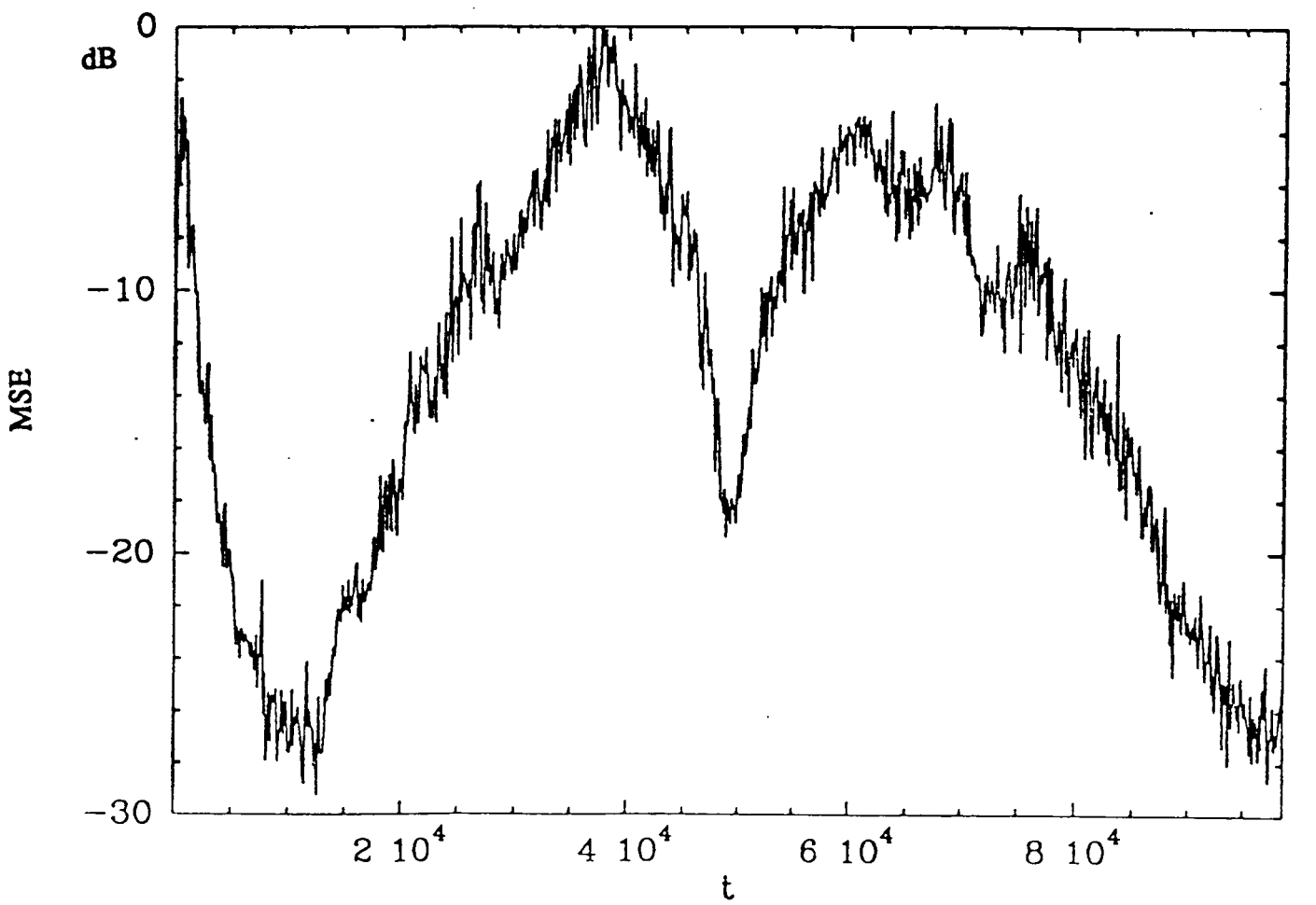
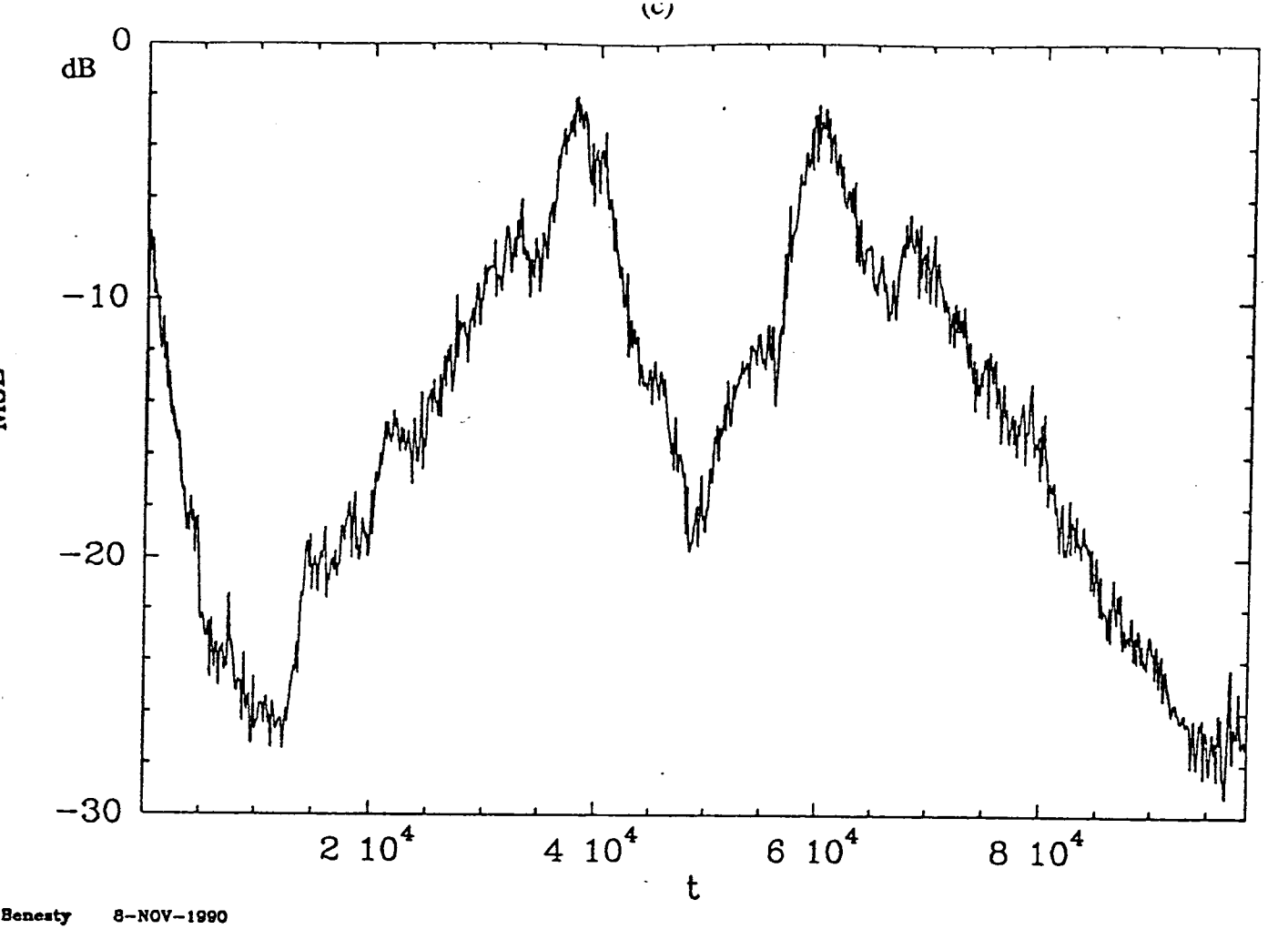


Fig. 4.2: Courbes d'erreur des algorithmes LMS (a) et MDF (b)



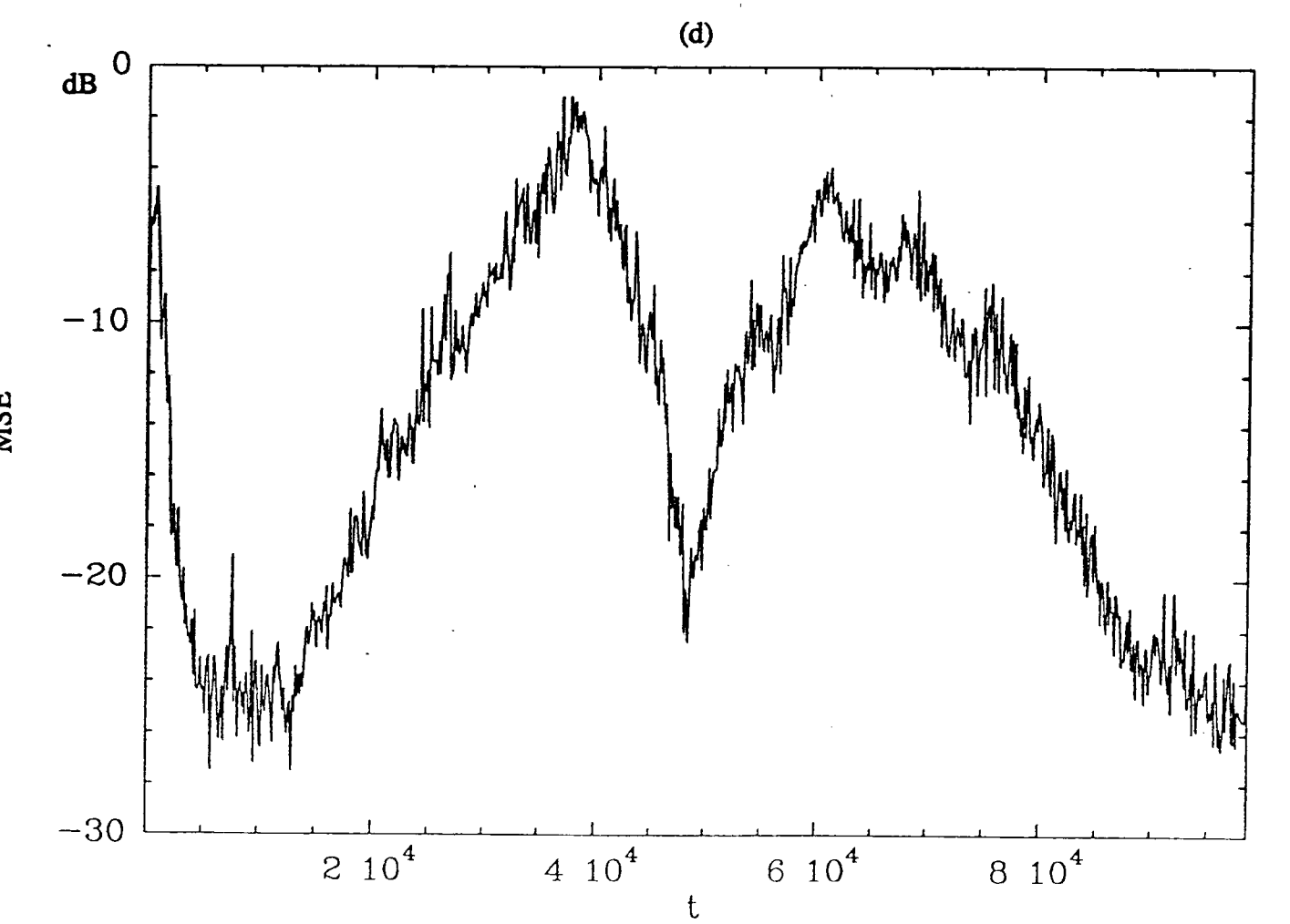


Fig. 4.2: Courbes d'erreur des algorithmes NB-FELMS (c) et SB-RLS (d)

CHAPITRE V

RESUME DES PRINCIPAUX ALGORITHMES OBTENUS

Le but de ce très court chapitre est de résumer partiellement la démarche que nous avons suivie, et de caractériser le comportement de chaque algorithme que nous avons proposé.

L'axe principal de ce travail se trouve à l'articulation entre complexité arithmétique et comportement adaptatif (convergence essentiellement). Jusqu'ici, une réduction de la charge de calcul semblait se jouer par une perte des possibilités en comportement adaptatif, sinon en vitesse de convergence (exception notable du FLMS), du moins en capacité de poursuite. Nous avons donc tout d'abord recherché à mieux comprendre le mécanisme de la formulation par bloc d'un algorithme séquentiel, qui demeure l'outil incontournable de la diminution du nombre d'opérations. La nouvelle formulation que nous avons proposée, appliquée au LMS, nous a conduit a des algorithmes équivalents au LMS et à charge de calcul réduite. Ensuite, cette compréhension du mécanisme par bloc nous a conduit à proposer un ensemble d'algorithmes au comportement adaptatif amélioré.

5.1. LES ALGORITHMES EQUIVALENTS (OU PROCHES) DU LMS

5.1.1. L'algorithme FELMS

Cet algorithme est une formulation exacte par bloc du LMS avec réduction de la charge de calcul, même pour une taille de bloc aussi petite que N=2. En effet, pour ce même bloc, le nombre de multiplications est réduit de 25% par rapport au LMS avec à peu près le même nombre d'additions. Ceci montre l'intérêt de cet algorithme pour des tailles de blocs petites et pour des filtres de petites longueurs, car dès que L>8, le nombre total d'opérations commence à diminuer.

Il faut bien noter que les algorithmes LMS et FELMS ont exactement le même comportement adaptatif, quelle que soit la taille du bloc, et ceci parce que dans cette nouvelle formulation par bloc, nous avons pris toutes les précautions nécessaires en vue de conserver l'équivalence. Cette remarque peut sembler surprenante, car dans un cas le filtre est adapté à chaque instant, tandis

que dans l'autre cas il est adapté une fois par bloc.

5.1.2. Les algorithmes FALMS ("Fast Approximate LMS")

Les algorithmes FALMS sont obtenus à partir du FELMS, en faisant des approximations (justifiées) sur la matrice triangulaire S qui est une estimation de la partie inférieure de la matrice de corrélation du signal d'entrée. Nous en proposons, en fait, deux:

- la première est de ne tenir compte que d'un certain nombre des premières diagonales de S qui sont les plus significatives puisque les dernières tendent souvent à être négligeables; et
- la seconde est d'arrêter l'adaptation des éléments de S après un certain nombre d'itérations, pour un signal d'entrée stationnaire.

Avec ces approximations, le nombre d'opérations des algorithmes correspondant est encore diminué par rapport au FELMS ainsi que la taille mémoire, et ceci, en ne modifiant que très légèrement leur comportement.

5.1.3. L'algorithme FD-FELMS

Celui-ci est une implantation exacte du LMS dans le domaine fréquentiel. Son intérêt est de travailler avec des FFT de petites longueurs (2N) et de diminuer davantage la complexité arithmétique ainsi que le nombre de mémoires par rapport au FELMS, lorsqu'on a affaire à des filtres longs. Pour des filtres de petites longueurs ou pour des tailles de blocs petites, il est plus avantageux d'utiliser le FELMS. Les deux algorithmes FELMS et FD-FELMS, qui sont mathématiquement équivalents, se complétent.

Il est clair que les approximations du paragraphe précédent s'appliquent aussi pour l'algorithme FD-FELMS.

Le principal inconvénient des algorithmes vus jusqu'ici est leur faible vitesse de convergence qui est identique à celle du LMS (dans le cas du FELMS et du FD-FELMS), sinon comparable (dans le cas des algorithmes FALMS).

Les algorithmes suivants accélérent la vitesse de convergence par rapport au LMS, tout en réduisant la charge de calcul.

5.2. UN ALGORITHME A VITESSE DE CONVERGENCE AMELIOREE (MDF)

Nous profitons, ici, des propriétés des transformées orthogonales qui, généralement, réduisent le rapport ($\lambda_{max}/\lambda_{min}$) de la matrice de corrélation et qui se traduit par une amélioration de la vitesse de convergence. On peut donc utiliser cette technique dans le FD-FELMS (puisque la FFT est une transformée orthogonale). Pour cela, on remplace le pas d'adaptation dans le domaine fréquence par une matrice diagonale dont les éléments sont proportionnels à l'inverse de l'énergie dans chacune des bandes de fréquence.

De plus, des simulations montrent que la matrice de corrélation triangulaire inférieure n'a plus aucun effet, elle sera donc négligée et l'algorithme obtenu (MDF) nécessite encore moins d'opérations que le FD-FELMS, car on n'aura plus à inverser S ni à calculer ses éléments.

Le MDF converge donc plus vite que le LMS, mais il poursuit moins bien les non-stationnarités du système du fait que le filtre reste fixe pendant tout le bloc.

5.3. UN ALGORITHME A CONVERGENCE ET POURSUITE AMELIOREES (NB-FELMS)

Nous avons été amenés à trouver cet algorithme à cause des problèmes que posent, en général, les algorithmes par bloc lorsqu'il s'agit de poursuite.

L'idée vient du vecteur erreur obtenu dans le FELMS. Pourquoi ne pas construire un critère à partir de ce bloc erreur? Ce faisant, nous avons dérivé un algorithme par bloc (NB-FELMS) qui a l'avantage de converger aussi vite que le MDF et de poursuivre mieux que le LMS. Il est tout aussi possible de diminuer sa charge de calcul en utilisant, par exemple, la FFT comme intermédiaire de calcul de la même manière que nous l'avons fait pour le FD-FELMS, et le nombre d'opérations requis est à peine supérieur à ce dernier.

On a donc là un algorithme par bloc qui poursuit mieux qu'un algorithme séquentiel, ce qui peut paraître paradoxal, la raison en est que l'algorithme NB-FELMS est un raffinement du LMS, qui apparaît à l'intérieur d'un bloc en tenant compte des variations des coefficients du filtre à l'intérieur du bloc.

5.4. UN ALGORITHME INTERMEDIAIRE ENTRE LE NLMS ET LE RLS (SB-RLS)

Nous savons, d'une part, que l'algorithme RLS a une vitesse de convergence optimale et que, d'autre part, pour de longues réponses impulsionnelles on peut approximer à zéro les derniers éléments de corrélation. L'idée est donc de déconnecter la taille de cette matrice de corrélation de la longueur du filtre. Un premier pas est fait dans ce sens, puisque l'algorithme proposé (SB-RLS) utilise une matrice de corrélation qui est liée à la dimension du bloc et non plus à la longueur du filtre. Cet algorithme a été simplement obtenu à partir du FELMS en complétant la matrice S.

On tire deux cas importants du SB-RLS suivant la taille du bloc. Pour N=1, on retrouve le NLMS et pour N=L, on retrouve le BRLS.

Le SB-RLS converge beaucoup mieux que le LMS mais il garde un comportement "typé" d'algorithmes par bloc que le NB-FELMS n'a pas.

CHAPITRE VI

COMPORTEMENT DES ALGORITHMES OBTENUS DANS LE CAS DE LA PAROLE

6.1. INTRODUCTION

Au chapitre 4, nous avons comparé différents algorithmes dans le cas d'un signal d'entrée stationnaire mais fortement corrélé. Nous avons vu, aussi, leur comportement pour un système non-stationnaire (mouvement d'un objet dans la salle). C'était une première étape pour bien maîtriser le fonctionnement des algorithmes obtenus.

Dans ce chapitre, nous proposons d'aller un peu plus loin dans les simulations, puisque l'objectif visé est l'annulation d'écho acoustique, et dans ce cas, le signal d'entrée est la parole qui est fortement non-stationnaire. Il serait donc intéressant de comparer nos algorithmes à des algorithmes aussi classiques que le NLMS et que le FRLS dans une telle situation, pour se rendre compte plus objectivement de ce que nous proposons.

Au prochain paragraphe, nous donnons une classification très succincte des sons de la parole pour mieux interpréter les courbes d'erreur des algorithmes, tandis qu'au paragraphe 6.3, nous décrivons les conditions de simulations, la phrase utilisée ainsi que les critères de performances. Enfin, dans la section 6.4, nous comparons les algorithmes FRLS, NLMS, MDF, NB-FELMS et SB-RLS, en utilisant deux critères différents.

6.2. CLASSIFICATION SOMMAIRE DES SONS DE LA PAROLE

Le signal de parole est fortement non-stationnaire. En fait, la parole est faite de phonèmes et peut être considérée comme stationnaire sur des durées allant de 10 à 30 ms. Les sons de la parole peuvent être classés, de manière un peu sommaire, en trois catégories:

a) Les sons voisés (ou sons sonores):

Ce sont des signaux quasi périodiques, très riches en harmoniques d'une fréquence fondamentale, appelée pitch ou encore fréquence de mélodie, qui varie en moyenne de 70 à 150 Hz pour les hommes et de 100 à 400 Hz chez les femmes et les enfants. Les sons voisés sont de forte énergie.

b) Les sons non voisés (ou sons sourds):

Les sons sourds sont des signaux qui ne présentent pas de structure périodique. Ils ont les caractéristiques spectrales d'un bruit légèrement coloré. Ils sont d'énergie moyenne.

c) Les silences:

Ce sont des intervalles où il n'y a pas de signal utile. Ces intervalles de silence sont nécessaires pour l'intelligibilité du signal vocal, ils occupent une part importante du temps de locution. En pratique, il s'agit de bruits d'origines diverses, d'énergie négligeable devant celle du signal utile.

A ces trois catégories s'ajoutent les sons voisés très pauvres en harmoniques (voisées nasales) et des sons plosifs caractérisés par un apport instantané d'énergie, faisant passer de manière très brève du silence à un son qui peut être voisé ou non.

C'est par la succession temporelle de tous ces différents sons qu'est constituée la parole. Il en résulte son caractère fortement non-stationnaire.

6.3. DESCRIPTION DU SIGNAL AINSI QUE DES CRITERES DE PERFORMANCES UTILISES

La sortie du système à modéliser est le résultat d'un signal de parole convolué avec une réponse impulsionnelle connue de salle mesurée et tronquée à L=256 points. Cette sortie peut donc être vue comme le signal capté par le microphone à partir du haut-parleur (qui reçoit de la parole lointaine) en l'absence du locuteur dans cette même salle. Cette sortie est aussi l'écho acoustique que l'on cherche à réduire.

Le signal d'entrée est une phrase prononcée par un locuteur masculin, échantillonnée à 16 kHz. Cette phrase est la suivante:

"Un loup s'est jeté immédiatement sur la petite chèvre".

La figure 6.1 représente l'évolution dans le temps de cette phrase complète, alors que sur la figure 6.2, c'est l'évolution de la puissance (moyennée sur 128 points) de ce signal qui est donnée. On pourra remarquer que cette puissance varie considérablement au cours du temps et que les pics autour de -55 dB correspondent aux intervalles de très faible énergie (silences).

Le critère de performance couramment utilisé en annulation d'écho est l'évolution temporelle de l'énergie de l'erreur de filtrage normalisée par l'énergie du signal d'écho y(n), exprimé en dB:

$$J_1(n) = 10\log_{10}\left(\frac{<e^2(n)>}{<y^2(n)>}\right)$$

où < > désigne une moyenne temporelle effectuée sur un certain nombre d'échantillons (128). Ce critère est en fait l'atténuation de l'énergie de l'écho.

Malheureusement, le critère de performance précédent privilégie les zones du signal de forte énergie et il n'est pas très significatif dans les zones de très faible énergie (silences) où l'erreur de filtrage et le signal ont des niveaux comparables. Ainsi, sur une courbe d'erreur utilisant ce critère, il sera difficile d'avoir un jugement objectif surtout si les "temps morts" sont nombreux. Pour cela, on utilisera aussi le critère suivant:

$$J_2(n) = 10\log_{10}\left(<e^2(n)>\right)$$

qui donne l'évolution temporelle de l'erreur de filtrage et qui semble être plus pertinent puisqu'il ne privilégie aucune zone.

6.4. COMPORTEMENT DES ALGORITHMES

Dans ce paragraphe, nous comparons les algorithmes NLMS et FRLS aux trois suivants: MDF, NB-FELMS et SB-RLS.

La longueur du filtre RIF est égale à 256 et la taille du bloc est: N=64.

Un bruit blanc stationnaire est ajouté à la sortie du système et le rapport signal à bruit est de

l'ordre de 40 dB.

Pour l'algorithme NLMS, nous avons pris:

a) $e(n) = y(n) - X_n^t H_n$

b) $H_{n+1} = H_n + \frac{1}{X_n^t X_n} X_n e(n)$

L'algorithme FRLS numériquement stable diverge quand le signal d'entrée est la parole, à cause du caractère fortement non-stationnaire de ce dernier. Il existe, cependant, plusieurs méthodes simples pour le faire fonctionner sans interruption. La première [24] consiste à vérifier à chaque instant que la variable de vraisemblance est supérieure à une certaine valeur (0,1, ici). Quand elle est plus petite ou égale, on réinitialise, alors, certaines variables de l'algorithme. Pour cela, il faut constamment estimer l'énergie du signal d'entrée avec une fenêtre exponentielle (égale à 0,999) identique à celle utilisée dans l'algorithme. Cet algorithme est le FTFR [24].

La seconde méthode est d'ajouter deux variables dans l'algorithme FTF: une variable de régularisation (c) et une autre de rappel à zéro (g) [24]. Nous avons pris: c=0,008 et g=0,002. Cet algorithme est le FTFSR.

Nous les avons simulés tous les deux, car ils n'ont pas le même comportement.

L'algorithme MDF est celui explicité au chapitre 4, en prenant ici: α=0,5 et β=0,8.

Pour l'algorithme SB-RLS, nous n'avons rien changé sauf que nous avons initialisé $s_0(n-N)$ à 1, pour éviter des problèmes avec l'inversion de la matrice d'autocorrélation.

Enfin, le seul algorithme qui a été vraiment modifié est le NB-FELMS, car la version que nous avons donnée au chapitre 4 suppose que les statistiques du signal d'entrée ne varient pas au cours du temps, ce qui n'est pas le cas de la parole. La nouvelle version qui est déduite à partir du NLMS est la suivante:

(6.1)

a) $G_0(n) = \left[S(n)S_0^{-1}(n) + I\right]^{-1}$

b) $E_n^0 = G_0(n)\varepsilon_n$

c) $H_{n+1} = H_{n-N+1} + \rho \underline{X}^t(n) S_0^{-1}(n) G_0^t(n) E_n^0$

avec $0<\rho\leq 3$ et où la matrice diagonale $S_0(n)$ a déjà été définie.

Dans nos simulations, nous avons choisi $\rho=2,2$ et nous avons initialisé $s_0(n-N)$ à 1.

Les figures 6.3 représentent les courbes d'erreur des six algorithmes (NLMS, MDF, NB-FELMS, SB-RLS, FTFR et FTFSR), utilisant le critère $J_1(n)$. Les pics qu'on peut voir sur ces courbes autour de 0 dB sont dûs aux intervalles de faible énergie (silences) et coincident (mais dans le sens opposé) avec les pics autour de -55 dB de la figure 6.2. Comme nous l'avons expliqué au paragraphe précédent, ces pics ne sont pas significatifs et il faut les ignorer pour des comparaisons. Par ailleurs, pour avoir une écoute confortable, il faut que les courbes d'erreur soient en dessous de -20 dB.

Les figures 6.3 se ressemblent et il est difficile de les comparer.

Avec le critère $J_1(n)$, on ne se rend pas bien compte de la vitesse de convergence des algorithmes, ni de leur "réaction" pour un signal d'entrée non-stationnaire. C'est pour cela que nous donnons les courbes d'erreur des six algorithmes déjà cités, en fonction du second critère $J_2(n)$. Celui-ci ne représente pas la réalité de l'écoute, mais il a l'avantage d'être clair pour une comparaison de certains aspects des algorithmes.

On remarque, tout d'abord, que le comportement adaptatif du NLMS (voir fig. 6.4a) est très gêné par un signal d'entrée telle que la parole, et une conséquence de cette constatation, c'est qu'il converge très lentement. La courbe d'erreur du NLMS atteint son minimum que pendant les intervalles de silences.

L'algorithme FRLS est connu pour avoir une vitesse de convergence optimale. Cependant, les versions stabilisées numériquement et pour un signal comme la parole sont légèrement modifiées, et suivant les techniques utilisées, le comportement n'est pas le même. En effet, le FTFR (fig. 6.4e) converge très vite, mais présente parfois des pics, ce qui peut être gênant à l'écoute. L'algorithme FTFSR (fig. 6.4f), quant à lui, converge moins vite, mais une fois qu'il a convergé, il ne bouge presque plus.

L'algorithme SB-RLS (fig. 6.4d) ne semble pas être affecté par la parole et sa vitesse de convergence est comparable à celle du FTFR; il converge plus vite que le NB-FELMS. Toutefois, on peut voir des pics très réguliers sur la courbe d'erreur, dûs au calcul de l'erreur par bloc.

L'algorithme NB-FELMS (voir fig. 6.4c) converge plus vite que le FTFSR, tout en ayant un

régime permanent comparable. Cet algorithme semble être un bon compromis, et ce d'autant plus qu'il nécessite une charge de calcul nettement inférieure à celle du NLMS.

Enfin, le MDF (fig. 6.4b) en régime transitoire est moins bon que le NLMS, mais après un certain nombre d'itérations, il devient un peu meilleur, mais lui aussi a du mal à converger. L'autre inconvénient du MDF est sa mauvaise capacité de poursuite des non-stationnarités du système. Par conséquent, il ne semble pas être une bonne solution au problème de l'annulation d'écho, même si c'est l'algorithme qui nécessite le moins d'opérations de tous ceux que nous avons donnés.

6.5. CONCLUSION

Dans ce chapitre, nous avons comparé nos algorithmes au NLMS et au FRLS, quand le signal d'entrée est la parole. Nous avons remarqué que les algorithmes proposés sont supérieurs au NLMS, et en particulier le NB-FELMS qui a un comportement adaptatif comparable à celui du FRLS stabilisé, avec d'autre part, une charge de calcul inférieure à celle du NLMS. L'algorithme NB-FELMS est donc un bon compromis entre la complexité arithmétique, la vitesse de convergence et la capacité de poursuite.

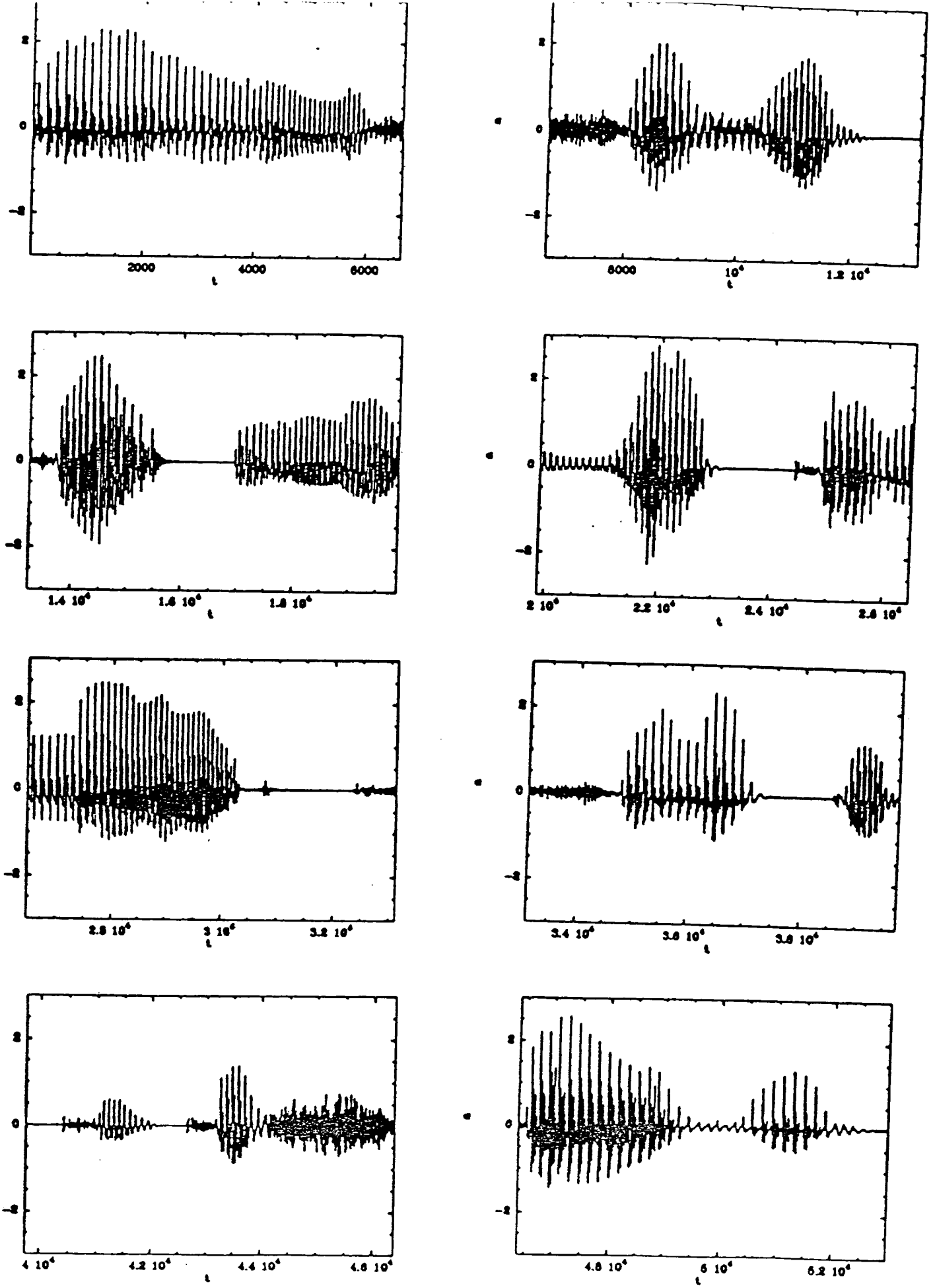


Fig. 6.1: Le signal de parole correspondant à la phrase: "Un loup s'est jeté immédiatement sur la petite chèvre"

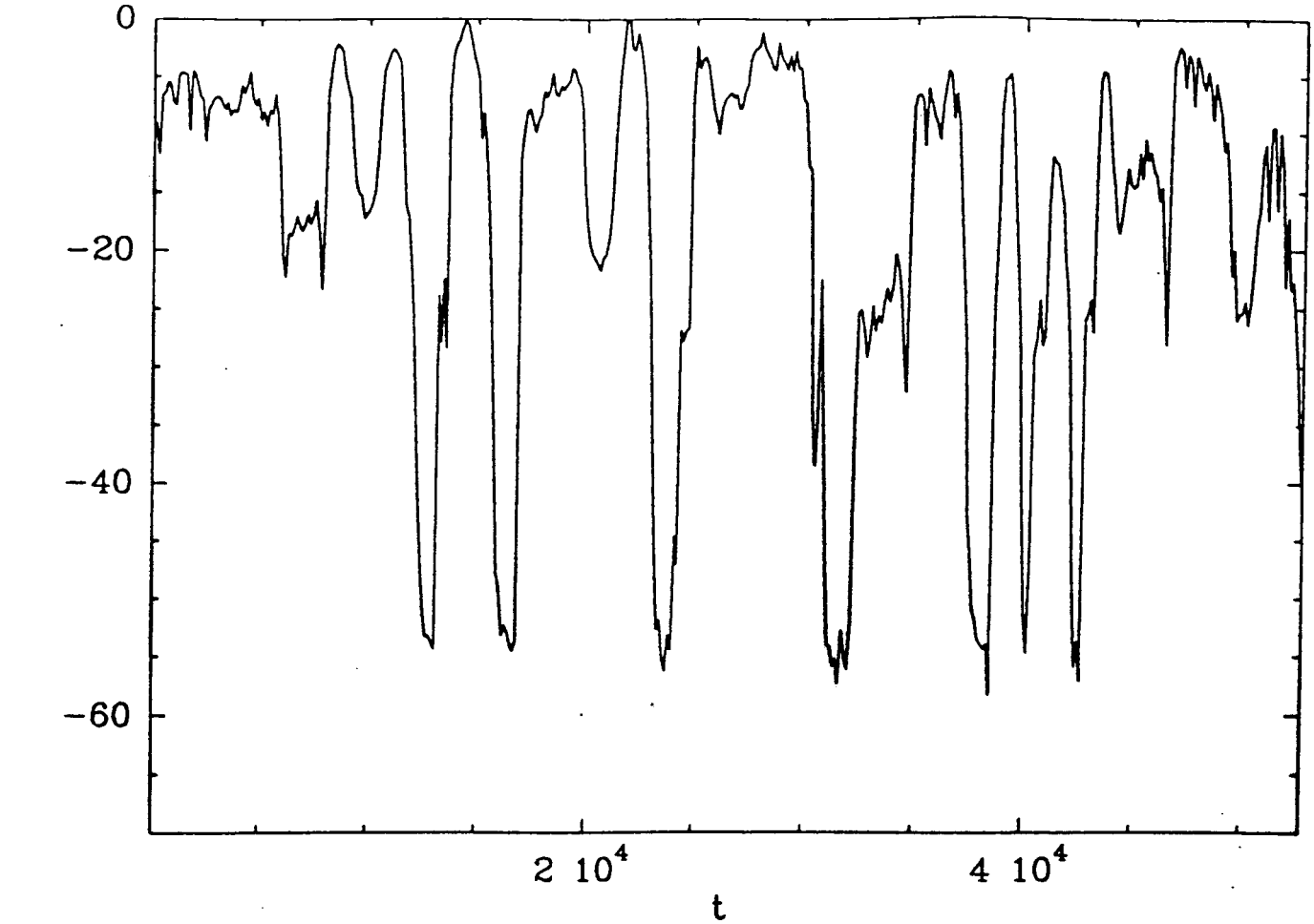
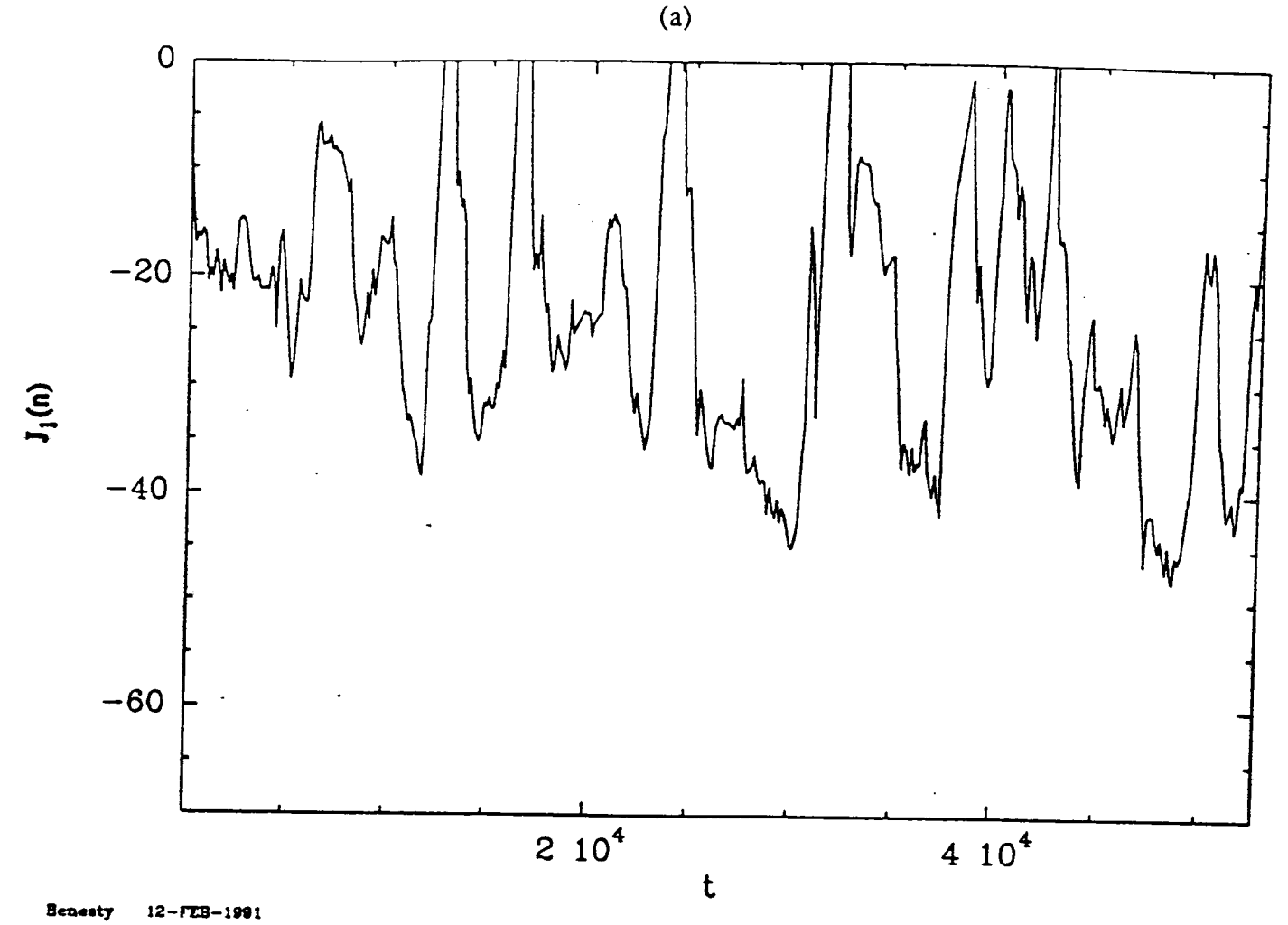


Fig. 6.2: Evolution de la puissance du signal d'entrée au cours du temps



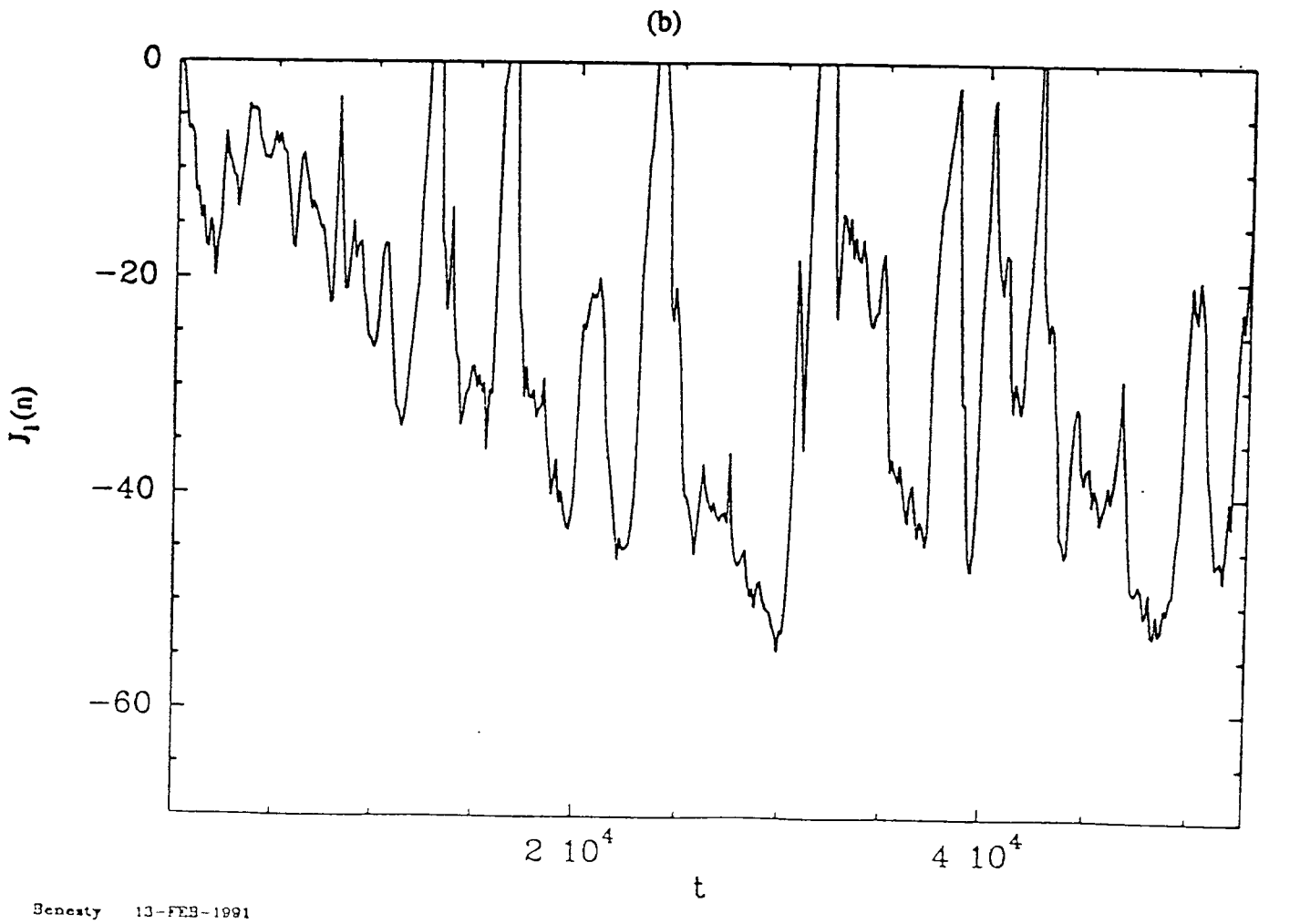
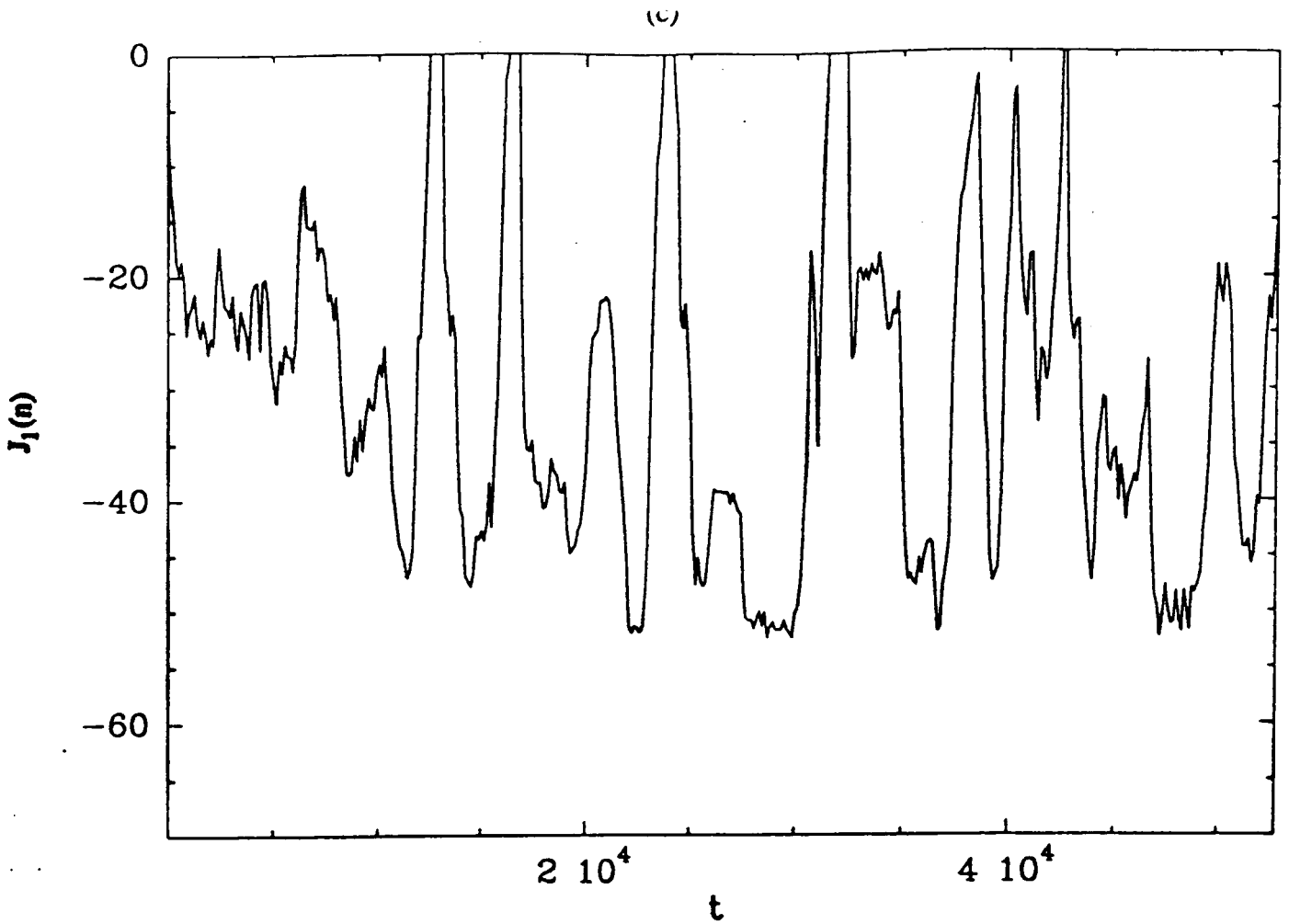


Fig. 6.3: Courbes d'erreur des algorithmes NLMS (a) et MDF (b), en fonction du critère $J_1(n)$



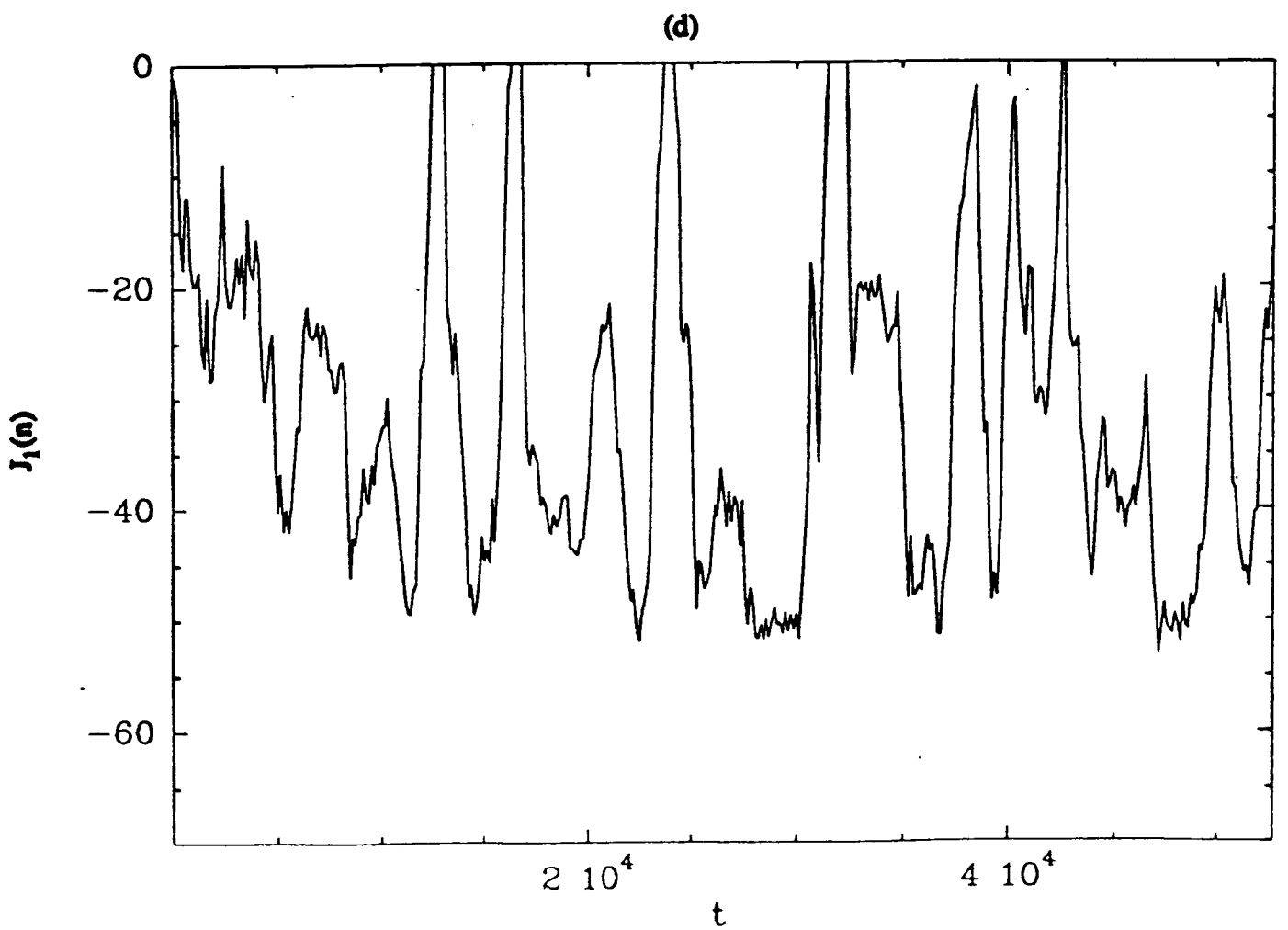
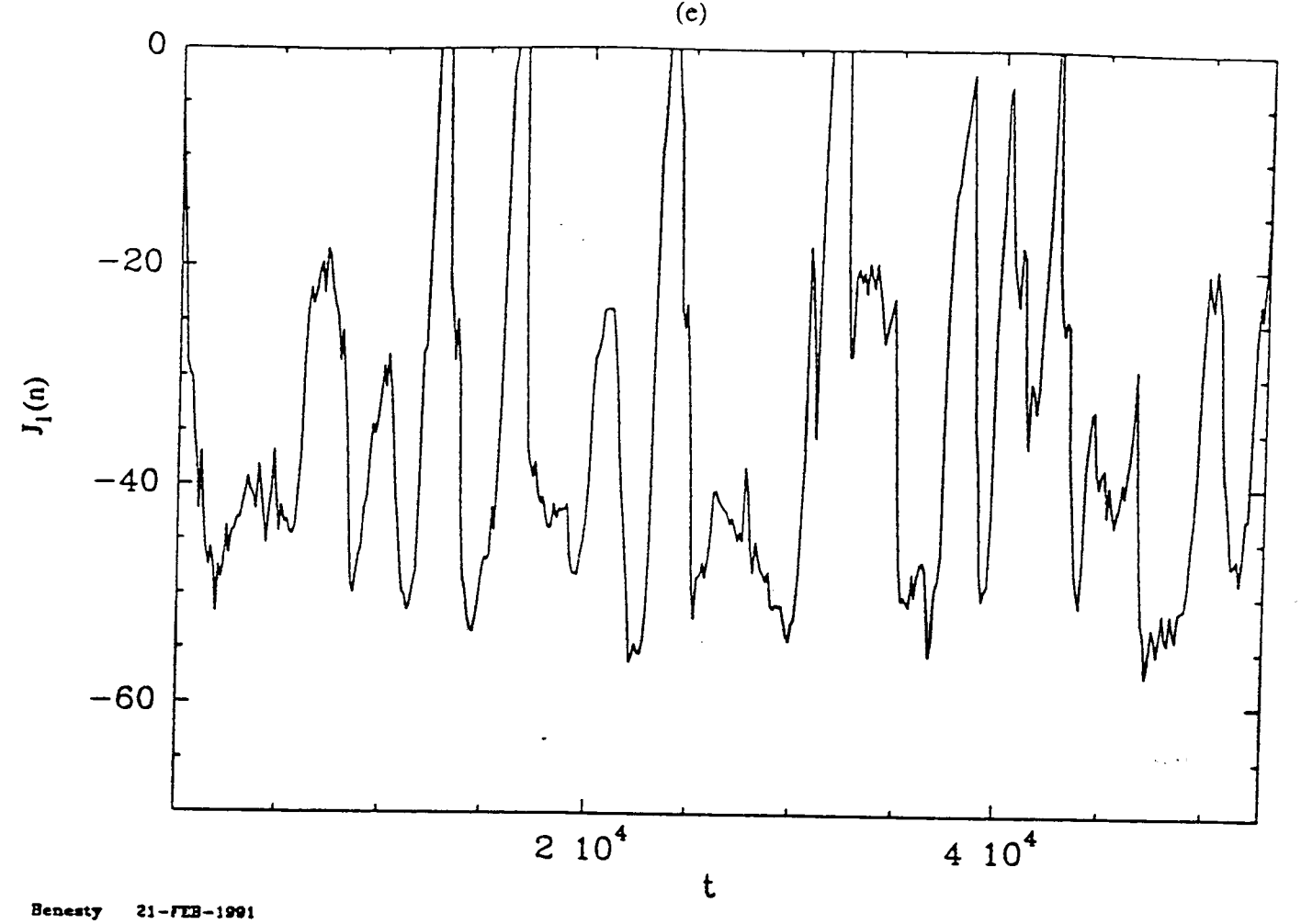


Fig. 6.3: Courbes d'erreur des algorithmes NB-FELMS (c) et SB-RLS (d), en fonction du critère $J_1(n)$.



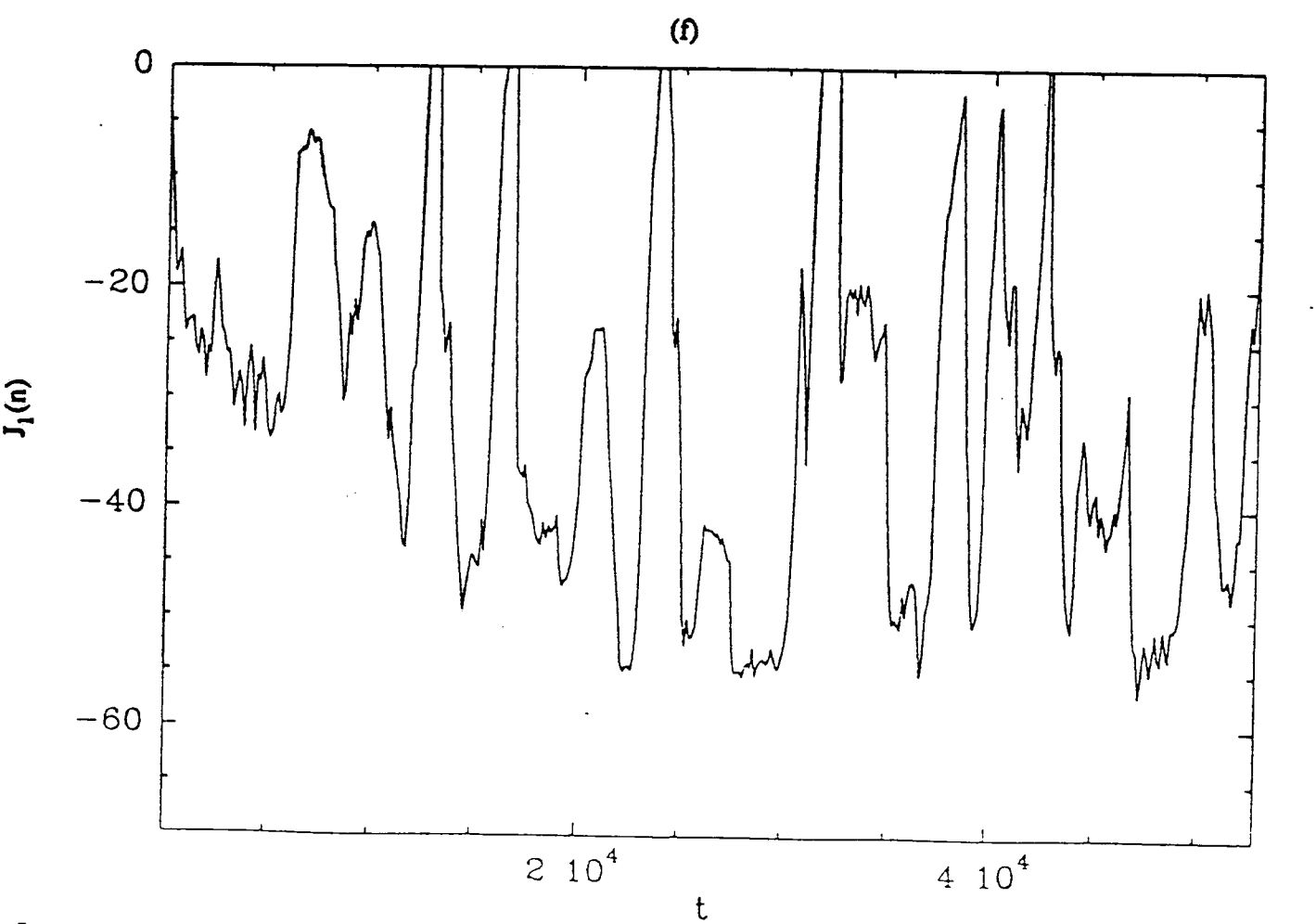
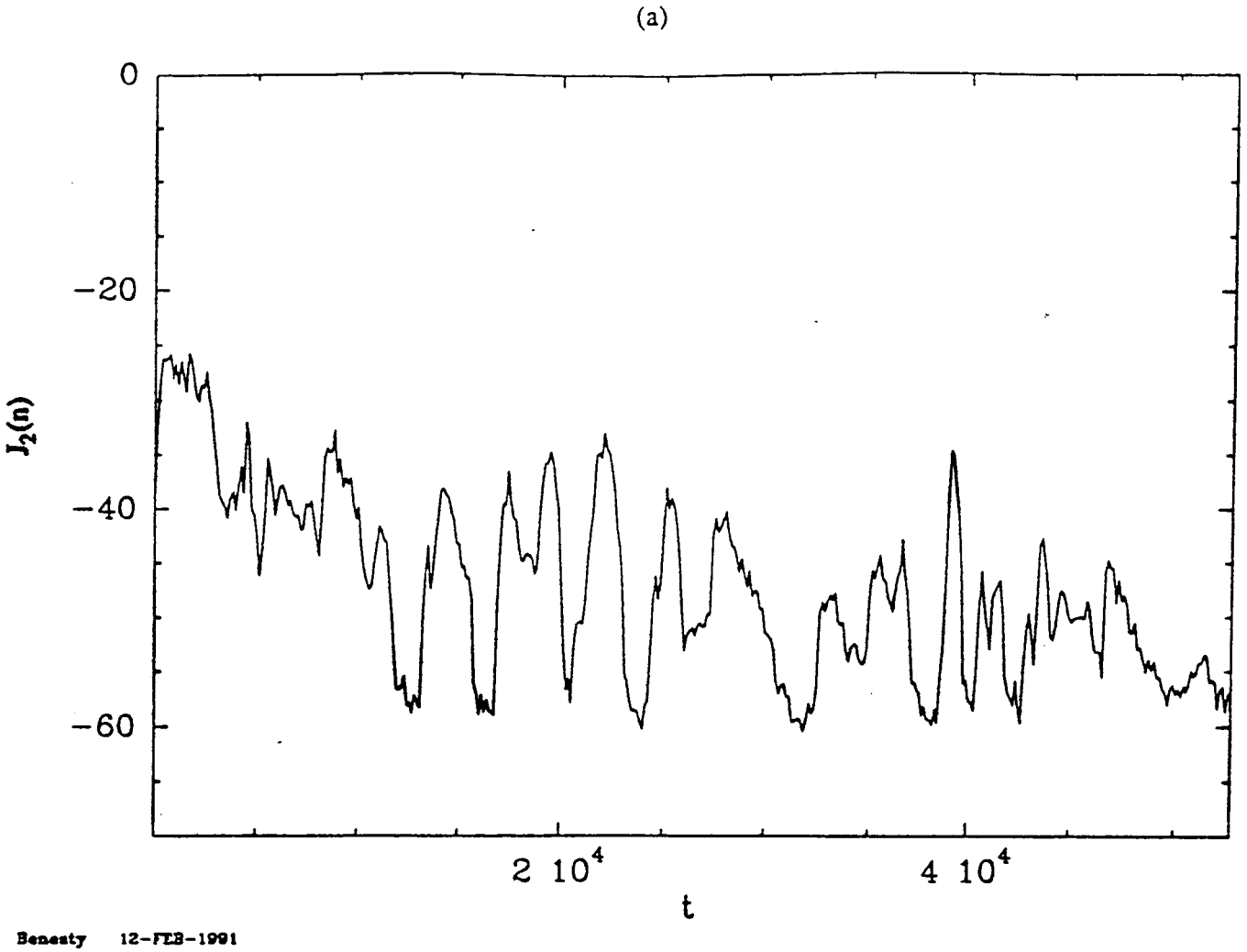


Fig. 6.3: Courbes d'erreur des algorithmes FTFR (e) et FTFSR (f), en fonction du critère $J_1(n)$



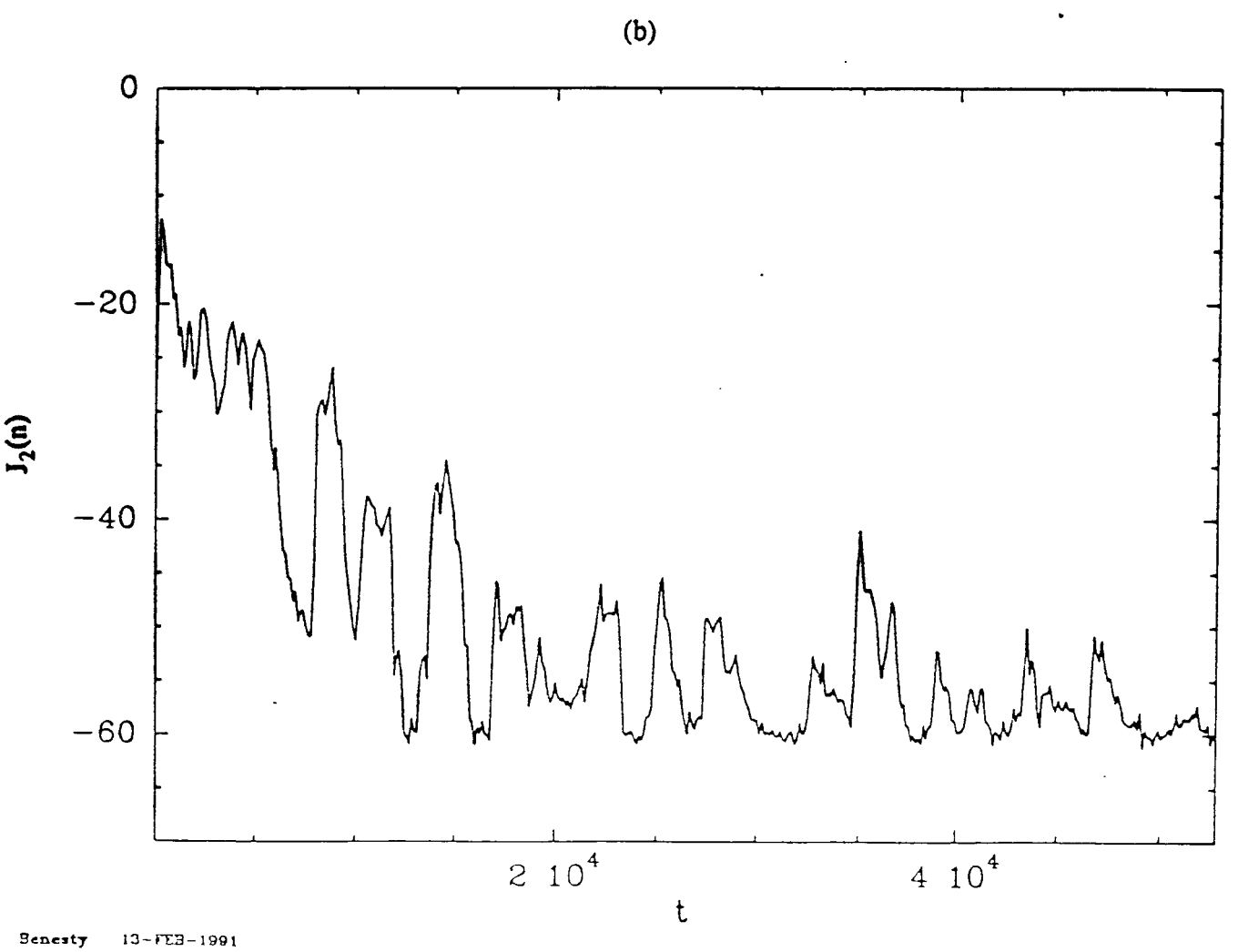
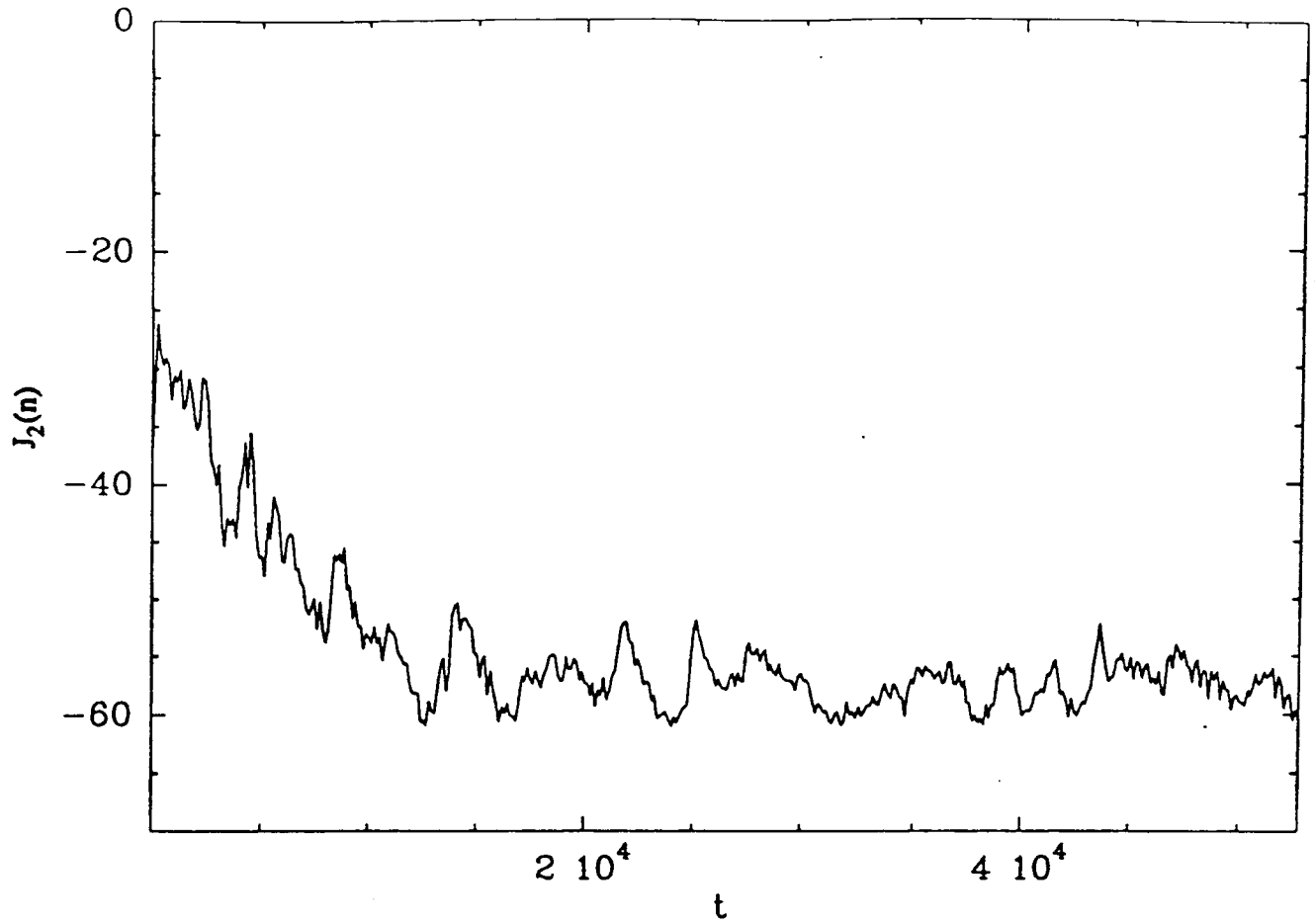


Fig. 6.4: Courbes d'erreur des algorithmes NLMS (a) et MDF (b), en fonction du critère $J_2(n)$



(d)

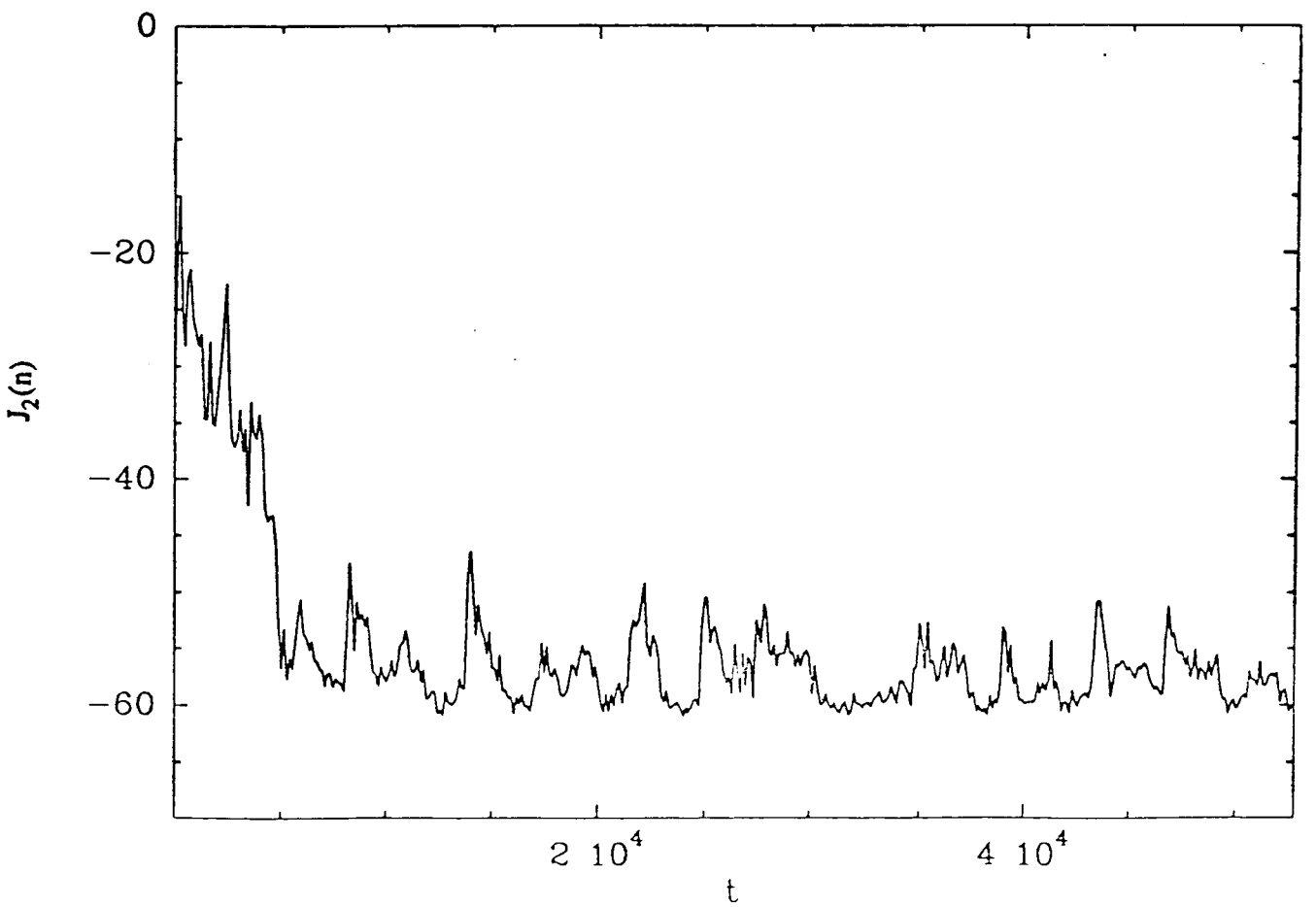
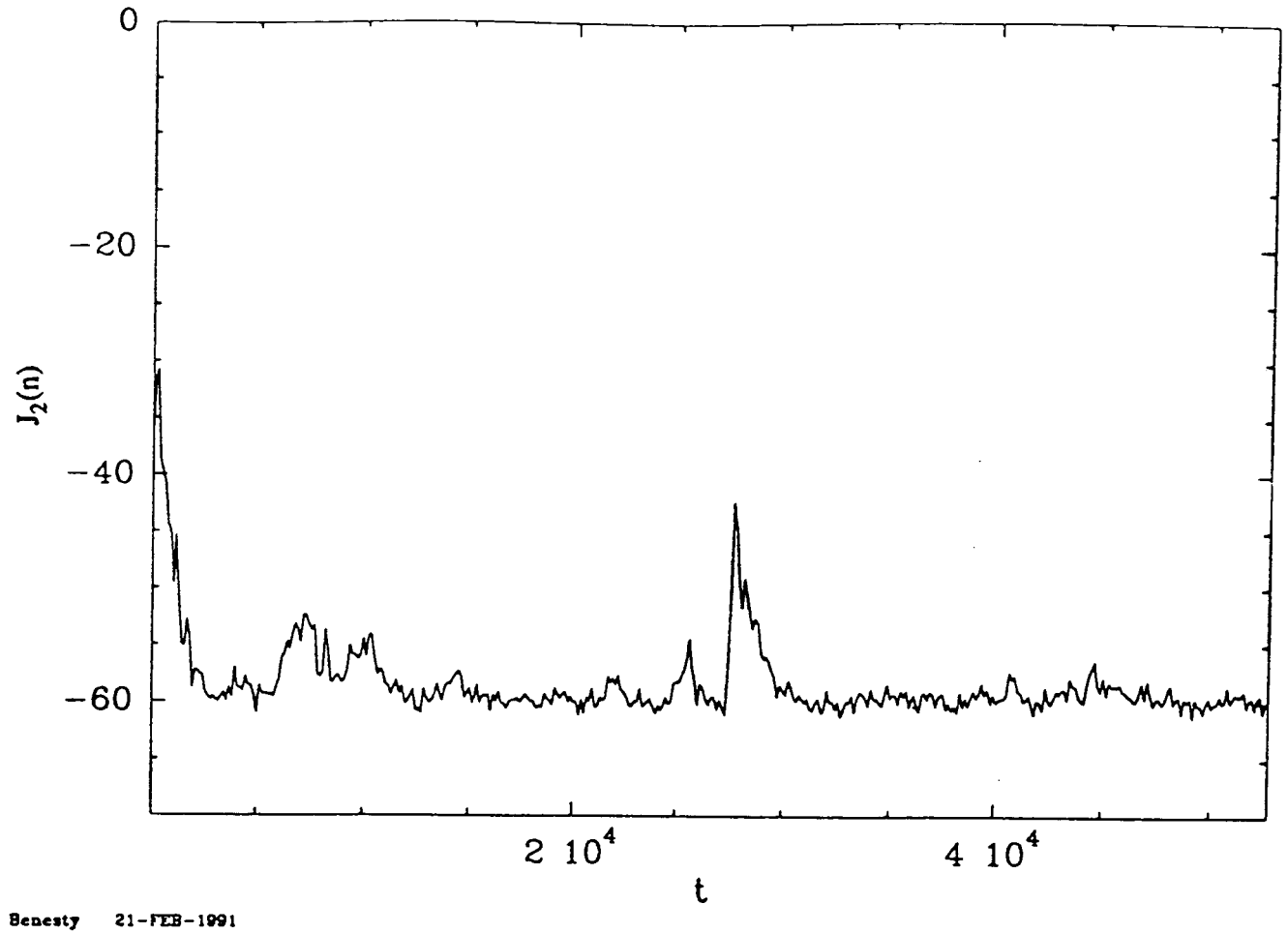


Fig. 6.4: Courbes d'erreur des algorithmes NB-FELMS (c) et SB-RLS (d), en fonction du critère $J_2(n)$

(e)



(f)

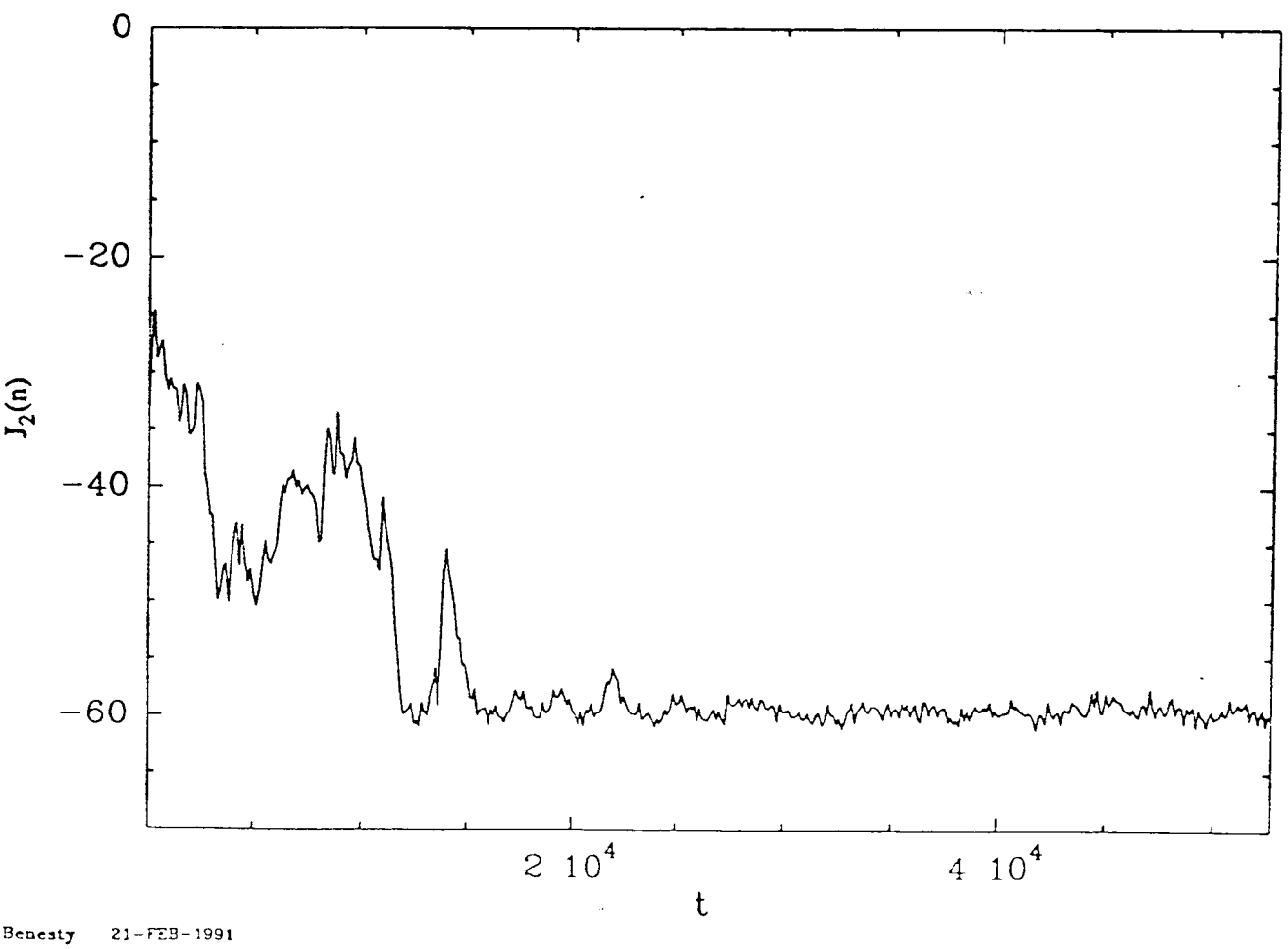


Fig. 6.4: Courbes d'erreur des algorithmes FTFR (e) et FTFSR (f), en fonction du critère $J_2(n)$

CHAPITRE VII

UN ALGORITHME CMA RAPIDE

(au sens de la complexité arithmétique)

7.1. INTRODUCTION

Dans ce chapitre, nous proposons de montrer que la formulation par bloc présentée dans le cas du LMS ainsi que la réduction de la charge de calcul qui découle, sont plus générales et s'appliquent à d'autres types d'algorithmes tel que l'Algorithme à Module Constant (CMA). Nous nous limiterons ici à la diminution du nombre d'opérations de cet algorithme et nous ne fournirons aucune "piste" en vue d'améliorer son comportement adaptatif.

Le CMA a été pour la première fois proposé par Godard [35] puis repris et étudié par Treichler et *al.* [37] pour une application en communication. En effet, dans plusieurs techniques de modulations, comme la modulation de fréquence (FM) et la modulation de phase (PM), le signal transmis a une enveloppe constante. Cependant, le signal reçu a perdu cette propriété à cause des trajets multiples et des interférences. Le CMA restaure un signal à module constant et augmente le rapport signal à bruit. Cet algorithme pour son fonctionnement, n'a besoin que de la connaissance du module du signal transmis et a l'avantage de ne pas dépendre d'une référence.

Néanmoins, le CMA pose quelques problèmes. Le premier est la minimisation d'une fonction coût non convexe, ce qui implique l'existence de minimums locaux. Le second problème est que l'algorithme peut capter un signal à module constant dû aux interférences au lieu du signal émis. Ces deux problèmes peuvent être résolus par simple initialisation du filtre [38]. Un autre défaut de cet algorithme est sa charge de calcul. Pour réduire celle-ci, Treichler et *al.* [36] ont proposé de calculer l'erreur (non-linéaire) dans le domaine temporel et d'implanter la partie filtrage et son adaptation dans le domaine fréquentiel. Malheureusement, cet algorithme n'est qu'une approximation du CMA et leurs auteurs ont observé qu'il avait une vitesse de convergence très lente [36]. Une conclusion de cette constatation est que les techniques classiques pour accélérer le temps de calcul de certains algorithmes sont très pénalisantes au niveau du comportement adaptatif.

Dans ce chapitre, nous montrons que l'on peut diminuer considérablement le nombre d'opérations du CMA sans aucune approximation et en travaillant par bloc (dont la dimension peut être choisie aussi petite que l'on veut), grâce à la nouvelle formulation que nous avons présentée au chapitre III, dans le cas du LMS. Ainsi, l'algorithme résultant (FCMA) a exactement le même comportement que l'algorithme initial. Nous donnons plusieurs versions rapides, toutes équivalentes entre elles au sens mathématique.

Le critère utilisé pour obtenir le CMA est le suivant:

(7.1)

$$J = \frac{1}{4} E\left\{\left[|y(n)|^2 - 1\right]^2\right\}$$

où y(n) est la sortie du filtre RIF complexe qui a pour entrée le signal émis après la transmission dans un canal. On suppose que le module du signal complexe transmis est égal à 1.

En minimisant la fonction coût précédente et en utilisant la technique du gradient, nous obtenons le CMA (voir figure 7.1) qui est décrit par l'ensemble des équations suivant:

(7.2)

a) $y(n) = X_n^t H_n$

b) $\alpha(n) = \mu\left[|y(n)|^2 - 1\right] y(n)$

c) $H_{n+1} = H_n - \alpha(n) X_n^*$

où * dénote le complexe conjugué et μ un réel strictement positif.

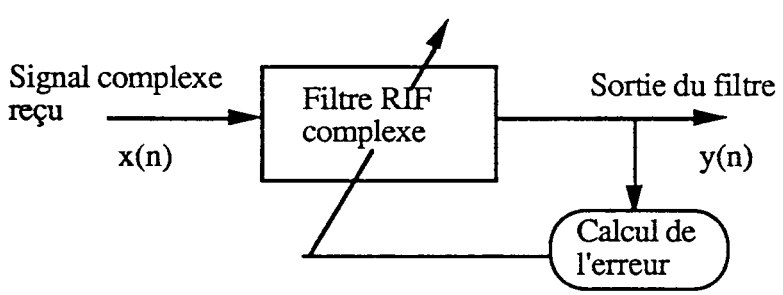


Figure 7.1: L'algorithme CMA

Sachant que le coût d'une multiplication complexe est de 4 multiplications et de 2 additions réelles, la complexité arithmétique de l'algorithme est de:

8L+5 multiplications

8L addtitions

Cet algorithme est le CMA1.

Si maintenant on utilise une version différente de la multiplication complexe (3 multiplications réelles au lieu de 4) [33], on obtient le CMA2 qui demande:

6L+5 multiplications

8L+4 additions

(pour plus de détails, voir l'article II).

On remarque que le nombre de multiplications réelles a été réduit de 25% dans la seconde version de l'algorithme.

7.2. PRINCIPE DE L'ALGORITHME CMA RAPIDE (FCMA)

Le principe de cet algorithme est le même que le FELMS à l'exception d'une difficulté due à une non-linéarité de l'erreur. On doit donc utiliser "l'astuce" du FELMS à un autre niveau de calcul.

Nous montrons dans ce paragraphe, sur un exemple (N=2), que l'on peut diminuer le nombre d'opérations de l'algorithme initial, pour cela, nous donnons d'abord une formulation exacte par bloc du CMA [39].

Considérons les équations (7.2) mais à l'instant n-1:

(7.3)

a) $y(n-1) = X_{n-1}^t H_{n-1}$

b) $\alpha(n-1) = \mu\left[|y(n-1)|^2 - 1\right] y(n-1)$

c) $H_n = H_{n-1} - \alpha(n-1) X_{n-1}^*$

en substituant (7.3c) dans (7.2a), nous obtenons:

(7.4)

$$y(n) = X_n^t H_{n-1} - \alpha(n-1) X_n^t X_{n-1}^*$$
$$= X_n^t H_{n-1} - \alpha(n-1)\, s(n)$$

où:

$$s(n) = X_n^t X_{n-1}^*$$

Ecrivons maintenant les sorties successives du filtre complexe aux instants n-1 et n:

(7.5)

$$\begin{bmatrix} y(n-1) \\ y(n) \end{bmatrix} = \begin{bmatrix} X_{n-1}^t \\ X_n^t \end{bmatrix} H_{n-1} - \begin{bmatrix} 0 & 0 \\ s(n) & 0 \end{bmatrix} \begin{bmatrix} \alpha(n-1) \\ \alpha(n) \end{bmatrix}$$

Le premier terme de cette expression est le calcul de la sortie d'un filtre RIF fixe à deux instants successifs, donc en utilisant les mêmes notations qu'au chapitre III, on a:

(7.6)

$$\begin{bmatrix} X_{n-1}^t \\ X_n^t \end{bmatrix} H_{n-1} = \begin{bmatrix} A_1^t & A_2^t \\ A_0^t & A_1^t \end{bmatrix} \begin{bmatrix} H_{n-1}^0 \\ H_{n-1}^1 \end{bmatrix}$$

L'adaptation du filtre va se faire une fois tous les deux instants en remplaçant (7.3c) dans (7.2c):

(7.7)

$$H_{n+1} = H_{n-1} - \alpha(n) X_n^* - \alpha(n-1) X_{n-1}^*$$

soit:

(7.8)

$$\begin{bmatrix} H_{n+1}^0 \\ H_{n+1}^1 \end{bmatrix} = \begin{bmatrix} H_{n-1}^0 \\ H_{n-1}^1 \end{bmatrix} - \alpha(n) \begin{bmatrix} A_0^* \\ A_1^* \end{bmatrix} - \alpha(n-1) \begin{bmatrix} A_1^* \\ A_2^* \end{bmatrix}$$

A présent, le CMA est strictement équivalent aux équations suivantes pour un bloc de deux sorties:

(7.9)

a) $$\begin{bmatrix} y'(n-1) \\ y'(n) \end{bmatrix} = \begin{bmatrix} A_1^t & A_2^t \\ A_0^t & A_1^t \end{bmatrix} \begin{bmatrix} H_{n-1}^0 \\ H_{n-1}^1 \end{bmatrix}$$

b) $$\begin{bmatrix} y(n-1) \\ y(n) \end{bmatrix} = \begin{bmatrix} y'(n-1) \\ y'(n) \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ s(n) & 0 \end{bmatrix} \begin{bmatrix} \alpha(n-1) \\ \alpha(n) \end{bmatrix}$$

c) $$\begin{bmatrix} H_{n+1}^0 \\ H_{n+1}^1 \end{bmatrix} = \begin{bmatrix} H_{n-1}^0 \\ H_{n-1}^1 \end{bmatrix} - \begin{bmatrix} A_1^* & A_0^* \\ A_2^* & A_1^* \end{bmatrix} \begin{bmatrix} \alpha(n-1) \\ \alpha(n) \end{bmatrix}$$

Notons que l'expression (7.9b) pose un problème: le calcul de y(n) semble nécessiter la connaissance de $\alpha(n)$ qui lui même dépend de y(n). Cependant, puisque la matrice (2x2) de (7.9b) est strictement triangulaire inférieure, y(n) en fait, ne dépend que de $\alpha(n-1)$ et cette équation est toujours soluble.

La diminution de la complexité arithmétique se fait comme pour le FELMS par les techniques de réduction de calcul du filtrage fixe, en écrivant (7.9) de la manière suivante:

(7.10)

a) $$\begin{bmatrix} y'(n-1) \\ y'(n) \end{bmatrix} = \begin{bmatrix} A_1^t (H_{n-1}^0 + H_{n-1}^1) + (A_2 - A_1)^t H_{n-1}^1 \\ A_1^t (H_{n-1}^0 + H_{n-1}^1) - (A_1 - A_0)^t H_{n-1}^0 \end{bmatrix}$$

b) $$\begin{bmatrix} y(n-1) \\ y(n) \end{bmatrix} = \begin{bmatrix} y'(n-1) \\ y'(n) \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ s(n) & 0 \end{bmatrix} \begin{bmatrix} \alpha(n-1) \\ \alpha(n) \end{bmatrix}$$

c) $$\begin{bmatrix} H_{n+1}^0 \\ H_{n+1}^1 \end{bmatrix} = \begin{bmatrix} H_{n-1}^0 \\ H_{n-1}^1 \end{bmatrix} - \begin{bmatrix} A_1^* (\alpha(n-1) + \alpha(n)) - (A_1 - A_0)^* \alpha(n) \\ A_1^* (\alpha(n-1) + \alpha(n)) + (A_2 - A_1)^* \alpha(n-1) \end{bmatrix}$$

Donnons les différentes étapes de calcul de cet algorithme:

étape a) calcul de y'(n-1) et de y'(n) par (7.10a)

étape b) calcul récursif de s(n):

(7.11)

$$s(n) = s(n-2) + [x(n)x^*(n-1) + x(n-1)x^*(n-2) - x(n-L)x^*(n-L-1) - x(n-L-1)x^*(n-L-2)]$$

étape c) y(n-1)=y'(n-1). Calculer α(n-1) par (7.3a) puis ensuite y(n) par (7.10b) et enfin α(n) par (7.2b)

étape d) calcul de H -éq.(7.10c)

étape e) incrémentation de n par 2 et retour à l'étape a).

Si l'on utilise le schéma classique d'une multiplication complexe, on a l'algorithme FCMA1 et son nombre d'opérations réelles par point de sortie est:

6L+11 multiplications

7L+11 additions

Dans l'algorithme que l'on référera par FCMA2, c'est un schéma rapide de la multiplication complexe qui est utilisé et la charge de calcul réelle par point de sortie est:

4.5L+11 multiplications

7L+18 additions

On peut remarquer que chaque version rapide réduit d'à peu près 25% le nombre de multiplications et diminue aussi légèrement le nombre d'additions (ce qui n'est pas le cas pour le FELMS quand N=2) comparée à la version initiale. Le nombre total d'opérations réelles est diminué de 20% dans cet exemple par rapport au CMA.

7.3. DISCUSSION ET CONCLUSION

L'exemple N=2 se généralise pour une taille de bloc quelconque égale à une puissance de 2, par un calcul récursif. Plus la taille du bloc est grande et plus le nombre d'opérations diminue. Mais notre approche n'est valable que pour des tailles de blocs assez petites par rapport à la longueur du filtre et l'on doit toujours avoir: N<L.

Une comparaison du nombre d'opérations par point de sortie des algorithmes implantés dans le domaine temporel pour différentes longueurs de filtres est donnée dans le tableau 7.1. On peut constater que la complexité arithmétique est diminuée de 50% pour un filtre de longueur L=64 et une taille de bloc aussi petite que N=8.

		CMA1		CMA2		FCMA1		FCMA2	
L	N	Nbre d'adds	Nbre de mults	Nbre d'adds	Nbre de mults	Nbre d'adds	Nbre de mults	Nbre d'adds	Nbre de mults
32	8	256	261	260	197	214	155	229	128
64	8	512	517	516	389	346	263	361	209
128	16	1024	1029	1028	773	591	419	614	338
256	32	2048	2053	2052	1541	967	677	999	555
512	64	4096	4101	4100	3077	1586	1112	1696	920

Tableau 7.1: Comparaison du nombre d'opérations par point de sortie des algorithmes CMA et FCMA (version temporelle)

Il est aussi possible d'utiliser la FFT comme intermédiaire de calcul. Dans ce cas, la longueur de la FFT est à peine deux fois plus grande que la taille du bloc et elle est donc complètement indépendante de la longueur du filtre. Le tableau 7.2 donne la complexité arithmétique pour plusieurs longueurs de filtres de l'implantation du FCMA dans le domaine fréquentiel. On remarquera que la version temporelle est intéressante pour les filtres de petites

longueurs tandis que la version fréquentielle est efficace pour des tailles moyennes et grandes. A partir de L=128, il commence à devenir intéressant de travailler avec la FFT.

		FFT-CMA1		FFT-CMA2	
L	N	Nbre d'adds	Nbre de mults	Nbre d'adds	Nbre de mults
32	8	275	143	283	127
64	16	378	206	386	190
128	16	628	304	636	272
256	32	839	435	847	403
512	64	1164	664	1172	632

Tableau 7.2: Complexité arithmétique de l'algorithme FCMA (version fréquentielle)

C'est la première fois qu'est proposée une version rapide de cet algorithme puisque, comme nous l'avons déjà dit, les techniques classiques pour diminuer le nombre d'opérations ne fonctionnent plus, à cause de la non-linéarité de l'erreur [36].

Par ailleurs, nous pensons que les techniques du chapitre IV, pour améliorer le comportement adaptatif dans le cas du LMS, peuvent s'appliquer ici.

D'autre part, nos techniques s'étendent sans aucune difficulté à toute la classe des algorithmes de déconvolution aveugle de Godard [35].

CHAPITRE VIII

CONCLUSION GENERALE

L'objectif de cette thèse était, dans un premier temps, de diminuer le nombre d'opérations de certains algorithmes adaptatifs. Cela est devenu possible, parce que nous avons, d'une part, proposé une formulation par bloc nouvelle (équivalente à l'algorithme séquentiel et, où la taille du bloc n'est pas liée à la longueur du filtre pour la réduction de la charge de calcul) et d'autre part, fait intervenir à chaque fois la structure de la matrice de Toeplitz. Celle-ci est liée au domaine temporel puisqu'elle traduit la stationnarité d'un système, alors que la matrice circulante est liée au domaine fréquentiel puisqu'elle est diagonalisable par une matrice de Fourier. Nous avons pu passer d'une forme à une autre par une simple transformation, ce qui nous a permis d'obtenir des algorithmes rapides aussi bien dans le domaine temporel que fréquentiel, en conservant l'équivalence mathématique entre les différentes versions.

Dans un second temps, nous nous sommes servis de cette nouvelle formulation en vue de dériver des algorithmes performants au niveau du comportement adaptatif (vitesse de convergence et poursuite).

Les algorithmes obtenus ont tous un point commun: ce sont des algorithmes par bloc (N) où la taille de ce bloc est indépendante de la longueur du filtre (L). Nous avons toujours pris $N<L$, pour les raisons suivantes:

- dans la plupart des algorithmes proposés, la taille de bloc optimale (N_{opt}) au sens de la complexité arithmétique est largement inférieure à la longueur du filtre;
- plus N est grand et plus il faut de mémoires;
- plus N est grand et plus le retard à la sortie du filtre est important. Ce problème est crucial, car beaucoup d'applications ne tolérent que peu de retard;
- à partir d'une certaine taille de bloc, certains de ces algorithmes se comportent de moins en moins bien, car le filtre est mis à jour moins souvent.

Nous avons commencé cette étude par une présentation unifiée des principaux algorithmes de filtrage adaptatif. Nous avons vu que les algorithmes fréquentiels découlent des algorithmes

temporels et qu'on pouvait dériver un algorithme par bloc à partir d'un algorithme séquentiel sans aucune approximation.

Tout au long de cette thèse, nous avons proposé de nouveaux algorithmes efficaces (complexité arithmétique, comportement adaptatif). Le premier d'entre eux est une formulation exacte par bloc de l'algorithme LMS avec une réduction considérable de la charge de calcul. Cet algorithme est le FELMS qui a les mêmes performances que l'algorithme initial. D'autre part, nous avons montré que l'algorithme BLMS (déjà connu) est un cas particulier du FELMS, ce qui nous a permis d'expliquer son comportement suivant la nature du signal d'entrée.

Ensuite, nous avons montré comment utiliser des FFT de petites longueurs dans l'algorithme FELMS. L'algorithme obtenu est le FD-FELMS et contrairement aux algorithmes fréquentiels classiques, il est strictement équivalent au LMS et la dimension de la FFT n'est plus liée à la longueur du filtre mais à la taille du bloc. Ce dernier point est très important car nous avons la possibilité de contrôler le retard à la sortie du filtre par le choix de la taille de ce bloc.

Après avoir réduit la complexité arithmétique du LMS, nous nous sommes ensuite intéressé à améliorer son comportement adaptatif (vitesse de convergence et capacité de poursuite).

Nous avons dérivé, connaissant le FD-FELMS, un autre algorithme (MDF) en faisant une certaine approximation et en se servant des propriétés de la DFT. Cet algorithme va plus vite en temps de calcul et converge mieux que le FD-FELMS (donc que le LMS), mais il a l'inconvénient de poursuivre moins bien les non-stationnarités du système.

Nous avons proposé un nouveau critère à partir duquel un algorithme par bloc (NB-FELMS) a été déduit. Celui-ci, converge et poursuit mieux que le LMS et avec une complexité arithmétique toujours inférieure.

Nous avons donné un autre algorithme (SB-RLS), généralisant en quelque sorte le FELMS, qui fait le lien entre l'algorithme du gradient normalisé (NLMS) et le "Block-RLS" suivant la taille du bloc choisie. Le SB-RLS a une meilleure vitesse de convergence que le LMS.

Enfin, nous avons montré que les méthodes proposées, dans le cadre du LMS, s'étendent à d'autres types d'algorithmes adaptatifs tel que le CMA qui semble être un bon "candidat" pour l'égalisation aveugle. Ce dernier, nécessite un grand nombre d'opérations et le calcul d'une erreur non-linéaire qui le rend compliqué, en particulier, sur la possibilité d'accélérer son temps

de calcul. Nous avons proposé plusieurs versions rapides de cet algorithme qui sont, d'ailleurs, les seules existant dans la littérature.

Comme perspectives de recherches dans le prolongement de cette thèse, nous proposons les points suivants:

1) Etude théorique de la convergence de certains des algorithmes obtenus.

2) Implantation de certains de ces algorithmes sur DSP pour se rendre compte plus précisément de leur temps de calcul et pour les comparer plus objectivement.

3) Extension des résultats de cette thèse au CMA en vue d'améliorer sa vitesse de convergence et ses capacités en poursuite (une thèse à ce sujet est actuellement préparée au CNET).

4) Est-il possible de faire encore mieux que le "Fast-RLS"? Nous pouvons diminuer le nombre d'opérations du FRLS en utilisant les mêmes techniques que celles expliquées au chapitre III et dans le cas N=2. L'algorithme résultant nécessite à peu près 6,5L multiplications au lieu de 7L, mais il est très compliqué et n'a certainement aucun intérêt pratique. Son seul intérêt est de montrer que le nombre 7L n'est pas une borne comme on le croit souvent.

5) L'algorithme SB-RLS (voir chapitre IV) est un bon départ pour essayer de dériver un algorithme séquentiel du type RLS mais avec une matrice de corrélation plus petite et indépendante de la longueur du filtre.

REFERENCES

[1] A. Gilloire, J.-P. Jullien, "L'acoustique des salles dans les télécommunications", L'écho des RECHERCHES, N° 127, premier timestre 1987, pp. 43-54.

[2] G. Demoment, "Algorithmes rapides", Notes de cours, E.S.E., 1987.

[3] O. Macchi, "Le filtrage adaptatif en télécommunications", Annales des Télécommunications, vol. 36, N° 11-12, 1981, pp. 615-625.

[4] B. Widrow, S.D. Stearns, "Adaptive signal processing", Prentice Hall, Englewood Cliffs, N.J., 1985.

[5] S. Haykin, "Adaptive filter theory", Prentice Hall, Englewood Cliffs, N.J., 1986.

[6] M. Bellanger, "Traitement numérique du signal", Masson et CNET-ENST, Paris, 1980-1987.

[7] M. Bellanger, "Analyse des signaux et filtrage numérique adaptatif", Masson et CNET-ENST, Paris, 1989.

[8] B. Widrow, J.M. Maccool, "A comparison of adaptive algorithms based on the methods of steepest descent and random search", IEEE Trans. on A.P., vol. 24, N° 5, Sept. 1976, pp. 615-637.

[9] A. Gilloire, "Structure d'identification transverse. Application aux réponses acoustiques", Rapport de synthèse, GT1, GRECO, 1990.

[10] D.N. Godard, "Channel equalization using a Kalman filter for fast data transmission", IBM J. Research and Develop., May 1974, pp. 267-273.

[11] C. Gueguen, "An introduction to displacement ranks and related fast algorithms", Les

Houches, Signal Processing, session XLV, 1985.

[12] O. Macchi, M. Bellanger, "Le point sur le filtrage adaptatif transverse", Proc. of GRETSI, Nice, Juin 1987, pp. 1G-14G.

[13] G. Carayannis, D. Manolakis, N. Kalouptsidis, "A fast sequential algorithm for least-squares filtering and prediction", IEEE Trans. on ASSP, vol. 31, N° 6, Dec. 1983, pp. 1394-1402.

[14] L. Ljung, M. Morf, D. Falconer, "Fast calculation of gain matrices for recursive estimation schemes", Int. J. Control, vol. 27, N° 1, Janv. 1987.

[15] J.M. Cioffi, T. Kailath, "Fast RLS transversal filters for adaptive filtering", IEEE Trans. on ASSP, vol. 32, N° 2, April 1984, pp. 304-337.

[16] E.R. Ferrara, "Fast implementation of LMS adaptive filters", IEEE Trans. on ASSP, vol. 28, Aug. 1980, pp. 474-475.

[17] D. Mansour, A.H. Gray, Jr., "Unconstrained frequency-domain adaptive filters", IEEE Trans. on ASSP, vol. 30, Oct. 1982, pp. 726-734.

[18] G.A. Clark, S.K. Mitra, S.R. Parker, "Block implementation of adaptive digital filters", IEEE Trans. on CAS, vol. 28, June 1981, pp. 584-592.

[19] Z.J. Mou, P. Duhamel, "Fast FIR filtering: algorithms and implementation", Signal Processing, Dec. 1987, pp. 377-384.

[20] H.K. Kwan, M.T. Tsim, "High speed 1-D FIR digital filtering architectures using polynomial convolution", Proc. of ICASSP, 1987, pp. 1863-1866.

[21] M. Vetterli, "Running FIR and IIR filtering using multirate filter banks", IEEE Trans. on ASSP, vol. 36, N° 5, May 1988, pp. 730-738.

[22] Z.J. Mou, "Fast FIR filtering: algorithms and architectures", Thèse de Doctorat de l'Université d'Orsay, Sept. 1989.

[23] P. Duhamel, Z.J. Mou, J. Benesty, "Une présentation unifiée du filtrage rapide fournissant tous les intermédiaires entre traitements temporels et fréquentiels", Proc. of GRETSI, Juan-Les-Pins, Juin 1989, pp. 37-40.

[24] A. Benallal, "Etude des algorithmes des moindres carrés transversaux rapides et application à l'identification des réponses impulsionnelles acoustiques", Thèse de Doctorat de l'Université de Rennes I, Janv. 1989.

[25] D.T.M. Slock, T. Kailath, "Numerically stable fast RLS transversal filters", Proc. of ICASSP, 1988, pp. 926-929.

[26] S.S. Narayan, A.M. Peterson, M.J. Narasimha, "Transform domain LMS algorithm", IEEE Trans. on ASSP, vol. 31, N°3, June 1983, pp. 609-615.

[27] J.S. Soo, K.K. Pang, "Multidelay block frequency domain adaptive filter", IEEE Trans. on ASSP, vol. 38, N°2, Feb. 1990, pp. 373-376.

[28] P.C.W. Sommen, "On the convergence properties of a partitioned block frequency domain adaptive filter (PBFDAF)", Proc. of EUSIPCO, 1990, pp. 201-204.

[29] M. Xu, Y. Grenier, "Acoustical channel identification", Proc. of EUSIPCO, 1988, pp. 1401-1405.

[30] M. Xu, "Antenne acoustique adaptative pour la prise de son", Thèse de Doctorat de l'ENST, Dec.1988.

[31] J. Benesty, P. Duhamel, "A fast exact least mean square adaptive algorithm", Proc. of ICASSP, 1990, pp. 1457-1460.

[32] Z.J. Mou, P. Duhamel, J. Benesty, "Fast complex FIR filtering algorithms with

applications to real FIR and complex LMS filters", Proc. of EUSIPCO, 1990, pp. 549-552.

[33] R.E. Blahut, "Fast algorithms for signal processing", Addison-Wesley, Reading, MA, 1985.

[34] P. Duhamel, "Implementation of split-radix FFT algorithm for complex, real and real-symmetric data", IEEE Trans. on ASSP, vol. 34, N°2, April 1986, pp. 285-295.

[35] D.N. Godard,"Self-recovering equalization and carrier tracking in two-dimensional data communication systems", IEEE Trans. on COM, vol. 28, N°11, Nov. 1980, pp. 1867-1875.

[36] J.R. Treichler, S.L. Wood, M.G. Larimore, "Convergence rate limitations in certain frequency-domain adaptive filters", Proc. of ICASSP, 1989, pp. 960-963.

[37] J.R. Treichler, B.G. Agee, "A new approach to multipath correction of constant modulus signals", IEEE Trans. on ASSP, vol. 31, N°2, April 1983, pp. 459-471.

[38] J.R. Treichler, M.G. Larimore, "The tone capture properties of CMA-based interference suppressors", IEEE Trans. on ASSP, vol. 33, N°4, Aug. 1985, pp. 946-958.

[39] J. Benesty, P. Duhamel, "A fast constant modulus adaptive algorithm", Proc. of EUSIPCO, 1990, pp. 241-244.

[40] H.M. Zhang, "Algorithmes rapides et matrices de Toeplitz", Thèse de Doctorat de l'ENST, Sept. 1989.

ANNEXE

ARTICLE I

(à paraître dans IEEE Trans. on ASSP)

A Fast Exact Least Mean Square Adaptive Algorithm

Jacob BENESTY, Pierre DUHAMEL

CNET/PAB/RPE

38-40, rue du Général Leclerc

92131 ISSY-LES-MOULINEAUX

FRANCE

ABSTRACT

We present a general block-formulation of the LMS algorithm for adaptive filtering. This formulation has an exact equivalence with the original LMS algorithm, hence retaining the same convergence properties, while allowing a reduction in the arithmetic complexity, even for very small block lengths. Working with small block lengths is very interesting from an implementation point of view (large blocks result in large memory and large system delay) and allows nevertheless a signifiant reduction in the number of operations. Furthermore, tradeoffs between number of operations and convergence rate are obtainable, by applying certain approximations to a matrix involved in the algorithm. The usual block-LMS (BLMS) hence appears as a special case, which explains its convergence behaviour according to the type of input signal (correlated or uncorrelated).

I. INTRODUCTION

Adaptive filters are widely used in many applications, including system modeling, adaptive antennas, interference cancelling and so on [1,2,3]. Some of these applications require large FIR adaptive filters (straightforward adaptive acoustic echo cancellation would require filters of about 4000 taps), and it is therefore important to reduce the computational load of these tasks.

Usually, this has been performed by block processing, in which the filter's coefficents remain unchanged during a chosen number of samples N which is the block size. Reduction of the arithmetic complexity is obtained by making use of the redundancy between the successive computations, using the same techniques as the ones used in the fixed coefficient filtering.

Several techniques are known to reduce the arithmetic complexity of fixed coefficient FIR filtering. The usual ones are based on FFT's (most of them) or on aperiodic convolution (possibly) as an intermediate step. The main drawback of these methods is that they require large block processing: the block length N is usually equal to the filter's length, which becomes very large for the very applications where a reduction in the arithmetic complexity is required. Furthermore, these classical fast algorithms require an overlap-save sectioning, which is generally twice the filter length. Hence the transforms are applied to very long vectors, a computation which intrinsically involves non-local computation (e.g. butterflies such as $x(n)\pm x(n+N)$), a situation always difficult to manage with in actual implementations. On the contrary, the original algorithm filtering involves a multiply-accumulate operation, which is very efficiently implemented, either in software or in hardware. In some sense, the fast algorithms have lost the structural regularity of the filtering process, namely the multiply-accumulate structure. This is the reason why classical fast algorithms are efficient only for larger blocks than could be predicted from arithmetic complexity considerations. Nevertheless, a new class of fast FIR filtering algorithms taking these considerations into account was recently proposed [9,12,13]: for a given block length, which may possibly be small, these algorithms allow a reduction of the arithmetic complexity (of course, the larger the block size is, the smaller the computational complexity per output point is), while retaining partially the usual FIR filtering structure: The multiply-accumulate operation is still a basic building block of these algorithms [19].

And indeed, all the above techniques were already used in the adaptive case, from the classical ones [7,8] to the most recent ones [15] or, even, more exotic ones [18], with implementations in the time or frequency domain [6,14,24,25,26].

Nevertheless, the requirement for these techniques to be applied in an adaptive filtering scheme (i.e. the "blocking" of the coefficients during N samples) results in a different behaviour of block adaptive algorithms compared to the sample-by-sample LMS, excepted in the very special case of uncorrelated inputs. Indeed, we illustrate on an example in this paper that, in the case of a correlated input, the block-LMS (BLMS) algorithm has a smaller convergence domain than the LMS algorithm, resulting in a slower overall convergence to ensure stability (this is the reason for the well-known fact that the adaptation step must be divided by N in a BLMS algorithm [8,14]).

In this paper, we show that this drawback is removable, and that the availability of small-block processing techniques allows the derivation of computationally efficient block-adaptive algorithms which behave exactly like their scalar version. Some additional degrees of freedom exist that can even make the block algorithms converge faster than the non-block version.

This paper concentrates on the LMS algorithm, in the FIR case.

First, by working on a simple example, we prove that we can reduce the arithmetic complexity of the LMS algorithm without modifying its behaviour: the algorithm obtained is mathematically equivalent to LMS.

This is generalized in section III where it is shown that the LMS algorithm on a block of data of size N can be turned into a "fixed" filtering with some corrections, plus an "updating" part which generates the next taps. This algorithm is only a rearrangement of the initial equations of the LMS. Using the method "fast FIR filtering" for the fixed FIR part results in a so-called "Fast Exact Least Mean Square" (FELMS) algorithm, in which the total number of operations (multiplications plus additions) is reduced whatever the block size (N) may be. For example, with N=2, the number of multiplications can be reduced by about 25% with a little increase of the number of additions, and this without any approximation. Furthermore, considering larger blocks results in a "mixed radix" LMS adaptive filter with increased arithmetic efficiency, the

reduction in the number of both multiplications and additions being commanded by the block size.

On the other hand, we prove that the usual BLMS algorithm [8] is a special case of the FELMS algorithm, with an approximation on a matrix involved in the updating of the coefficients, which is the origin of the difference between the convergence rates of BLMS and LMS for correlated inputs.

Other approximations on that matrix lead to new algorithms (Fast Approximate Least Mean Square - FALMS) where the convergence rate is almost that of LMS with a number of operations still reduced compared to FELMS.

A table comparing the number of operations of the various algorithms is provided.

Finally, we show that this technique can be applied to several variations of the LMS algorithm (such as normalized LMS, sign algorithm).

All these points are illustrated by simulations of an acoustic impulse response identification.

II. AN EXAMPLE OF AN LMS ALGORITHM WITH REDUCED NUMBER OF OPERATIONS

An LMS adaptive structure has the overall organization depicted in fig.1, where:

(1)

$$\hat{y}(n) = \sum_{i=0}^{L-1} x(n-i)h_i(n) = X^t(n)H(n) = H^t(n)X(n)$$

L being the length of the filter, X(n) the observed data vector at time n:

$$X(n) = [x(n), x(n-1), \ldots\ldots, x(n-L+1)]^t$$

and H(n) the filter weight vector at time n:

$$H(n) = [h_0(n), h_1(n), \ldots\ldots, h_{L-1}(n)]^t$$

The algorithm for changing the weights of the LMS adaptive filter is given by:

(2)

a)

$$e(n) = y(n) - X^t(n)H(n)$$

b)

$$H(n+1) = H(n) + \mu e(n)X(n)$$

where μ is a parameter that controls stability and rate of convergence, and e(n) is the system error at time n.

It is well known [4] that if the adaptation constant μ is chosen such that:

$$0 < \mu < \frac{2}{LE[x^2(n)]}$$

where E[.] denotes statistical expectation, then the mean of the weight vector H(n) converges to the optimal solution of Wiener-Hopf.

Our aim in this section is to provide, for a simple example, an exact equivalent of the algorithm of eq. (2) requiring a lower number of operations per output point. It is hoped that this will be obtained by working on small data blocks, in such a way that the overall organization of the algorithm is simple. Let us consider the simplest case: Block size N=2.

Let us write equations (2) at time n-1:

(3)

$$e(n-1) = y(n-1) - X^t(n-1)H(n-1)$$

(4)

$$H(n) = H(n-1) + \mu e(n-1)X(n-1)$$

substituting (4) into (2) a), we obtain:

(5)

$$\begin{aligned} e(n) &= y(n) - X^t(n)H(n-1) - \mu e(n-1)X^t(n)X(n-1) \\ &= y(n) - X^t(n)H(n-1) - e(n-1)s(n) \end{aligned}$$

with:

$$s(n) = \mu X^t(n) X(n-1)$$

Combining eq. (3) and (5) in matrix form results in:

(6)

$$\begin{bmatrix} e(n-1) \\ e(n) \end{bmatrix} = \begin{bmatrix} y(n-1) \\ y(n) \end{bmatrix} - \begin{bmatrix} X^t(n-1) \\ X^t(n) \end{bmatrix} H(n-1) - \begin{bmatrix} 0 & 0 \\ s(n) & 0 \end{bmatrix} \begin{bmatrix} e(n-1) \\ e(n) \end{bmatrix}$$

or

$$\begin{bmatrix} 1 & 0 \\ s(n) & 1 \end{bmatrix} \begin{bmatrix} e(n-1) \\ e(n) \end{bmatrix} = \begin{bmatrix} y(n-1) \\ y(n) \end{bmatrix} - \begin{bmatrix} X^t(n-1) \\ X^t(n) \end{bmatrix} H(n-1)$$

hence:

(7)

$$\begin{bmatrix} e(n-1) \\ e(n) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -s(n) & 1 \end{bmatrix} \left[\begin{bmatrix} y(n-1) \\ y(n) \end{bmatrix} - \begin{bmatrix} X^t(n-1) \\ X^t(n) \end{bmatrix} H(n-1) \right]$$

The second term of this equation appears to be the computation of two successive outputs of a fixed coefficient filter. Thus, we can apply the same technique as explained in ref. [9]:

(8)

$$\begin{bmatrix} X^t(n-1) \\ X^t(n) \end{bmatrix} H(n-1) = \begin{bmatrix} x(n-1)\ x(n-2) \ldots\ldots x(n-L) \\ x(n)\ \ x(n-1) \ldots\ldots x(n-L+1) \end{bmatrix} \begin{bmatrix} h_0 \\ h_1 \\ \vdots \\ \vdots \\ h_{L-1} \end{bmatrix} (n-1)$$

$$= \begin{bmatrix} x(n-1)\ x(n-3) \ldots x(n-L+1) & x(n-2)\ x(n-4) \ldots x(n-L) \\ x(n)\ \ x(n-2) \ldots x(n-L+2) & x(n-1)\ x(n-3) \ldots x(n-L+1) \end{bmatrix} \begin{bmatrix} h_0 \\ h_2 \\ \vdots \\ h_{L-2} \\ h_1 \\ h_3 \\ \vdots \\ h_{L-1} \end{bmatrix} (n-1)$$

in which the even and odd numbered terms of the involved vectors have been grouped. Furthermore, in order to obtain a more compact notation, let us suppose L to be even, and

define:

$$A_0 = [x(n) \quad x(n-2) \ldots\ldots x(n-L+2)]$$
$$A_1 = [x(n-1) \; x(n-3) \ldots\ldots x(n-L+1)]$$
$$A_2 = [x(n-2) \; x(n-4) \ldots\ldots \quad x(n-L)\;]$$
$$H_0(n-1) = [h_0 \; h_2 \ldots\ldots h_{L-2}]^t(n-1)$$
$$H_1(n-1) = [h_1 \; h_3 \ldots\ldots h_{L-1}]^t(n-1)$$

Equation (8) can now be rewritten as:

(9)

$$\begin{bmatrix} X^t(n-1) \\ X^t(n) \end{bmatrix} H(n-1) = \begin{bmatrix} A_1 & A_2 \\ A_0 & A_1 \end{bmatrix} \begin{bmatrix} H_0 \\ H_1 \end{bmatrix} (n-1)$$

The same kind of work can be performed for the updating of the filter taps. First substitute (4) into (2) b):

(10)

$$H(n+1) = H(n-1) + \mu e(n) X(n) + \mu e(n-1) X(n-1)$$

or, with the above notations:

(11)

$$\begin{bmatrix} H_0 \\ H_1 \end{bmatrix} (n+1) = \begin{bmatrix} H_0 \\ H_1 \end{bmatrix} (n-1) + \mu e(n) \begin{bmatrix} A_0^t \\ A_1^t \end{bmatrix} + \mu e(n-1) \begin{bmatrix} A_1^t \\ A_2^t \end{bmatrix}$$

The following set of two equations is now seen to be the exact equivalent of the initial definition of LMS given in (2), for a block of 2 outputs:

(12)

a) $$\begin{bmatrix} e(n-1) \\ e(n) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -s(n) & 1 \end{bmatrix} \left[\begin{bmatrix} y(n-1) \\ y(n) \end{bmatrix} - \begin{bmatrix} A_1 & A_2 \\ A_0 & A_1 \end{bmatrix} \begin{bmatrix} H_0 \\ H_1 \end{bmatrix} (n-1) \right]$$

b) $$\begin{bmatrix} H_0 \\ H_1 \end{bmatrix} (n+1) = \begin{bmatrix} H_0 \\ H_1 \end{bmatrix} (n-1) + \mu \begin{bmatrix} A_1^t & A_0^t \\ A_2^t & A_1^t \end{bmatrix} \begin{bmatrix} e(n-1) \\ e(n) \end{bmatrix}$$

Now, the reduction in arithmetic complexity can take place, by rewriting (12) as:

(13)

a) $$\begin{bmatrix} e(n-1) \\ e(n) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -s(n) & 1 \end{bmatrix} \left[\begin{bmatrix} y(n-1) \\ y(n) \end{bmatrix} - \begin{bmatrix} A_1(H_0+H_1)+(A_2-A_1)H_1 \\ A_1(H_0+H_1)-(A_1-A_0)H_0 \end{bmatrix} (n-1) \right]$$

b) $$\begin{bmatrix} H_0 \\ H_1 \end{bmatrix} (n+1) = \begin{bmatrix} H_0 \\ H_1 \end{bmatrix} (n-1) + \mu \begin{bmatrix} A_1^t(e(n-1)+e(n)) - (A_1-A_0)^t e(n) \\ A_1^t(e(n-1)+e(n)) + (A_2-A_1)^t e(n-1) \end{bmatrix}$$

Considerations similar to the ones in [9] allow to evaluate precisely the number of arithmetic operations involved in (13):

The filtering operation $A_1(H_0+H_1)$ is common between the two terms of (13) a), and the overall arithmetic computation is now that of 3 length N/2 filters, instead of 4, like in eq. (12). Two of them are applied on combinations of the input samples, namely:

(14)

$$A_2 - A_1 = [x(n-2)-x(n-1),.....,x(n-L)-x(n-L+1)]$$
$$A_1 - A_0 = [x(n-1)-x(n),.....,x(n-L+1)-x(n-L+2)]$$

The apparent number of additions involved in (14) can be reduced by noticing that the previous set of scalar products (X^t(n-2)H(n-3), X^t(n-3)H(n-3)) already required nearly the same operations, which were stored in the filtering process and that only two new additions are to be computed: (x(n-2)-x(n-1)) and (x(n-1)-x(n)).

This results in the overall organization of the algorithm as depicted in fig. (2).

Note also that s(n), although being defined as a scalar product of length L, can be computed recursively from s(n-2) as:

(15)

$$s(n) = s(n-2) + \mu[x(n-1)(x(n)+x(n-2)) - x(n-L-1)(x(n-L)+x(n-L-2))]$$

The second expression in the brackets was already calculated at time n-L, thus (15) requires only one multiplication and three additions. Furthermore, if the adaptation step μ is chosen as a negative power of 2, multiplication by μ will be considered as a shift, instead of a general multiplication.

The comparison of the arithmetic complexities is now as follows:

Computation of the LMS algorithm, as expressed by (2) and (3) for two successive outputs

requires:

4L multiplications and 4L additions,

the proposed algorithm, as expressed by (13) requires:

3L+2 multiplications and 4L+8 additions

for the same computation. This means that the number of multiplications has been reduced (about 25% improvement) with only 4 additions per output more, and without any approximation in the initial equations describing the LMS algorithm.

The above approach is different from the one that has been used previously for describing block adaptive algorithms [8,15]. In the usual approach, the initial equations are rewritten in vector instead of scalar form without changing anything else. We shall see in the following that this fact has a number of consequences on the behaviour of the resulting algorithms. On the contrary, in our approach, the reduction in the number of operations is obtained only by a rearrangement of the initial equations, and there is an exact mathematical equivalence between the initial algorithm and our block version of it.

III. GENERALISATION TO ARBITRARY N

The algorithm explained above for two successive outputs can easily be generalized to an arbitrary block size N. We shall proceed in the same way as in section II: we first establish an exact block formulation of the LMS, on which a reduction of the arithmetic complexity is performed.

III.1. Exact block formulation of the LMS algorithm

Let us assume that the block length N is a factor of the length of the filter: L=NM (for a filter length that is not divisible by N, it is zero extended to the smallest multiple of N to satisfy the assumption), and let us write the LMS error equations at time n-N+1, n-N+2,....., n-1, n:

(16)

$$\begin{bmatrix} e(n-N+1) \\ e(n-N+2) \\ \vdots \\ \vdots \\ e(n-1) \\ e(n) \end{bmatrix} = \begin{bmatrix} y(n-N+1) \\ y(n-N+2) \\ \vdots \\ \vdots \\ y(n-1) \\ y(n) \end{bmatrix} - \begin{bmatrix} X^t(n-N+1) \\ X^t(n-N+2) \\ \vdots \\ \vdots \\ X^t(n-1) \\ X^t(n) \end{bmatrix} H(n-N+1) - S(n) \begin{bmatrix} e(n-N+1) \\ e(n-N+2) \\ \vdots \\ \vdots \\ e(n-1) \\ e(n) \end{bmatrix}$$

Nx1 Nx1 NxL Lx1 NxN Nx1

with matrix S defined as follows:

(17)

$$S(n) = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ s_1(n-N+2) & 0 & \cdots & 0 \\ s_2(n-N+3) & s_1(n-N+3) & & \vdots \\ \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ s_{N-1}(n) & s_{N-2}(n) & \cdots s_1(n) & 0 \end{bmatrix}$$

where:

$s_i(n) = \mu X^t(n)X(n-i)$, $i = 1,2,\ldots\ldots,N-1.$

The filter weight is then updated once per data block (instead of once per data sample) in such a way that these weights are equal to those that would be found in the initial LMS algorithm at the same time. This is performed by the following equation:

(18)

$$H(n+1) = H(n-N+1) + \mu[X(n-N+1)\ X(n-N+2)\ldots\ldots X(n)] \begin{bmatrix} e(n-N+1) \\ e(n-N+2) \\ \vdots \\ \vdots \\ e(n) \end{bmatrix}$$

Lx1 Lx1 LxN Nx1

Both equations are stated in a more convenient form as follows:

(19)

a) $\underline{e}(n) = \underline{y}(n) - \underline{X}(n)H(n-N+1) - S(n)\underline{e}(n)$

b) $H(n+1) = H(n-N+1) + \mu \underline{X}^t(n)\underline{e}(n)$

eq. (19) a) can be rewritten as:

(20)

$$[S(n)+I]\underline{e}(n) = \underline{y}(n) - \underline{X}(n)H(n-N+1)$$

$$\underline{e}(n) = [S(n)+I]^{-1}[\underline{y}(n) - \underline{X}(n)H(n-N+1)]$$
$$= G(n)[\underline{y}(n) - \underline{X}(n)H(n-N+1)]$$

Obtaining G(n) may seem intricate to compute, but, recalling that G(n) is lower triangular, it is easily seen to be obtained by:

$$G(n) = G_{N-1}(n)G_{N-2}(n).....G_1(n)$$

with:

(21)

$$G_i(n) = \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & & \vdots & \vdots \\ \vdots & \vdots & & \vdots & \\ \vdots & \vdots & & 0 & \vdots \\ -s_i(n-N+i+1) & -s_{i-1}(n-N+i+1) & \cdots & 1 & \vdots \\ 0 & & & 0 & \vdots \\ \vdots & & & \vdots & \vdots \\ \vdots & & & \vdots & 0 \\ 0 & & \cdots & 0 & 1 \end{bmatrix} \leftarrow \text{line } i+1$$

And, as a result, eq. (22) is now seen to be an exact block representation of the LMS:

(22)

a) $\underline{e}(n) = G(n)[\underline{y}(n) - \underline{X}(n)H(n-N+1)]$

b) $H(n+1) = H(n-N+1) + \mu \underline{X}^t(n)\underline{e}(n)$

Although being derived as an equivalence with an algorithm in which the filter coefficients change at each new input sample, eq. (22) a) is easily seen to contain a fixed coefficient filtering, during the period N.

Using the techniques described in [11], we can therefore apply a reduction of the arithmetic complexity of this filtering during the time its coefficients remain unchanged. This is obtained

by first performing a proper reordering of $\underline{X}$ and H:

Let us define:

(23)

$$A_j = [x(n-j)\ \ x(n-N-j)\ \ .....x(n-Ni-j).....x(n-L+N-j)]$$

a row vector (1 x L/N), for j=0,1,.....,2N-2; i=0,1,.....,(L/N)-1 and

(24)

$$H_k(n-N+1) = [h_k, h_{k+N}, ..., h_{k+Ni}, ..., h_{k+L-N}]^t(n-N+1)$$

for k=0,1,.....,N-1.

Then, $\underline{X}(n)$ expressed in terms of these polyphase components turns out to be a block Toeplitz matrix.

(25)

$$\underline{X}(n)H(n-N+1) = \begin{bmatrix} A_{N-1} & A_N \cdots\cdots A_{2N-3} & A_{2N-2} \\ A_{N-2} & A_{N-1} & A_{2N-3} \\ \vdots & & \vdots \\ \vdots & & \vdots \\ A_1 & \vdots & A_N \\ A_0 & A_1 \cdots\cdots A_{N-2} & A_{N-1} \end{bmatrix} \begin{bmatrix} H_0 \\ H_1 \\ \vdots \\ \vdots \\ H_{N-2} \\ H_{N-1} \end{bmatrix} (n-N+1)$$

This block-Toeplitz matrix can be seen as the representation of length-N FIR filtering where all the coefficients involved (x and h) are replaced by blocks. Hence, fast FIR filtering, as explained in [9] can apply. Note that, in the context of block-adaptive filtering, all the fast FIR algorithms should be used in their overlap-save version, rather than overlap-add (in other terms, following [9] they should be used in the version based on the transposition of polynomial products). This is due to the fact that any reuse of partial computations that has been performed in the previous blocks is to be avoided if it involves a filtering.

Let us now consider more precisely the computation of each term of eq. (22):

As explained for the example N=2, the computation of $S(n)=\{s_i(n)\}$ can be performed recursively as follows:

A first equation (26) provides the expression of the first column of matrix $S(s_i(n-N+i+1))$ in terms of the last row of this matrix ($s_i(n-N)$) during the previous block:

(26)

$$s_i(n-N+i+1) = s_i(n-N) + \mu[\sum_{j=0}^{i} x(n-N+i-j+1)x(n-N-j+1) - \sum_{j=0}^{i} x(n-L-N+i-j+1)x(n-L-N-j+1)]$$

for i=1,.....,N-1

and eq. (27) provides the computations to be performed along the sub-diagonals:

(27)

$$s_i(n+1) = s_i(n) + \mu[x(n+1)x(n-i+1) - x(n-L+1)x(n-i-L+1)]$$

Taken altogether, there are N(N-1)/2 such equations, requiring a total of N(N-2) multiplications and (3/2)N(N-1) additions.

Note that one multiplication can be saved in eq. (26) because x(n-N+1) appears twice. This fact has been taken into account in our operation counts.

It is also possible to show in eq. (22) that the combinations of the input samples required by the fast FIR filtering process in eq. (22) a) can be reused for the updating of the filter coefficients (22)b):

In fact, all fast FIR algorithms can be seen as a "diagonalization" of the filtering matrix as follows: If M is the number of multiplications required by the length N (short) length FIR filter, eq. (22) turns to:

(28)

a) $\underline{e}(n) = G(n)[\underline{y}(n) - A\underline{X}_d(n)(BH(n-N+1))]$

b) $H(n+1) = H(n-N+1) + \mu B^t \underline{X}_d^t(n) A^t \underline{e}(n)$

where A is an N x M block - matrix, and B an (ML/N)xN matrix, both of them involving only additions. $\underline{X}_d(n)$ is block-diagonal, each "block-coefficient" involving linear combinations of the sub-sequences defined by (23). Eq. (28) clearly shows that the update of H(n+1) involves the

same input combination $\underline{X}^t_d(n)$ as the computation of the error vector.

Let us illustrate these points on the special case N=3:

The error vector is provided by:

(29)

$$\begin{bmatrix} e(n-2) \\ e(n-1) \\ e(n) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -s_2(n) & -s_1(n) & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ -s_1(n-1) & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \left[\begin{bmatrix} y(n-2) \\ y(n-1) \\ y(n) \end{bmatrix} - \begin{bmatrix} X^t(n-2) \\ X^t(n-1) \\ X^t(n) \end{bmatrix} H(n-2) \right]$$

and the computation of the second term of (29) is performed by a length-3 fast FIR filter, as given in [19]:

(30)

$$\begin{bmatrix} X^t(n-2) \\ X^t(n-1) \\ X^t(n) \end{bmatrix} H(n-2) = \begin{bmatrix} A_2 & A_3 & A_4 \\ A_1 & A_2 & A_3 \\ A_0 & A_1 & A_2 \end{bmatrix} \begin{bmatrix} H_0 \\ H_1 \\ H_2 \end{bmatrix} (n-2)$$

$$= \begin{bmatrix} A_2(H_0+H_1+H_2)+(A_3-A_2)(H_1+H_2)+(A_4-A_3)H_2 \\ A_2(H_0+H_1+H_2)-(A_2-A_1)(H_0+H_1)+(A_3-A_2)(H_1+H_2)-[(A_3-A_2)-(A_2-A_1)]H_1 \\ A_2(H_0+H_1+H_2)-(A_2-A_1)(H_0+H_1)-(A_1-A_0)H_0 \end{bmatrix}$$

The equivalence with eq. (28) is as follows:

$$A = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \end{bmatrix}$$

$$\underline{X}_d(n) = \mathrm{diag}[A_2,\ A_3 - A_2,\ A_4 - A_3,\ A_2 - A_1,\ (A_3 - A_2) - (A_2 - A_1),\ A_1 - A_0]$$

(resulting in a 6x(6L/3) matrix)

$$B = \begin{bmatrix} I & I & I \\ O & I & I \\ O & O & I \\ -I & -I & O \\ O & -I & O \\ -I & O & O \end{bmatrix}$$

(resulting in a (6L/3)xL matrix)

I and O being identity and null matrices of size (L/3)x(L/3).

The elements $s_1(n)$ are computed recursively, as explained in the general case by eq. (26) and (27):

(31)

$$s_1(n-1) = s_1(n-3) + \mu[x(n-2)(x(n-1)+x(n-3)) - x(n-L-2)(x(n-L-1)+x(n-L-3))]$$

$$s_1(n) = s_1(n-1) + \mu[x(n)x(n-1) - x(n-L)x(n-L-1)]$$

$$s_2(n) = s_2(n-3) + \mu[x(n-2)(x(n)+x(n-4)) + x(n-1)x(n-3) - x(n-L-2)(x(n-L)+x(n-L-4)) - x(n-L-1)x(n-L-3)]$$

Now, we adjust the weight vector as:

(32)

$$\begin{bmatrix} H_0 \\ H_1 \\ H_2 \end{bmatrix}(n+1) = \begin{bmatrix} H_0 \\ H_1 \\ H_2 \end{bmatrix}(n-2) + \mu \begin{bmatrix} A_2^t & A_1^t & A_0^t \\ A_3^t & A_2^t & A_1^t \\ A_4^t & A_3^t & A_2^t \end{bmatrix} \begin{bmatrix} e(n-2) \\ e(n-1) \\ e(n) \end{bmatrix}$$

$$= \begin{bmatrix} H_0 \\ H_1 \\ H_2 \end{bmatrix}(n-2) + \mu B^t \underline{X}_d^t(n) A^t \begin{bmatrix} e(n-2) \\ e(n-1) \\ e(n) \end{bmatrix}$$

$$= \begin{bmatrix} H_0 \\ H_1 \\ H_2 \end{bmatrix}(n-2) + \mu \begin{bmatrix} A_2^t(e(n)+e(n-1)+e(n-2)) - (A_2-A_1)^t(e(n)+e(n-1)) - (A_1-A_0)^t e(n) \\ A_2^t(e(n)+e(n-1)+e(n-2)) - (A_2-A_1)^t(e(n)+e(n-1)) + (A_3-A_2)^t(e(n-1)+e(n-2)) - ((A_3-A_2)^t-(A_2-A_1)^t)e(n-1) \\ A_2^t(e(n)+e(n-1)+e(n-2)) + (A_3-A_2)^t(e(n-1)+e(n-2)) + (A_4-A_3)^t e(n-2) \end{bmatrix}$$

This algorithm requires, for three successive iterations 4L+7 multiplications and 5L+23 additions instead of 6L multiplications and 6L additions for the LMS algorithm.

Note that, compared to the case N=2, working with 3 outputs at a time is now seen to be more efficient: the total number of operations (multiplications plus additions) has been reduced

by 25%.

The main result of this section is that, provided that some caution is taken (the S matrix of eq. (17)), nearly any fast FIR filtering algorithm can be used in an LMS algorithm to reduce the arithmetic complexity, without modifying its convergence properties, since there is an exact equivalence between both versions of the algorithm.

III.2. Two special cases of interest

III.2.1. $N=2^n$

This is an important case, since simple and efficient fixed-coefficient fast FIR filtering algorithms are known for this type of length. A significant reduction of the arithmetic complexity is thus feasible, even with small to moderate block lengths. Straightforward application of the results of section III.1. results in the following operation counts for a block of $N=2^n$ outputs of a filter of length $L=NM$.

Total number of operations:

(33)

$2\,(3/2)^n\,L + 2^n\,(3\,2^n - 5)/2 + 1$ multiplications

(34)

$2\,(2\,(3/2)^n - 1)\,L + 2^{n+1}\,(2^n - 3) + 4\,3^n$ additions,

or, if we consider the number of operations per output:

(35)

$2\,(3/2)^n\,M + (3\,2^n - 5)/2 + 1/2^n$ multiplications

(36)

$2\,(2\,(3/2)^n - 1)\,M + 4\,(3/2)^n + 2\,(2^n - 3)$ additions,

to be compared with 2L multiplications and 2L additions per output required by the LMS algorithm.

It is easily verified that, as long as $M \geq 2$, the FELMS algorithm requires fewer operations

than the LMS algorithm. Note also that the second term of (35) and (36) involves $N=2^n$, instead of NM in the LMS. This term (2^n) is due to the computation of matrix S(n). This is the point where the availability of small block-processing made this approach feasible: In fact, if N was of the same order of magnitude as L, FELMS would require even more operations than LMS. The important point now is that arithmetic complexities involve a term growing with N (updating of S), and another one diminishing with N (fast FIR). Hence, eq. (35) has a minimum: the zero of the derivative of (35) results in the following "optimum" which provides the least number of multiplications:

(37)

$$n_{opt} \approx -0.6+0.7\log_2 L$$

Table 1 provides the number of operations required by both LMS and FELMS for various filter lengths, and a blocksize of the order of the one given by (37). It is seen that an adaptive filter of length 128 can be implemented with half as many multiplications per point than the LMS algorithm, with a block length of only 16. A reduction by a factor near 4 is obtained for a filter of length 1024, and a block length equal to 64. Just like in the case of fixed filtering, the choice of the block size for a given filter length depends on the type of device on which this algorithm is implemented. Eq. (37) provides the evaluation of the block length that minimizes the number of multiplications. Other criteria can be considered: e.g.: minimization of the number of multiply-accumulates, and so on. It is shown in [19] that the knowledge of a whole family of fast FIR algorithms allows to choose the precise one that suits best the chosen architecture. This paper provides the tools allowing the same kind of result in the case of the "fast LMS filtering". A precise treatment of this point is out of the scope of this paper. Nevertheless, it is our opinion that the most interesting algorithms always use block lengths smaller than the filter's length, since they allow a good tradeoff between structural regularity (they keep partly the multiply-accumulate structure) and arithmetic complexity, while keeping the same convergence rate as the LMS (for any input signal).

III.2.2. FFT- based implementations

One of the possible fixed-coefficient FIR schemes uses the FFT as an intermediate step [12, 10], in which case matrices A and B in eq. (28) turn out to be parts of Fourier matrices. Note

that, in this case, the Fourier transform is used only for "block-diagonalizing" the block Toeplitz matrix of eq. (25), and that the usual constraint of the size of the FFT being twice the filter's length does not hold. With the notations of this paper, the length of the FFT is twice the block size.

The important point is that some of the properties of FFT are known to be useful in the context of block-processing [15,16,17]. In particular, the possibility of increased speed of convergence by using different adaptation steps on the Fourier coefficients of the impulse response of the filter is usually thought to be linked with the orthogonality property of the DFT [21]. Hence, these possibilities of increased rate of convergence should be kept in the implementation proposed in this section, by using different adaption steps for each subfilter (there are several possibilities for this task and a paper is in preparation).

III.3. Simulations

The above algorithms have been programmed for an acoustic impulse response identification, in a scheme as depicted in fig. (3).

The system to be identified is described by an impulse response measured in a real room (500 ms duration, sampled to 16000 Hz). The model is an FIR filter of length 1024.

Fig. (4) provides the error curves of several algorithms in the case of white noise input, and fig. (5), (6), (7) in the case of USASI noise input (USASI noise is a correlated noise with the same spectrum as speech).

For the purpose of smoothing the curves, error samples are averaged over 128 points. Furthermore, they are normalized by the corresponding input energy.

III.3.1. FELMS versus LMS

Curves of fig. (5) a) and (7) a) refer to both LMS and FELMS: In both cases of simulations, they exactly superimpose. Since they have an exact mathematical equivalence, this is not astonishing. Nevertheless, this gives an indication on the accuracy issue of FELMS: the updating of coefficients $s_i(n)$ being performed recursively, one may wonder if this computation

could introduce any major drawback. Curves of fig. (5) a) and 7) a) show that, after 9000 iterations, for a simple precision (32 bits) floating point implementation, the two curves could not be distinguished. It is further shown in Appendix A that, in the case of fixed point implementations, this way of computing only result in a slight increase of the residual error. It is also shown that, in any realistic case, these errors cannot result in an instability of the algorithm.

III.3.2. FELMS versus BLMS

As seen above, the FELMS algorithm is derived directly by a rearrangement of the LMS algorithm, and, although the data are process blockwisely, there is a strict equivalence between them, when speaking of their adaptive characteristics. On another side, the classical block adaptive LMS algorithm (BLMS) is usually derived by re-writing eq.(2) in vector form. This algorithm supposes that the filter remains fixed during N samples, and is updated once per block, in a manner minimizing some "block criterion" [24].

Classical BLMS algorithm obtain a reduction in arithmetic complexity by using large block (equal to the filter's length), and we shall not consider them here. The use of recent fast-FIR filtering algorithms [19] allows the derivation of small-block BLMS algorithm, which was proposed in [15].

Table 1 provides a comparison of the arithmetic complexity of FELMS and BLMS algorithms. The numbers for the BLMS algorithm [15] are updates of the ones in [15] where the number of additions has been reduced by sharing some of them between the filtering and the weight update, in a manner similar to what has been explained in section 2.

In what follows, arithmetic complexity comparisons are based on the total number of operations (multiplications plus additions), which is a stronger criterion than just the number of multiplications. For a length 1024 filter, the LMS algorithm requires 200% more operations than the FELMS algorithm (that is three times more operations), while the BLMS requires 16% less operations than the FELMS algorithm.

The question now is about their relative convergence:

Simulations were first performed with the same adaptation step μ for both algorithms (the same μ as used in the LMS). Note that, compared to the adaptation step recommended in the classical paper on BLMS, this μ is N times larger.

In the uncorrelated case, the convergence curves were almost identical for any block lengths from N=2 to N=128, exhibiting only tiny differences: fig. (4) b) depicts the first BLMS curve that can visually be distinguished from the LMS-FELMS case (curves for FELMS are of course always identical to the one of fig. (4) a) whatever the block size N may be).

In the case of USASI noise input, the situation is much different: fig. (6) provides the error curve of BLMS with the same step μ as LMS and FELMS algorithm (fig. (5) a)): Divergence is seen to occur quickly. The solution to this problem is well-known : the adaptation step in BLMS should be N times smaller than in LMS in order to ensure stability. In this case, the algorithm converges, as shown in fig. (5) b). Nevertheless, it is seen that the convergence is now slower than in the LMS case, this drawback being stronger for increasing N (see the N=16 curve of fig. (5) c)).

From these simulations, it is clearly seen that BLMS performs best for uncorrelated inputs, in which case it is equivalent to LMS provided that the adaptation step is chosen N times larger than usually. Correlated inputs result in a reduced convergence region. The "better" convergence of BLMS reported elsewhere [7] in the case of frequency domain implementation seems to be due only to the use of a different adaptation step on each coefficient, which is also feasible for the proposed algorithm (see section III.2.2.).

This has to be compared with FELMS which always has the same error curve as LMS, whatever the input signal and block size may be. Eq. (22) is used in the next section to explain precisely why this behaviour of BLMS occurs.

IV. TRADING CONVERGENCE SPEED FOR ARITHMETIC COMPLEXITY

In fact, (22) can be seen to be the same set of equations as the definition of BLMS [8,15], but for the term G(n) : Taking G(n)=I, the identity matrix, turns FELMS to BLMS, together with an eventual change of μ. The role of this matrix (or matrix S(n) in eq.(19)) is thus seen to be crucial in these algorithms.

IV.1. Interpretation of matrix S

A straightforward calculation shows that, under the following conditions:

- the input signal is ergodic

- the filter's length L is large enough to provide acceptable average of the statistics of the signal.

We have as a result:

(38)

$$s_1(n) = \mu \sum_{i=0}^{L-1} x(n-i)x(n-1-i) \rightarrow \mu L E[x(n)x(n-1)] = \mu L r(1)$$

$$\vdots$$

$$\vdots$$

$$s_{N-1}(n) = \mu \sum_{i=0}^{L-1} x(n-i)x(n-N+1-i) \rightarrow \mu L E[x(n)x(n-N+1)] = \mu L r(N-1)$$

where r is the autocorrelation function of the input signal. Hence, S converges to:

(39)

$$S_A = \mu L \begin{bmatrix} 0 & 0 \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots 0 \\ r(1) & 0 \quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \vdots \\ r(2) & r(1) \quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \vdots \\ \vdots & r(2) \quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \vdots \\ \vdots & \vdots \quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \vdots \\ \vdots & \vdots \quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \vdots \\ r(N-1) & r(N-2)\cdots\cdots\cdots\cdots r(2) \quad\quad r(1) \quad\quad 0 \end{bmatrix}$$

Note that since:

$$0 < \mu < 2 / L\, r(0)$$

and:

$$r(0) \rangle |r(i)| \quad \forall\, i \neq 0$$

we obtain:

$$\mu L |r(i)| \langle 2 \quad \forall\, i \neq 0$$

which gives an indication on the range of the s_i.

IV.2. Various approximations on S:

IV.2.1. BLMS

We have already seen that BLMS can be seen as an FELMS where the S-matrix has been taken equal to zero. The results of section IV.1. explain why this does make sense whenever the input is uncorrelated, since in this case the autocorrelation coefficients are equal to zero.

Nevertheless, when the input is correlated, this may be a drastic approximation: fig. (6) shows that, when used with the same adaptation step as the initial LMS, BLMS diverges soon, even for such small blocks as N=4.

IV.2.2. FALMS

Hence, the difference between FELMS and BLMS is seen to depend only on the statistics of the input signal. In many cases, if N is large enough, it will not be necessary to take into account the high order correlations of the input signal. It is therefore reasonable to keep only a few sub-diagonals in matrix S, up to the point where the correlation coefficients are known to be small enough. This will result in a Fast Approximate Least Mean Square (FALMS) algorithm for which we hope to keep the best of both FELMS (larger convergence domain) and BLMS (low arithmetic complexity). Table 1 provides the corresponding number of operations, in the case where $\log_2 N$ subdiagonals are kept. Note that this number of sub-diagonals should not be used as such: The number to be kept clearly depends on the correlation of the input signal. This rule-of-thumb was used in our example to provide a precise evaluation of the number of operations.

We have simulated this so-called FALMS algorithm under the same conditions as previously. Using USASI noise input, $\log_2 N$ subdiagonals were enough to make FALMS behave much like LMS: fig. (7) a) gives the adaptation curve of FELMS for N=64, which is exactly the same one as would have been provided by the LMS algorithm. Fig. (7) b) provides the error curve of FALMS for the same blocklength, and the same adaptation step μ, but using only six subdiagonals. Both curves are seen to be quite similar. The error curve of BLMS, with an adaptation step ensuring stability would be completely out of scale, and has not been plotted. Note that the improvement brought by the FALMS over the BLMS is obtained at a very low

cost: a total of 28 operations for a length 1024 filter (see table 1).

Let us point out, however, that the main motivation for using an approximate S-matrix is not so much with arithmetic complexity, since it is seen in table 1 that FELMS, FALMS, and BLMS require comparable number of operations. Main advantages should be lower memory requirements and easier organization of the resulting program.

Table 1, together with fig.7 shows that FALMS is a good tradeoff between algorithm complexity and speed of convergence.

Let us also point out that in the simulations of fig.7 BLMS saved 60% computation time compared with LMS, but with a slower convergence, while FELMS saved 50%, with exactly the same behaviour as LMS. These timings should not be taken as definitive ratios, the best implementation of FELMS is a subject of further study.

Another remark is that FELMS keeps another advantage of BLMS: the filter weights are updated once per data block, which reduces the amount of data flow. This is useful in DSP or VLSI implementations.

IV.2.3. Blocking the S matrix

An important practical situation is the case when one knows that the input is stationary, but one does not know its statistics.

FELMS can be used to obtain the signal autocorrelation, and when the values of S are stabilized, the recursions on the elements s_i can be stopped, and the remainder of the algorithm uses the obtained values.

Fig.7-c provides such a simulation where the s_i were estimated during 2048 samples (twice the filter's length) using the FELMS equation (19) and then blocked to the obtained values. The resulting error curve is seen to behave much like the initial LMS.

V. VARIATIONS OF THE LMS ALGORITHM

Our purpose in this section is to show that, although derived for the straightforward LMS algorithm, the technique described above is of more general application. As examples, we have chosen the normalized LMS and the sign algorithms. Of course, we do not fully derive the algorithms, but only provide their block formulation, on which the fast FIR algorithms can easily be applied.

V.1. The normalized LMS algorithm

In this algorithm, the adaptation step is normalized by the input energy, and this results in a better convergence rate compared with the initial LMS. The normalized LMS equations are:

(40)

$$H(n+1) = H(n) + \mu(n)X(n)e(n)$$

$$e(n) = y(n) - X^t(n)H(n)$$

$$\mu(n) = \frac{\alpha}{X^t(n)X(n)}, \quad \text{with} \quad 0 < \alpha < 2$$

And, using the techniques described above, it is possible to obtain an exact block formulation of the normalized LMS:

(41)

$$\begin{bmatrix} e(n-N+1) \\ e(n-N+2) \\ \vdots \\ \vdots \\ e(n-1) \\ e(n) \end{bmatrix} = \begin{bmatrix} y(n-N+1) \\ y(n-N+2) \\ \vdots \\ \vdots \\ y(n-1) \\ y(n) \end{bmatrix} - \begin{bmatrix} X^t(n-N+1) \\ X^t(n-N+2) \\ \vdots \\ \vdots \\ X^t(n-1) \\ X^t(n) \end{bmatrix} H(n-N+1) - S(n) \begin{bmatrix} \bar{e}(n-N+1) \\ \bar{e}(n-N+2) \\ \vdots \\ \vdots \\ \bar{e}(n-1) \\ \bar{e}(n) \end{bmatrix}$$

S(n) being defined as in eq.(17), but with a different definition of $s_i(n)$:

$$s_i(n) = X^t(n)X(n-i)$$

$\bar{e}(n)$ and $\mu(n)$ are defined as follows :

(42)

$$\bar{e}(n) = \mu(n)e(n)$$

$$\mu(n) = \frac{\alpha}{X^t(n)\,X(n)} = \frac{\alpha}{s_0(n)}$$

The block-adaptation is as follows:

(43)

$$H(n+1) = H(n-N+1) + [X(n-N+1) \ldots\ldots\ldots\ldots X(n)] \begin{bmatrix} \bar{e}(n-N+1) \\ \vdots \\ \vdots \\ \vdots \\ \bar{e}(n) \end{bmatrix}$$

Note that a slight change of notation has been necessary, compared to the LMS, (μ is no more included in the S matrix) , in order to allow the same techniques to be applied: recursive computation of $s_i(n)$ and fast FIR computation, that can be partially reused in the updating of H.

V.2. The sign algorithm

The purpose of this algorithm is not to increase convergence rate, but to reduce the number of operations to be performed for updating the coefficients:

(44)

$$H(n+1) = H(n) + \mu \, \mathrm{sgn}(e(n))X(n)$$

$$e(n) = y(n) - X^t(n)H(n)$$

where :

$$\mathrm{sgn}(\theta) = +1 \text{ if } \theta \geq 0$$

$$= -1 \text{ if } \theta < 0$$

The block formulation of the algorithm is performed exactly in the same manner as for the LMS, with e(n) replaced by sgn(e(n)) in eq. (22).

Note that the resulting fast algorithm involves some terms of the form:

$$\sum \mathrm{sgn}(e(n))$$

which are no more equal to ± 1. A multiplication by such a quantity is nevertheless much simpler for small N than a general multiplication.

Other variations of the LMS, such as delayed LMS, can be treated by the same techniques

In fact, all these techniques are of much more general application than even the variations of

LMS, since we were able to derive an exact block formulation of the Constant Modulus Algorithm (CMA),[17] which was thought to be impossible by the usual techniques. This work will be reported elsewhere [23].

VI. CONCLUSION

Several algorithms were already known to reduce the arithmetic complexity of the LMS adaptive filter, most of them relying on a block formulation known as BLMS. This formulation involves a filter with fixed coefficients during N samples (N the block size), which allows various techniques (FFT, fast FIR) to be applied for reducing the arithmetic complexity.

Nevertheless, this block formulation is not an exact equivalent of the LMS algorithm in any circumstances: it is well known that the adaptation step must be N times smaller than in the LMS algorithm to ensure stability, which reduces its convergence rate.

In this paper, we first showed through an example that the LMS algorithm can be speeded up without changing any of its adaptive characteristics (section 2). Then, we showed that this was obtained by the use of another block formulation, which allows block-LMS algorithms to be exactly equivalent to the sample by sample LMS. Since this block formulation also involves a fixed coefficient filtering, most techniques that are known to reduce the arithmetic complexity of a filtering can be applied here, resulting in a so-called Fast Exact LMS algorithm. We detailed the case of the fast-FIR technique, and outlined the FFT case.

The understanding of the differences between the usual block-formulation and ours also allowed to explain precisely the origin of the differences between LMS and BLMS algorithms: We showed that different approximations on a matrix led to a whole family of algorithms, the BLMS being one of its members.

Although explained in the special case of the LMS algorithm, these techniques can be applied to many different algorithms in the LMS family. We provided an outline of their application to two variations of the LMS. In a work reported elsewhere [23], we applied the basic technique described herein to derive an efficient version of the Constant Modulus Algorithm.

It is our strong belief that the essential of this paper relies not only in the algorithm itself, but also in the block formulation, that opens new ways towards simple LMS-type algorithms with low arithmetic complexity altogether with fast convergence rate. This work is under the process of being written, and will be reported.

AKNOWLEDGMENTS

The authors would like to thank M. Vetterli, A. Gilloire, and Z.J. Mou for useful discussions on the subject of this paper. Simulation data were kindly provided by A. Gilloire. We also wish to thank the reviewers for their helpful comments and suggestions in improving the earlier version of the paper.

APPENDIX A Errors due to finite precision arithmetic

The theoretical analysis of the errors due to finite precision arithmetic in an adaptive algorithm is usually based on the assumption that the input samples are zero mean uncorrelated white Gaussian random variables. Nevertheless, in this case, the "autocorrelation" matrix S is zero, and there is no difference between FELMS and BLMS algorithms. So, a detailed analysis of FELMS is very difficult, and we shall only provide in the following some indications on the behaviour of FELMS under fixed point arithmetic.

Throughout this appendix, we use unprimed and primed symbols to represent quantities of infinite and finite precision, respectively. Another assumption is that a scalar product is computed with full precision, and quantized afterwards. This corresponds to the kind of implementation that is found in many digital signal processors.

We address separately the problem of computing the updating of the filter taps and that of the recursive computation of s_i. Another limitation of our analysis is that we consider only the influence of finite precision on the adaptive aspects, and assume that the fast FIR technique is applied in such a way that the resulting accuracy is comparable with that obtained in usual implementations. Preliminary results show that this is obtained by using $(1/2)\log_2 N$ additional bits in the fast FIR computation [22].

Let

σ_x^2

be the input signal power, and

$\sigma_e^2(n)$

the error power at time n. Based on [20], the condition for the i^{th} component of the weight vector not to be updated in the LMS algorithm is:

(A1)

$$\left|\mu x(n-i)e'(n)\right| < 2^{-B_{LMS}-1}$$

where B_{LMS} is the number of bits used for representing the coefficients. Squaring both sides of (A1), and with the assumption that n is large enough, so that the filter is near convergence, [20]

shows that a reasonable approximation of (A1) is:

(A2)

$$\mu^2 \sigma_x^2 \sigma_{e_{LMS}}^2(n) < \frac{2^{-2B_{LMS}}}{4}$$

Hence:

(A3)

$$B_{LMS} < -\frac{1}{2}[\log_2(\mu^2 \sigma_x^2) + \log_2(\sigma_{e_{LMS}}^2(n)) + 2]$$

Concerning the FELMS algorithm, the same kind of calculation holds: (A4) is the condition for the i^{th} component of the weight vector not to be updated in the FELMS algorithm:

(A4)

$$\left| \mu \sum_{j=0}^{N-1} x(n-j-i)\, e'(n-j) \right| < 2^{-B_{FELMS}-1}$$

which gives:

(A5)

$$\mu^2 N \sigma_x^2 \sigma_{e_{FELMS}}^2(n) < \frac{2^{-2B_{FELMS}}}{4}$$

Hence:

(A6)

$$B_{FELMS} < -\frac{1}{2}[\log_2(\mu^2 \sigma_x^2) + \log_2(\sigma_{e_{FELMS}}^2(n)) + \log_2(N) + 2]$$

A scaling by 1/N in (A4) has not been taken into account, the multiplication by μ (which is very small usually) playing more than necessary this role.

And, if both algorithms converge, we have as a result:

(A7)

$$B_{FELMS} + \frac{1}{2}\log_2(N) \geq B_{LMS}$$

It is seen that, from the strict point of view of adaptation, FELMS would require (1/2) $\log_2 N$ bits less than LMS.

Nevertheless, the main problem in FELMS is certainly the recursive computation of $s_i(n)$,

and a necessary condition for stability of FELMS is the stability of s_i, because their computation is independent of the remainder of the algorithm.

Let

$$d'_i(n) = s'_i(n) - s_i(n)$$

where d'_i is the error between the fixed-point estimate s'_i and the true (infinite precision) s_i.

A straightforward computation shows that:

(A8)

$$\sigma^2_{d'_i}(n) = \frac{n(N-i)}{N}\sigma^2_\beta$$

where $\sigma^2_\beta = \frac{2^{-2b}}{12}$,

b being the number of bits used for the computation of s_i.

The worst case is seen to occur for s_1, in which case:

(A9)

$$\sigma^2_{d'_1}(n) \approx n\,\sigma^2_\beta$$

We have seen in section IV.1. that:

$$s_1(n) \approx \mu L r(1)$$

for large n (r being the autocorrelation function).

Furthermore, it is well known that:

(A10)

$$|r(i)| \le r(0) \quad \forall i \ge 0$$

So, we can write the condition such that, at time t, the time of convergence of the algorithm, the recursive computation of $s_i(n)$ will meet inequality (A10):

(A11)

$$|s'_1(t)| \le \mu L \sigma^2_x$$

which can be rewritten as:

(A12)

$$|s'_1(t) - s_1(t)| \le 2\mu L \sigma^2_x$$

In summary, the (few) results we have obtained for the analysis of FELMS under finite precision arithmetic are:

- from the strict point of view of the updating of the coefficients, FELMS should require $(1/2)\log_2 N$ bits less than LMS for representing the coefficients,

- from the point of view of the quantization noise at the output of the filter, FELMS should require $(1/2)\log_2 N$ bits more than LMS for representing the data.

- updating the s_i with the same number of bits as that used in the LMS never results in unrealistic values of the correlation coefficients, even if the recursive computation is performed up to the time of convergence. Furthermore, we have seen (fig. 6.e) that this recursive computation could be stopped much earlier in the case of stationary inputs without modifying the convergence of the algorithm.

Note that, as usual, all these results are asymptotic.

As a conclusion, we think that FELMS can be implemented with same number of bits as LMS, FELMS exhibiting possibly an increased residual error.

Note also that these finite precision problems should always remain manageable, since our approach requires the use of very small block lengths: Implementing efficiently a filter length $2^{10}=1024$ by our algorithm requires only a block size equal to $2^6=64$.

REFERENCES:

[1] S. Haykin, "Adaptive filter theory", Prentice Hall, Englewood cliffs, N.J., 1986.

[2] B. Widrow, J.M. McCool, "A comparison of adaptive algorithms based on the methods of steepest descent and random search", IEEE Trans. Antennas Propagat., vol. AP-24, pp. 616 - 637, Sept. 1976.

[3] O. Macchi, "Le filtrage adapatatif en Télécommunications", Annales des Télécommunications, vol. 36, n° 11-12, 1981.

[4] B. Widrow, J.M. McCool, "Stationary and nonstationary learning characteristics of the LMS adaptive filter", Proc. IEEE, vol. 64, pp. 1151 - 1162, Aug. 1976.

[5] M.G. Bellanger, "Adaptive digital filters and signal analysis", Marcel Dekker Inc., New-York, 1987.

[6] E.R. Ferrara, "Fast implementation of LMS adaptive filters", IEEE Trans. Acoust., Speech and Signal processing, vol. ASSP-28, pp.474 - 475, Aug. 1980.

[7] D. Mansour, A.H. Gray, Jr., "Unconstrained frequency-domain adaptive filter", IEEE Trans. Acoust., Speech, Signal Processing, vol. ASSP-30, PP. 726 - 734, Oct. 1982.

[8] G.A. Clark, S.K. Mitra, S.R. Parker, "Block implementation of adaptive digital filters", IEEE Trans. on CAS., vol. 28, pp. 584 - 592, June 1981.

[9] Z.J. Mou, P. Duhamel, "Fast FIR filtering : algorithms and implementation", Signal Processing, pp. 377 - 384, Dec. 1987.

[10] Z.J. Mou, P. Duhamel, "A unified approach to the fast FIR filtering algorithms", Proc. ICASSP' 88, pp. 1914-1917.

[11] P. Duhamel, Z.J. Mou, J. Benesty, "Une présentation unifiée du filtrage rapide fournissant

tous les intermédiaires entre traitements temporels et fréquentiels", 12th GRETSI Conf., Juan-les-Pins, pp. 37 - 40, June 1989.

[12] H.K. Kwan, M.T. Tsim, "High speed 1-D FIR digital filtering architecture using polynomial convolution", Proc. ICASSP' 87, pp. 1863 - 1866.

[13] M. Vetterli, "Running FIR and IIR filtering using multirate filter banks", IEEE Trans. on ASSP, Vol. 36, n°5, pp. 730 - 738, May 1988.

[14] G.A. Clark, S.R. Parker, and S.K. Mitra, "A unified approach to time and frequency domain realization of FIR adaptive digital filters", IEEE Trans. on ASSP, vol. 31, n°5, pp. 1073 - 1083, Oct. 1983.

[15] A.O. Ogunfunmi, A.M. Peterson, "Fast direct implementation of block adaptive FIR filtering", Proc. ICASSP-1989, pp. 920 - 923.

[16] J.C. Lee, C.K. Un , "Block realization of multirate Adaptive Digital Filters", IEEE Trans. on ASSP , vol. 34, n°1, pp. 105 - 117, Feb. 1986.

[17] J.R. Treichler, S.L. Wood, M.G. Larimore, "Convergence Rate limitations in certain frequency-domain adaptive filters", Proc. ICASSP'89, pp. 960 - 963.

[18] G. Panda, P.M. Grant, "Rectangular Transform band adaptive filter", Electronics Letters, Vol. 21, n°7, pp. 301 - 303, March 1985.

[19] Z.J. Mou, P. Duhamel , "Short-length FIR filters and their use in fast FIR filtering" to appear, IEEE Trans. on ASSP, June 1990.

[20] C. Caraiscos, B. Liu , "A roundoff error analysis of the LMS adaptive algorithm", IEEE Trans. on ASSP, vol. 32, n°1, pp. 34 - 41, Feb. 1984.

[21] J.C. Lee, C.K. Un, "Performance of transform-domain LMS adaptive digital filters",

IEEE Trans. on ASSP, vol. 34, pp. 499 - 510, June 1986.

[22] Z.J. Mou, "Fast FIR filtering : algorithms and architectures", Doctorate thesis, University of Paris Sud, Sept. 1989.

[23] J. Benesty, P. Duhamel, "A fast constant modulus adaptive algorithm", Proc. EUSIPCO-1990, pp. 241 - 244.

[24] G.A. Clark, S.K. Mitra, S.R. Parker, "Block adaptive filtering", Proc. ISCAS-1980, pp. 384 - 387.

[25] T. Walzman, M. Schwartz, "Automatic equalization using the discrete frequency domain", IEEE Trans. on I.T., vol. 19, pp. 59 - 68, Janv. 1973.

[26] M.J. Dentino, J.M. McCool, B. Widrow, "Adaptive filtering in the frequency domain", Proc. IEEE, vol. 66, pp. 1658 - 1659, Dec. 1978.

Figure captions

Figure 1: LMS adaptive structure

Figure 2: An FELMS structure for block length N=2

Figure 3: Adaptive structure used for system identification

Figure 4: The error curves of LMS and BLMS algorithms when the input signal is a white noise and with the same adaptation step:
a) LMS algorithm
b) BLMS algorithm with a block size N=128

Figure 5: The error curves of LMS and BLMS algorithms in the case of USASI noise input, μ being chosen to insure convergence:
a) LMS algorithm
b) BLMS algorithm with N=4. μ is divided by N
c) BLMS algorithm when N=16 and with μ divided by 16

Figure 6: Error curve of BLMS, N=4, in the case of USASI noise input, the adaptation step is the same one as in the LMS case

Figure 7: Error curves of the proposed algorithms in the case of USASI noise and with the same μ than LMS. The block size N is equal to 64:
a) FELMS algorithm
b) First approximation (computation of matrix S only with 6 subdiagonals instead of 63)
c) Second approximation (blocking of the S matrix after 2048 samples)

Table caption

Table 1: Comparison of the number of operations per output point required by the various algorithms

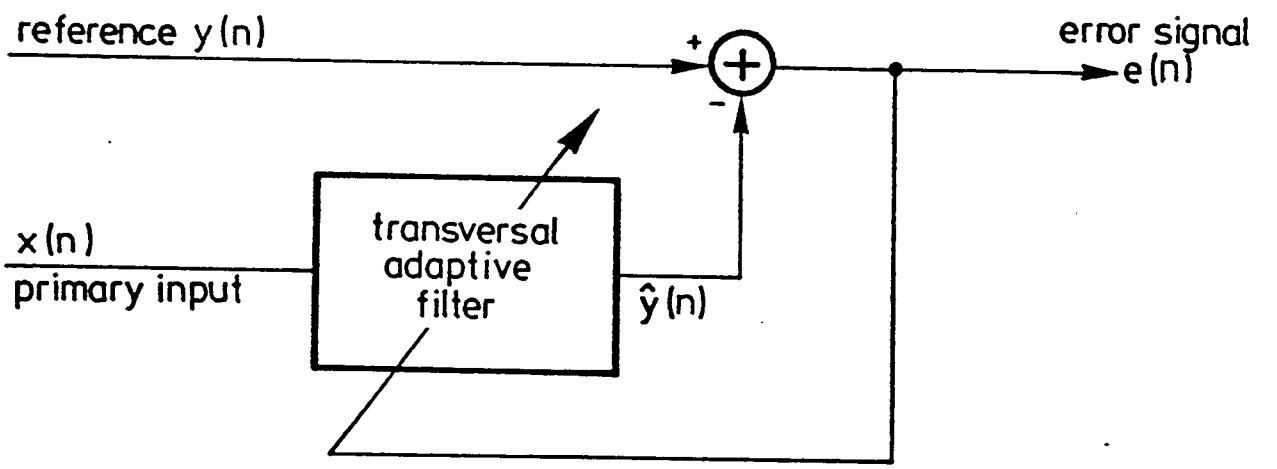


Fig.1: LMS adaptive structure

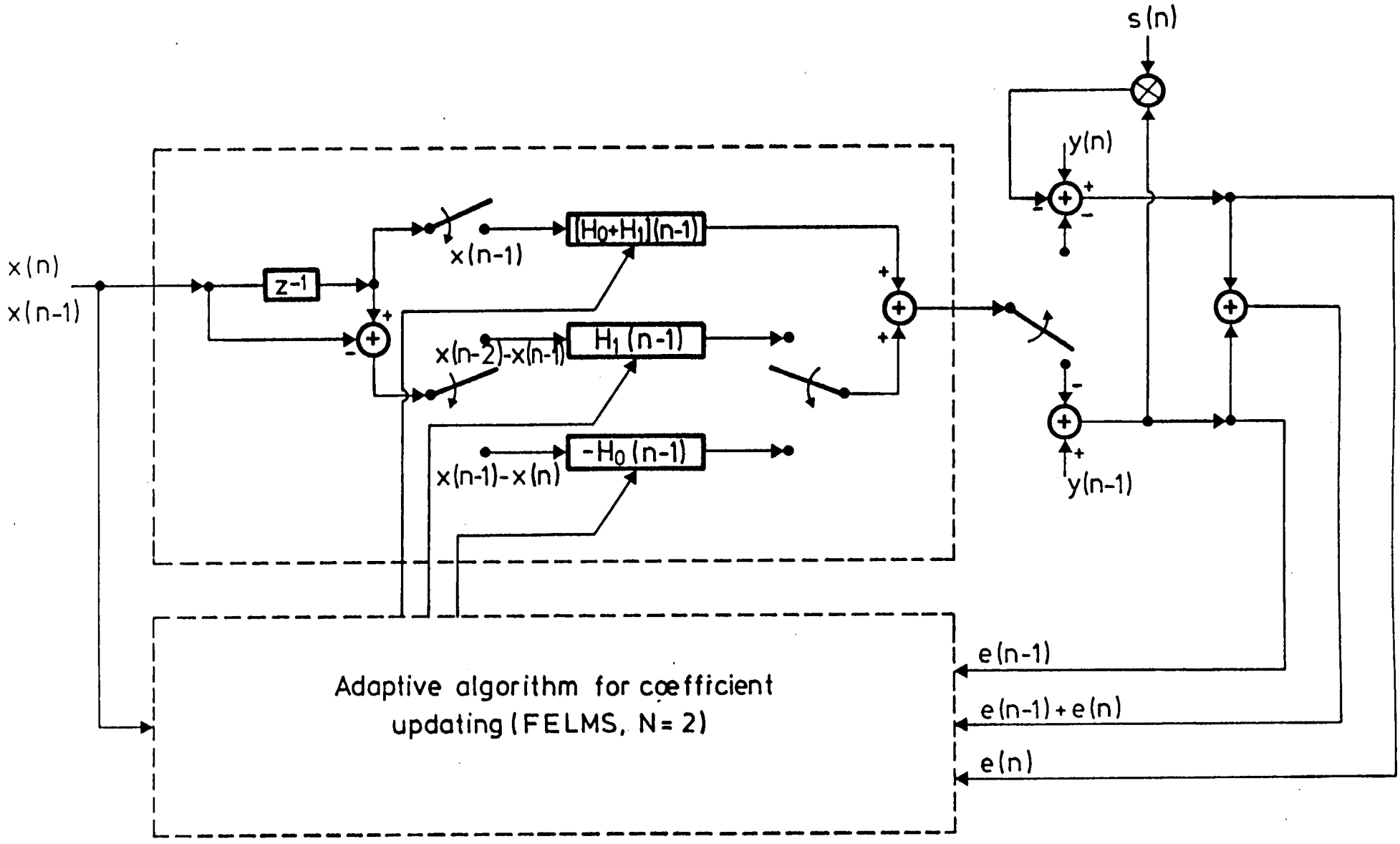


Fig.2: An FELMS structure for block length N = 2

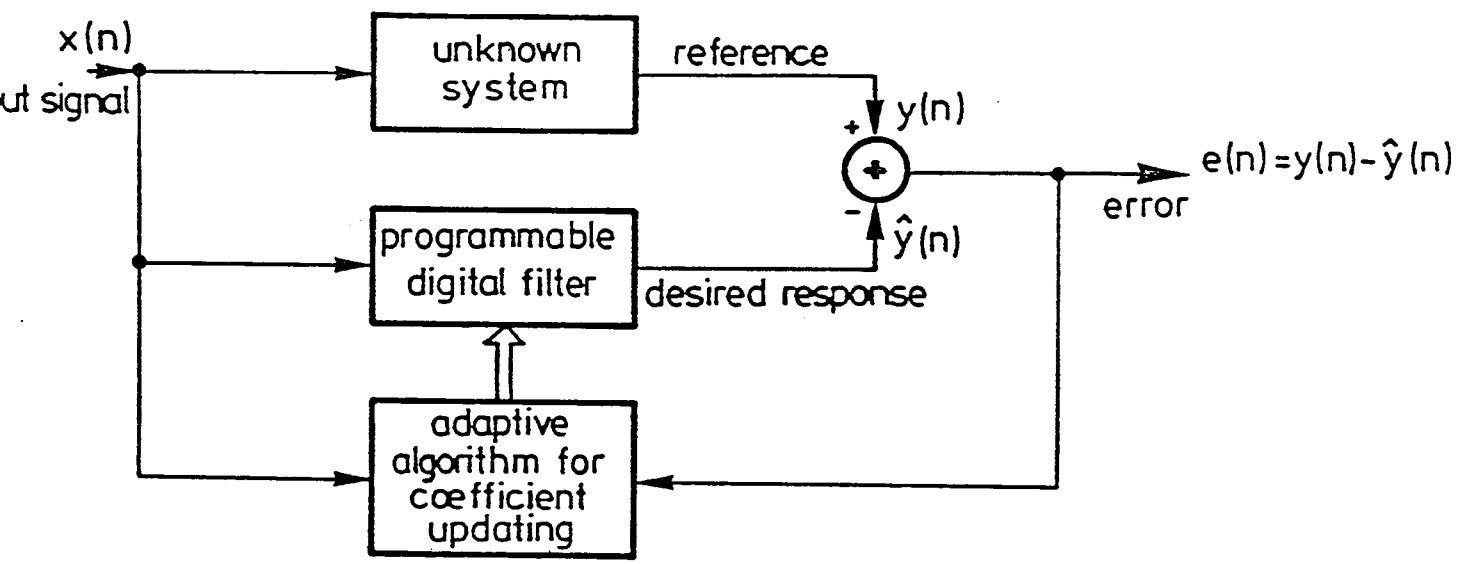


Fig.3: Adaptive structure used for system identification

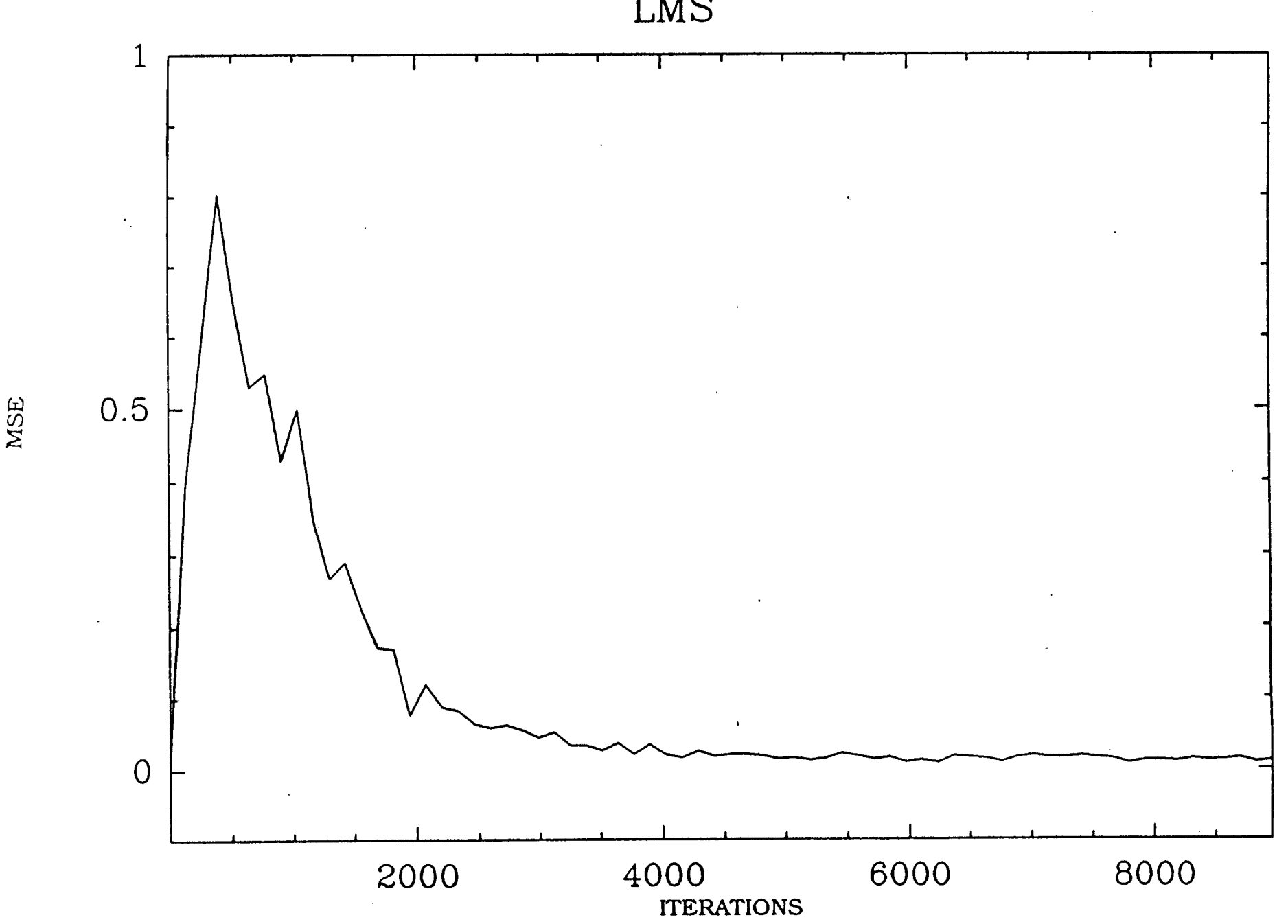

LMS
1
0.5
0
MSE
2000
4000
6000
8000
ITERATIONS
Benesty 19-JUL-1989

Fig. (4) a)

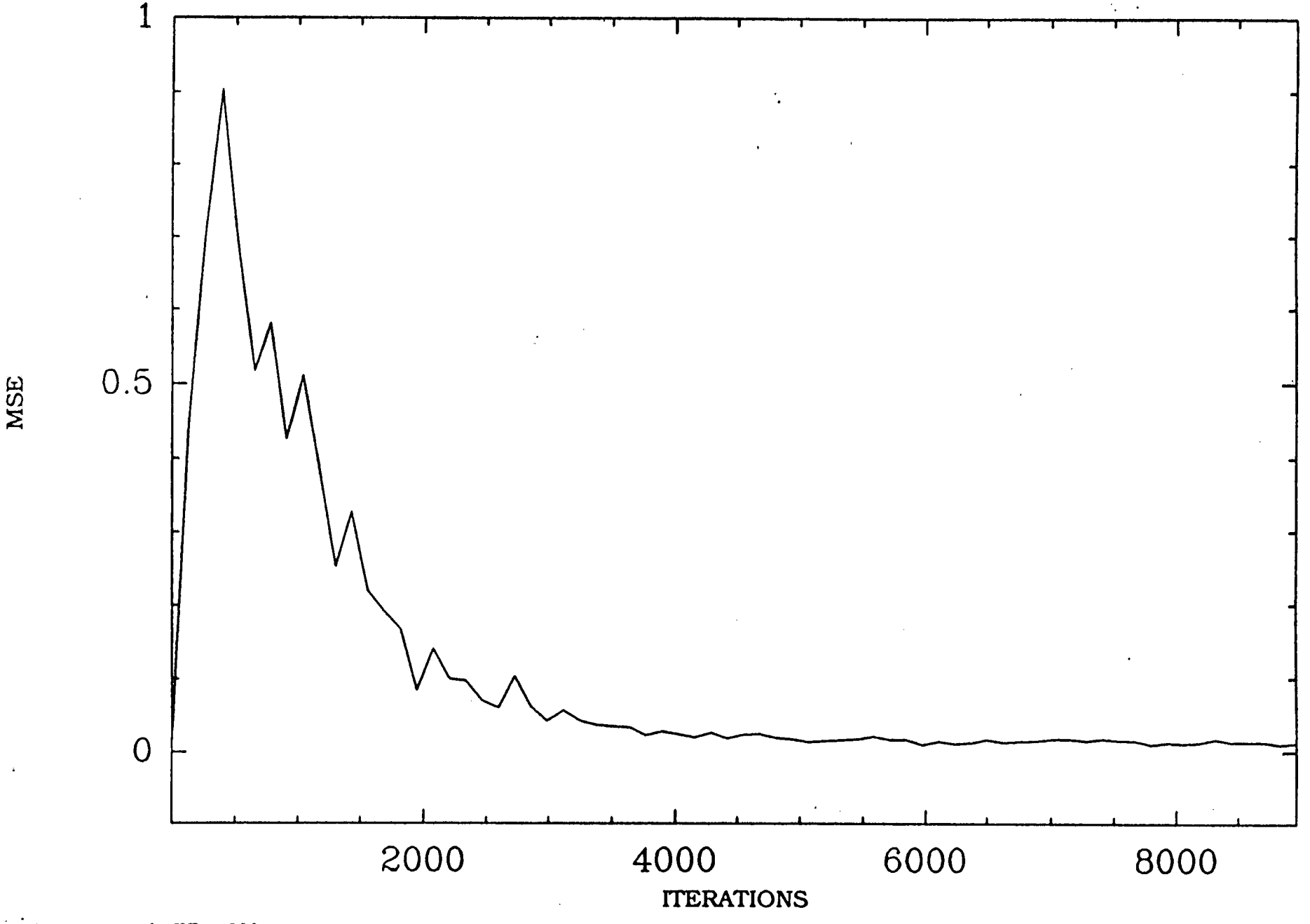


Fig. (4) b)

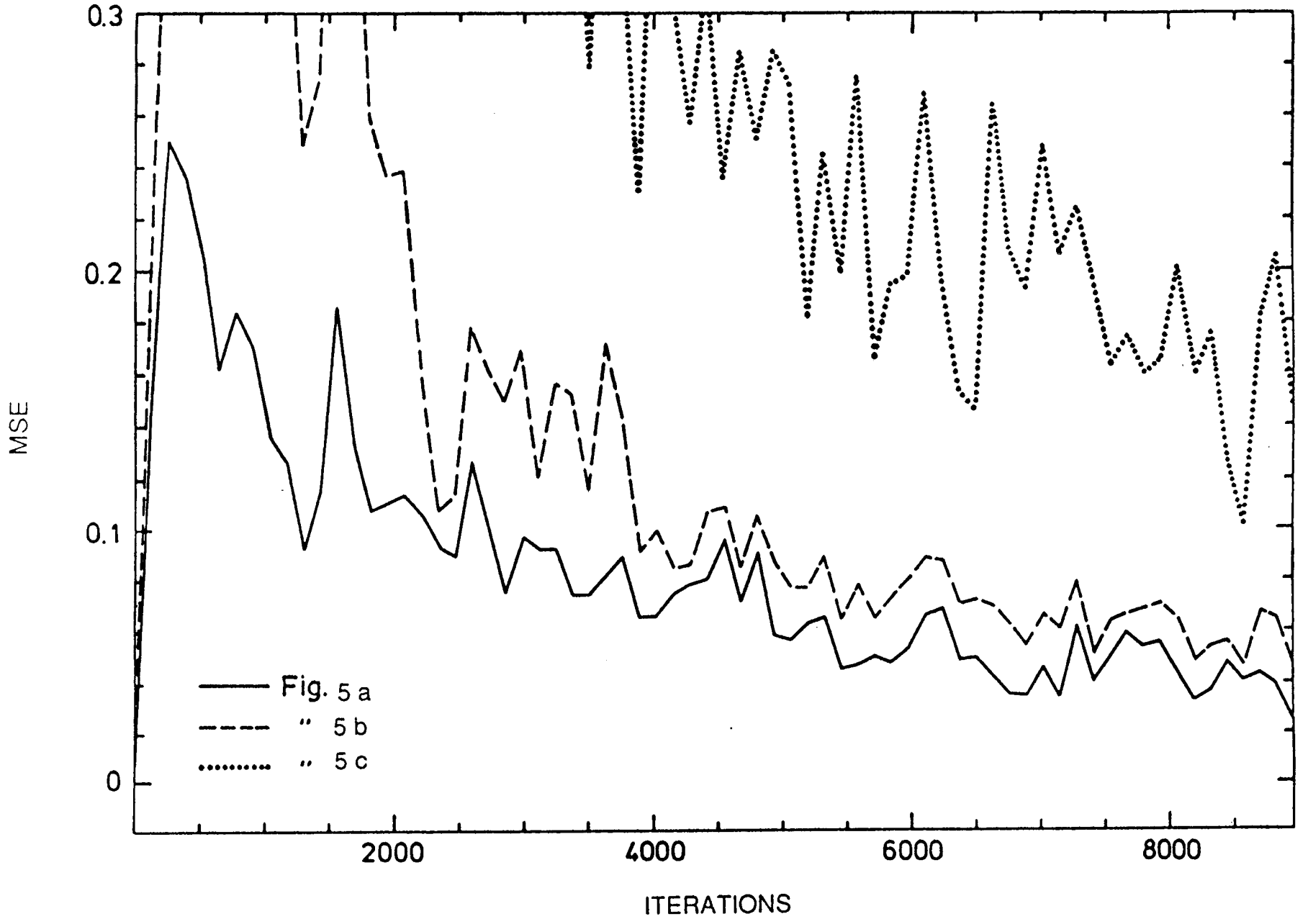


Fig. 5

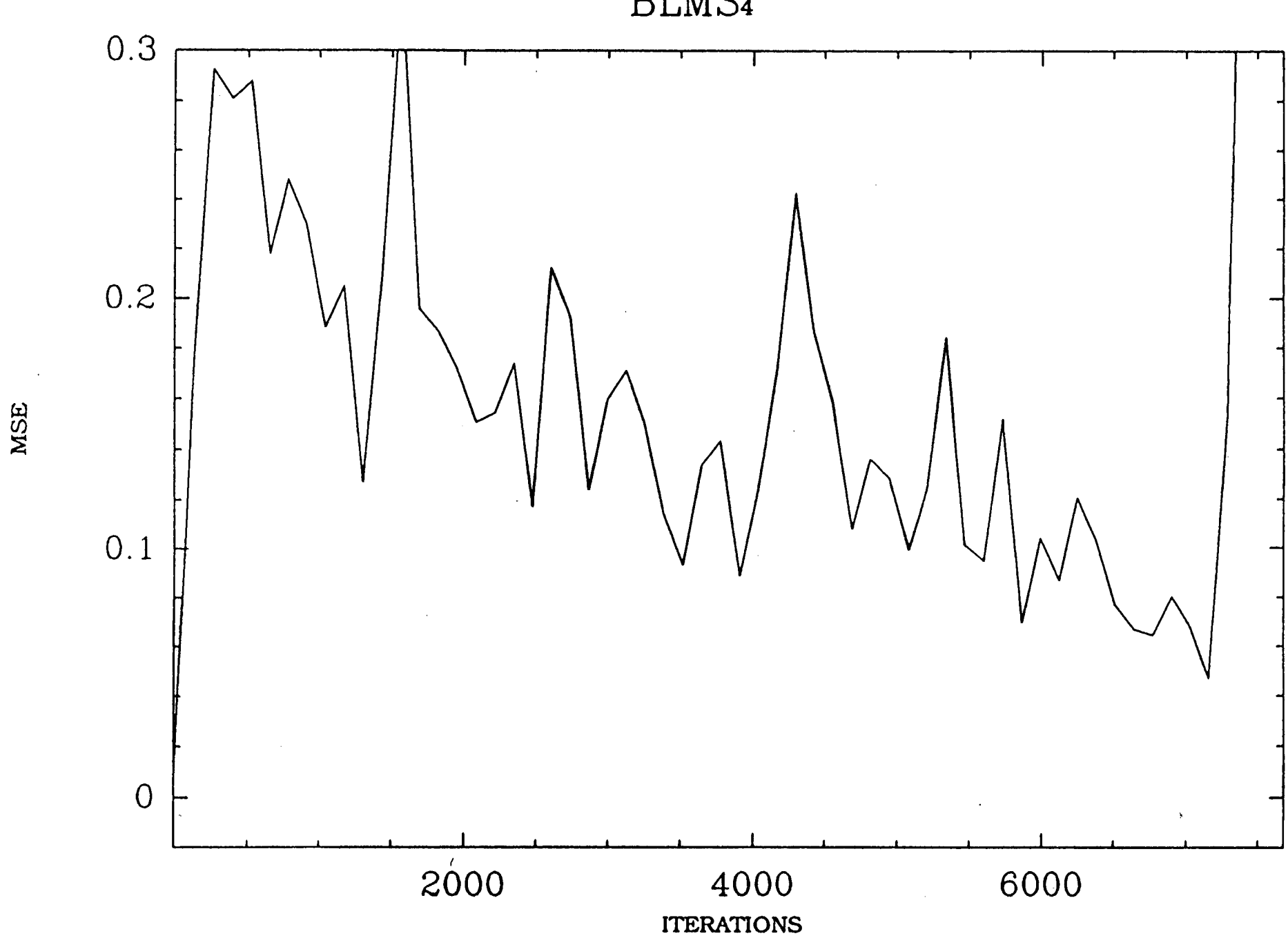

BLMS4
0.3
0.2
0.1
0
MSE
2000
4000
6000
ITERATIONS
Benesty 26-APR-1989

Fig. 6

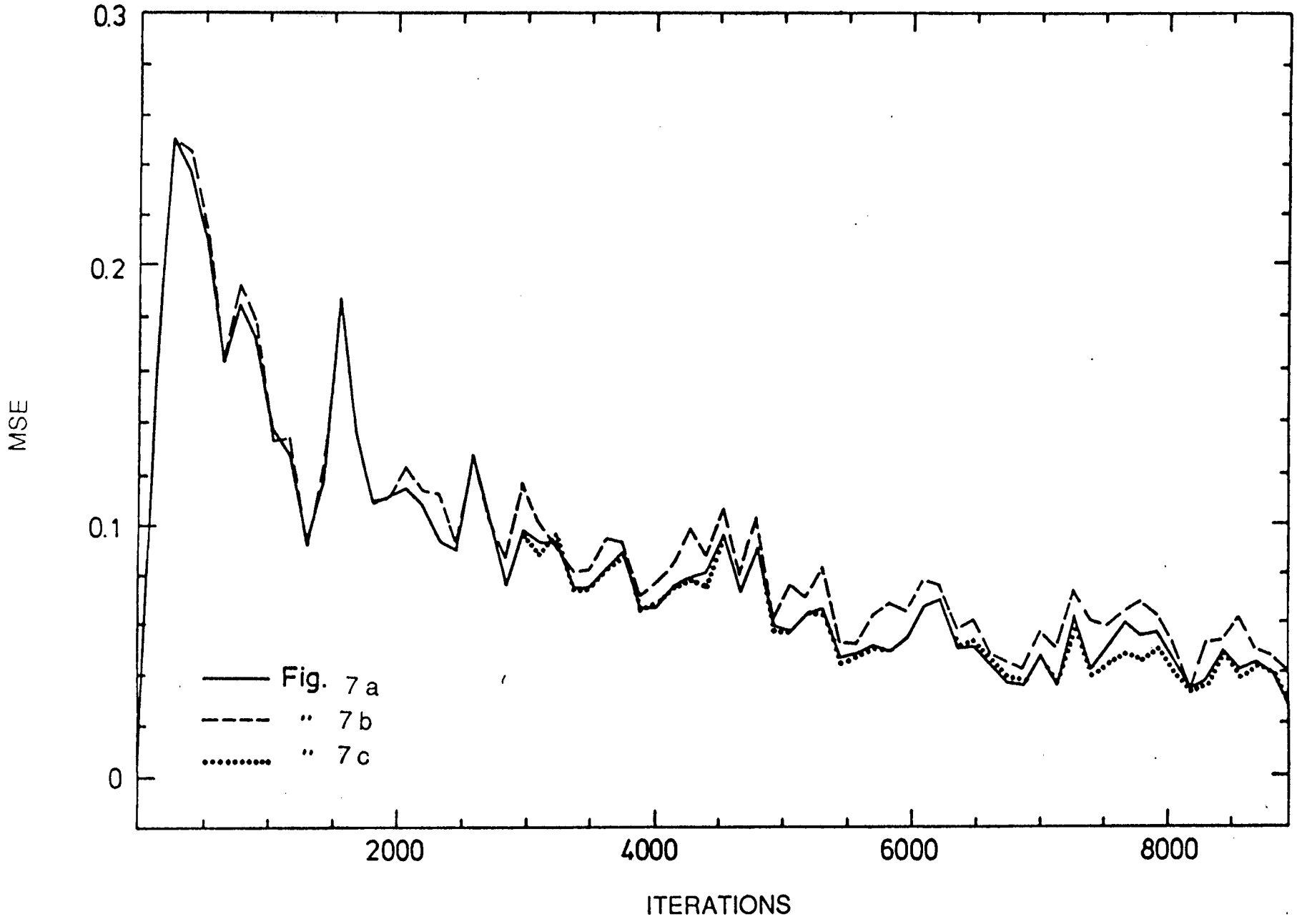


Fig. 7

Filter length L	block length N	LMS algorithm		FELMS algorithm		FALMS algorithm		BLMS algorithm	
		number of additions	number of multiplications	number of additions	number of multiplications	number of additions	number of multiplications	number of additions	number of multiplications
32	8	64	64	70	37	63	32	56	27
64	8	128	128	116	64	109	59	102	54
128	16	256	256	192	103	173	88	162	81
256	32	512	512	315	167	267	131	253	122
512	32	1024	1024	542	289	494	252	480	243
1024	64	2048	2048	865	458	756	376	739	365

Table 1 : comparison of the number of operations per output point required by the various algorithms

ANNEXE

ARTICLE II

(à paraître dans IEE Proceedings Part F Special Issue on Adaptive Filters)

A Fast Constant Modulus Adaptive Algorithm

Jacob BENESTY, Pierre DUHAMEL

CNET/PAB/RPE

38-40, rue du Général Leclerc

92131, ISSY-LES-MOULINEAUX

FRANCE

ABSTRACT

In this paper, an exact block formulation of the Constant Modulus Algorithm (CMA) is presented, on which a reduction of arithmetic complexity is achieved. Two types of fast algorithms are explained, either in time-domain or in frequency-domain. The first one is of greater interest for small block lengths, while the second one, using the FFT as an intermediate step, has greater advantage for large blocks. Due to the equivalence between the original CMA formulation and ours, the convergence properties of the CMA are maintained, which is not the case in the Treichler et *al*. implementation in frequency domain of this algorithm. Furthermore, our approach allows the use of very small block lengths (e.g., N=2), the reduction of the arithmetic complexity increasing with the block size.

I. INTRODUCTION

The Constant Modulus Adaptive Algorithm (CMA) is a special case of a more general algorithm that was first proposed by D.N. Godard [1] as a method for blind equalization for data modems. Another similar algorithm was proposed by Benveniste et *al.* [9]. The CMA was extensively studied by Treichler et *al.* [2,3] for a communication application. Indeed, in many modulation schemes, such as frequency modulation (FM) and phase modulation (PM), the signal to be transmitted possesses the constant envelope property. The received signal, however, has lost this property due to multipath and interference effects. The CMA restores the constant envelope property of the signal and increases the SNR. This algorithm thus employs just the a priori knowledge about the envelope of the transmitted signal and has the nice characteristic that no reference signal is required.

Nevertheless, the CMA has some shortcomings. First, it involves the minimization of a nonconvex cost function [1]. This non-convexity implies the existence of local minima, and a satisfactory convergence of the algorithm does not imply a true minimization of the cost function. Second, the algorithm may capture a constant modulus interferer rather than the constant modulus signal of interest [3]. These two problems can be overcome by a simple filter initialization [1,3] and will not be considered here. Another drawback is the large number of arithmetic operations required for this algorithm. In order to reduce this load, Treichler et *al.* [4] proposed to compute the nonlinear error in the time-domain while updating weights and filtering in the frequency-domain. Unfortunately, this algorithm is only an approximation of the initial one, and has been observed to have very slow convergence [4].

The main result of this paper is that it is possible to both reduce the arithmetic complexity of the CMA -by working in blocks that may be very small, if required- and maintain convergence properties: from the mathematical point of view, the algorithm thus obtained is strictly equivalent to the CMA.

Section II briefly recalls the initial version of the CMA, and provides an evaluation of the arithmetic complexity in two cases of implementation.

Section III provides the basis of our approach: merging the computations of two successive

CMA outputs permits to reduce the required number of operations per output point.

This is generalized in section IV in which we establish an exact block formulation of the CMA on which a reduction of the arithmetic complexity is feasible by using the "Fast FIR" technique [5,8]. Two special cases are studied with more details: The first one is the recursive application of the fast FIR of length 2, which has the advantage of allowing an improvement of the arithmetic complexity even for very small block lengths. The second one, which uses FFT as an intermediate step, is more efficient for larger blocks.

Note that the blocksize never depends on the filter's length, and that the usual constraint that the FFT length should be at least twice the filter's length does not hold here. This fact has a lot of advantages when thinking of memory requirements or overall system delay.

II. THE INITIAL CMA

2.1. Derivation of the algorithm

The Constant Modulus Algorithm's organization is depicted in Fig.1, where y(n) is the output of a complex FIR filter:

(1)

$$y(n) = X_n^t H = H^t X_n = \sum_{i=0}^{L-1} x(n-i)h_i$$

L being the length of the filter, X_n the vector of the past L complex data at time n , and H the vector of complex weights:

$$X_n = [x(n)\ x(n-1)\ .....\ x(n-L+1)]^t$$

$$H = [h_0\ h_1\ .....\ h_{L-1}]^t$$

The purpose of the adaptation process is to find a weight vector H that minimizes fluctuations in the complex envelope of the output y(n). Hence, a natural criterion J measures the distance between the modulus of y(n) and the constant modulus of the transmitted signal:

(2)

$$J = \frac{1}{4} E\left\{\left[|y(n)|^2 - 1\right]^2\right\}$$

where E denotes statistical expectation and where the modulus of the signal is assumed to be equal to 1.

A possible algorithm for the coefficients updating is as follows:

(3)

$$H_{n+1} = H_n - \mu \nabla J(n)$$

where μ is a positive step size and ∇ the gradient operator:

(4)

$$\nabla J(n) = \frac{\partial J(n)}{\partial H_n}$$
$$= E\left\{\left[|y(n)|^2 - 1\right] y(n) X_n^*\right\}$$

where * denotes complex conjugation.

Of course, since eq.(4) involves a mathematical expectation, it cannot be used as it is. It has been proposed in [1,2] to replace it by an instantaneous gradient estimate, as given in (5):

(5)

$$\hat{\nabla} J(n) = \left[|y(n)|^2 - 1\right] y(n) X_n^*$$

The CMA is thus described by the following set of equations:

(6)

a) $y(n) = X_n^t H_n$

b) $\alpha(n) = \mu\left[|y(n)|^2 - 1\right] y(n)$

c) $H_{n+1} = H_n - \alpha(n) X_n^*$

2.2. Arithmetic complexity of the CMA

2.2.1. Initial version (CMA1)

Assuming the usual 4 mult - 2 add complex multiplication scheme is used, the arithmetic complexity of the initial CMA, as described by eq.(6), is as follows: Eq.(6a) requires 4L real multiplications, and (4L - 2) real additions. The computation of (6b) requires 5 real

multiplications and 2 real additions, while eq.(6c) requires 4L real multiplications and 4L real additions. This results in the following total number of real operations per output point:

(7)

8L+5 multiplications

(8)

8L additions

Note that since μ has not been constrained to be a negative power of 2, it appears in this count.

2.2.2. "Fast" complex multiply-based version

It is well known that a complex multiplication can be computed by any of the following two equations [7]:

(9)

a) $$(x_r + jx_i)(h_r + jh_i) = \left[(x_r + x_i)h_r - x_i(h_r + h_i)\right] + j\left[(x_r + x_i)h_r - x_r(h_r - h_i)\right]$$

b) $$= \left[(h_r + h_i)x_r - h_i(x_r + x_i)\right] + j\left[(h_r + h_i)x_r - h_r(x_r - x_i)\right]$$

In the case of fixed coefficient FIR filtering, eq. (9a) is preferred, because $h_r \pm h_i$ can be precomputed, so that the overall computational load is 3 mults and 3 adds, which results in an exchange of one multiplication for one addition. When h_r and h_i are not fixed, the apparent cost is (3 mults, 5 adds). However, it is shown in the following that the use of (9b) in the CMA equations (6) allows for a reduction in the total number of operations as compared to the initial algorithm. Using (9b), (6) is rewritten as follows:

(10)

a) $$y(n) = \left[(H_n^r + H_n^i)^t X_n^r - (X_n^r + X_n^i)^t H_n^i\right] + j\left[(H_n^r + H_n^i)^t X_n^r - (X_n^r - X_n^i)^t H_n^r\right]$$

b) $$\alpha(n) = \mu\left[|y(n)|^2 - 1\right]y(n)$$

c) $$H_{n+1}^r = H_n^r - \left[(\alpha_r(n) - \alpha_i(n))X_n^r + \alpha_i(n)(X_n^r + X_n^i)\right]$$

$$H_{n+1}^i = H_n^i - \left[(\alpha_r(n) - \alpha_i(n))X_n^r - \alpha_r(n)(X_n^r - X_n^i)\right]$$

A straightforward operation count would be as follows: 3L mults and 6L - 1 adds in (10a), 5 mults and 2 adds in (10b) and 3L mults and 4L + 1 adds in (10c).

Nevertheless, remembering that $(X^r_n \pm X^i_n)$ has a single additional term compared to $(X^r_{n-1} \pm X^i_{n-1})$ and that identical terms are stored in the filtering process, it is easily seen (Fig.2) that 2(L - 1) additions can be saved.

The overall CMA process, based on a complex FIR scheme as depicted in Fig.2 hence requires:

(11)

6L+5 mults

(12)

8L+4 adds

which reduces the total number of operations by about 2L. In other words, one fourth of the number of multiplications has been saved, at the cost of 25% additional memory locations in some implementations. This second algorithm will be referred to as the CMA2.

With these two versions of the CMA as starting points, we shall derive in the remaining of this paper an exact block formulation of this algorithm, which allows a reduction of the arithmetic complexity. The tools we use are the same ones as in the fixed coefficients case [8], and this derivation follows the same lines as for the LMS case [6], on which similar work was already performed. The main difference between the treatment of LMS and CMA algorithms are found in the type of signals (complex for the CMA, real-valued in [6]), and in the non-linearity of the instantaneous gradient expression (5). Both differences do not bring major drawbacks on the derivation of the algorithms.

III. AN EXAMPLE OF CMA WITH REDUCED NUMBER OF OPERATIONS

Let us first consider the required computations in the CMA at two successive time samples n-1 and n. By appropriately re-arranging the corresponding equations, we shall obtain an exact equivalent of eq.(6) requiring a lower number of operations per output point.

Consider eq.(6), written at time n-1:

(13)

a) $y(n-1) = X_{n-1}^t H_{n-1}$

b) $\alpha(n-1) = \mu\left[|y(n-1)|^2 - 1\right] y(n-1)$

c) $H_n = H_{n-1} - \alpha(n-1) X_{n-1}^*$

Substituting (13c) into (6a) results in:

(14)

$$y(n) = X_n^t H_{n-1} - \alpha(n-1) X_n^t X_{n-1}^*$$
$$= X_n^t H_{n-1} - \alpha(n-1)\, s(n)$$

where:

$$s(n) = X_n^t X_{n-1}^*$$

Equations (14), (13a) can be combined in matrix form, thus providing two successive outputs of the system:

(15)

$$\begin{bmatrix} y(n-1) \\ y(n) \end{bmatrix} = \begin{bmatrix} X_{n-1}^t \\ X_n^t \end{bmatrix} H_{n-1} - \begin{bmatrix} 0 & 0 \\ s(n) & 0 \end{bmatrix} \begin{bmatrix} \alpha(n-1) \\ \alpha(n) \end{bmatrix}$$

The first term of this equation appears to be the computation of two successive outputs of a fixed coefficient filter. Thus, we can apply on the fixed part of eq.(15) the same techniques as explained in [8]:

(16)

$$\begin{bmatrix} X_{n-1}^t \\ X_n^t \end{bmatrix} H_{n-1} = \begin{bmatrix} x(n-1) & x(n-2).....x(n-L) \\ x(n) & x(n-1).....x(n-L+1) \end{bmatrix} \begin{bmatrix} h_0(n-1) \\ h_1(n-1) \\ \vdots \\ \vdots \\ h_{L-1}(n-1) \end{bmatrix}$$

$$= \begin{bmatrix} x(n-1) & x(n-3)...x(n-L+1) & x(n-2) & x(n-4)...x(n-L) \\ x(n) & x(n-2)...x(n-L+2) & x(n-1) & x(n-3)...x(n-L+1) \end{bmatrix} \begin{bmatrix} h_0(n-1) \\ h_2(n-1) \\ \vdots \\ h_{L-2}(n-1) \\ h_1(n-1) \\ h_3(n-1) \\ \vdots \\ h_{L-1}(n-1) \end{bmatrix}$$

where the even and odd terms of the involved vectors have been grouped. Furthermore, in order to obtain a more compact notation, assume L is even, and define:

$$A_0 = [x(n)\ x(n-2)\ .....\ x(n-L+2)]^t$$

$$A_1 = [x(n-1)\ x(n-3)\ .....\ x(n-L+1)]^t$$

$$A_2 = [x(n-2)\ x(n-4)\ .....\ x(n-L)]^t$$

$$H_{n-1}^0 = [h_0(n-1)\ h_2(n-1)\ .....\ h_{L-2}(n-1)]^t$$

$$H_{n-1}^1 = [h_1(n-1)\ h_3(n-1)\ .....\ h_{L-1}(n-1)]^t$$

Equation (16) is now rewritten as:

(17)

$$\begin{bmatrix} X_{n-1}^t \\ X_n^t \end{bmatrix} H_{n-1} = \begin{bmatrix} A_1^t & A_2^t \\ A_0^t & A_1^t \end{bmatrix} \begin{bmatrix} H_{n-1}^0 \\ H_{n-1}^1 \end{bmatrix}$$

The same kind of work can be performed for the updating of the filter taps. First substitute (13c) into (6c):

(18)

$$H_{n+1} = H_{n-1} - \alpha(n) X_n^* - \alpha(n-1) X_{n-1}^*$$

Or, with the above notations:

(19)

$$\begin{bmatrix} H^0_{n+1} \\ H^1_{n+1} \end{bmatrix} = \begin{bmatrix} H^0_{n-1} \\ H^1_{n-1} \end{bmatrix} - \alpha(n) \begin{bmatrix} A^*_0 \\ A^*_1 \end{bmatrix} - \alpha(n-1) \begin{bmatrix} A^*_1 \\ A^*_2 \end{bmatrix}$$

Now, the following set of equations is exactly equivalent to the definition of the CMA, for a block of two outputs:

(20)

a) $$\begin{bmatrix} y'(n-1) \\ y'(n) \end{bmatrix} = \begin{bmatrix} A^t_1 & A^t_2 \\ A^t_0 & A^t_1 \end{bmatrix} \begin{bmatrix} H^0_{n-1} \\ H^1_{n-1} \end{bmatrix}$$

b) $$\begin{bmatrix} y(n-1) \\ y(n) \end{bmatrix} = \begin{bmatrix} y'(n-1) \\ y'(n) \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ s(n) & 0 \end{bmatrix} \begin{bmatrix} \alpha(n-1) \\ \alpha(n) \end{bmatrix}$$

c) $$\begin{bmatrix} H^0_{n+1} \\ H^1_{n+1} \end{bmatrix} = \begin{bmatrix} H^0_{n-1} \\ H^1_{n-1} \end{bmatrix} - \begin{bmatrix} A^*_1 & A^*_0 \\ A^*_2 & A^*_1 \end{bmatrix} \begin{bmatrix} \alpha(n-1) \\ \alpha(n) \end{bmatrix}$$

Note that from a computational point of view, eq.(20b) states some problem: the computation of y(n) seems to require the knowledge of $\alpha(n)$ which itself is defined in terms of y(n). Nevertheless, since the matrix involved in eq.(20b) is strictly lower triangular,y(n) depends on $\alpha(n-1)$ and this equation can be solved as follows: y(n-1) is readily obtained, then compute $\alpha(n-1)$ by its definition (13b), then obtain y(n) by the second line of (20b), and finally compute $\alpha(n)$ from y(n). Although $\alpha(n)$ and y(n) are related in a non-linear manner, this kind of equation will always be solvable by substitution, due to the nature of the matrix involved in eq.(20b) (strictly lower triangular). (Nevertheless, the non-linearity of y(n) in terms of $\alpha(n)$ prevents the possibility of a closed-form solution, as was the case for the LMS algorithm [6]).

Now, reduction of arithmetic complexity can take place, by rewriting (20) as:

(21)

a) $$\begin{bmatrix} y'(n-1) \\ y'(n) \end{bmatrix} = \begin{bmatrix} A_1^t(H_{n-1}^0 + H_{n-1}^1) + (A_2 - A_1)^t H_{n-1}^1 \\ A_1^t(H_{n-1}^0 + H_{n-1}^1) - (A_1 - A_0)^t H_{n-1}^0 \end{bmatrix}$$

b) $$\begin{bmatrix} y(n-1) \\ y(n) \end{bmatrix} = \begin{bmatrix} y'(n-1) \\ y'(n) \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ s(n) & 0 \end{bmatrix} \begin{bmatrix} \alpha(n-1) \\ \alpha(n) \end{bmatrix}$$

c) $$\begin{bmatrix} H_{n+1}^0 \\ H_{n+1}^1 \end{bmatrix} = \begin{bmatrix} H_{n-1}^0 \\ H_{n-1}^1 \end{bmatrix} - \begin{bmatrix} A_1^*(\alpha(n-1) + \alpha(n)) - (A_1 - A_0)^* \alpha(n) \\ A_1^*(\alpha(n-1) + \alpha(n)) + (A_2 - A_1)^* \alpha(n-1) \end{bmatrix}$$

The reduction of the number of operations has mainly been obtained in the first equation of the set: (21a) involves only three different length L/2 inner products instead of 4 inner products of the same size in (20a).

We shall now more precisely explain the organization of the computation, step by step, and evaluate the arithmetic complexity:

step a) computation of y'(n-1) and y'(n) by eq.(21a)

step b) recursive computation of s(n):

(22)

$$s(n) = s(n-2) + [x(n)x^*(n-1) + x(n-1)x^*(n-2) - x(n-L)x^*(n-L-1) - x(n-L-1)x^*(n-L-2)]$$

step c) y(n-1)=y'(n-1). Compute α(n-1) by (13b), then substitute in (21b) to get y (n), and finally use (6b) to obtain α(n)

step d) compute the update of H - eq. (21c)

step e) incrementation of n by 2 then go to step a)

Several considerations similar to the ones in [6] , allow to precisely evaluate the number of complex arithmetic operations involved in (21):

Step a) The filtering operation $A^t{}_1(H^0{}_{n-1}+H^1{}_{n-1})$ is common between the two terms of (21a),

which requires 3 length L/2 filters, two of which are applied to combinations of the input samples, namely:

(23)

$$A_2 - A_1 = [x(n-2) - x(n-1), \cdots\cdots, x(n-L) - x(n-L+1)]^t$$
$$A_1 - A_0 = [x(n-1) - x(n), \cdots\cdots, x(n-L+1) - x(n-L+2)]^t$$

Just like in section II, the apparent number of additions involved in (23) can be reduced by noticing that the previous set of scalar products involved in the computation of y(n-3), y(n-2) already required nearly the same combinations of the input samples, and that only two new complex additions are to be computed: (x(n-2)-x(n-1)) and (x(n-1)-x(n)).

Step b) involves two complex multiplications and two complex additions (half the complex mults were already computed previously).

Step c) involves the computation of y(n) with one complex addition and one complex multiplication, plus two equations of the type (13b), which require a total of six real mults.

Finally, step d) requires the complex product $A^*_1(\alpha(n-1)+\alpha(n))$, which is common between the two equations (21c). Moreover, (A_1-A_0) and (A_2-A_1) were already calculated in (21a).

The above considerations allow to evaluate the reduction in the number of complex operations per output point required for implementing the CMA. Nevertheless, the precise improvement in terms of real operations depends on the way the complex multiplications are performed (see section 2.2).

When the complex multiplications are performed with the usual 4 mult-2 add scheme, the total number of operations for computing two outputs is:

(24)

12L+22 real multiplications

(25)

14L+22 real additions

We denote this algorithm by FCMA1, for a block length N=2.

When the "fast" complex multiply scheme is used for computing the length L/2 filters, as explained in section 2.2.2, the corresponding algorithm is called FCMA2, and its computational load for a blocklength N=2 is:

(26)

9L+22 real multiplications

(27)

14L+36 real additions

Table 1 gives the number of operations per output point for all the algorithms explained up to now: CMA1, CMA2, FCMA1 and FCMA2.

It can be observed that each "fast" algorithm reduces by about 25% the number of multiplications compared to their initial counterpart, while slightly reducing the number of additions. The total number of real operations is seen to be 20% less than the one required by a straightforward implementation of the CMA.

Note that this reduction is obtained only by a re-arrangement of the initial equations, and that there is an exact equivalence, mathematically speaking, between the initial algorithm and our block version of it. Hence, all these algorithms have the same convergence rate.

The above explanations follow closely the work we have performed on the LMS algorithm. Nevertheless, because of the nonlinear error of the CMA, some equations (eq.(21b)) need more inspection than in the LMS case.

This method has been explained in a rather specific manner, by merging the computations of two successive CMA outputs. We show in the next section that this approach is much more general: grouping the computations of more outputs results in greater computational savings.

IV. GENERALISATION TO ARBITRARY N

We shall follow the same lines as in section III: First, we provide an exact block formulation of the CMA for arbitrary N, which is in the form of a fixed filtering, followed by a correction of

the outputs, and an update of the coefficients that are to be used in the next block. Concerning the fixed coefficients filtering, we shall refer essentially to ref. [5,8], and only recall some results. We shall rather concentrate on the adaptive part of this algorithm.

4.1. Exact block formulation of the CMA

Let us write the fixed FIR filter output equations at time n-N+1, n-N+2,....., n-1, n:

(28)

$$\begin{bmatrix} y'(n-N+1) \\ y'(n-N+2) \\ \vdots \\ \vdots \\ y'(n-1) \\ y'(n) \end{bmatrix} = \begin{bmatrix} X^t_{n-N+1} \\ X^t_{n-N+2} \\ \vdots \\ \vdots \\ X^t_{n-1} \\ X^t_n \end{bmatrix} H_{n-N+1}$$

In the same manner as with the example N=2, we may write the exact output equations at time n-N+1, n-N+2, ..., n-1, n, of the CMA:

(29)

$$\begin{bmatrix} y(n-N+1) \\ y(n-N+2) \\ \vdots \\ \vdots \\ y(n-1) \\ y(n) \end{bmatrix} = \begin{bmatrix} y'(n-N+1) \\ y'(n-N+2) \\ \vdots \\ \vdots \\ y'(n-1) \\ y'(n) \end{bmatrix} - S(n) \begin{bmatrix} \alpha(n-N+1) \\ \alpha(n-N+2) \\ \vdots \\ \vdots \\ \alpha(n-1) \\ \alpha(n) \end{bmatrix}$$

with

(30)

$$S(n) = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ s_1(n-N+2) & 0 & & \vdots \\ s_2(n-N+3) & s_1(n-N+3) & \cdots & \vdots \\ \vdots & \vdots & & \\ \vdots & \vdots & & \\ s_{N-1}(n) & s_{N-2}(n) & \cdots s_1(n) & 0 \end{bmatrix}$$

where

$s_i(n) = X_n^t X_{n-i}^*$, $i = 1,2,.....,N-1$

The remaining part of the algorithm is the coefficients updating which is expressed as follows:

(31)

$$H_{n+1} = H_{n-N+1} - \left[X_{n-N+1}^* \quad X_{n-N+2}^* \quad ..... \quad X_{n-1}^* \quad X_n^* \right] \begin{bmatrix} \alpha(n-N+1) \\ \alpha(n-N+2) \\ \vdots \\ \vdots \\ \alpha(n-1) \\ \alpha(n) \end{bmatrix}$$

(28), (29) and (31) can be written in matrix form as follows:

(32)

a) $Y_n' = \underline{X}(n) H_{n-N+1}$

b) $Y_n = Y_n' - S(n)\alpha_n$

c) $H_{n+1} = H_{n-N+1} - \underline{X}^{\dagger}(n)\alpha_n$

where † denotes the transpose conjugate, Y_n' is a vector of N successive outputs of a fixed filter, Y_n represents N successive outputs of the complex CMA filter, $\underline{X}(n)$ is a matrix (NxL) of the N last input vectors and α_n the vector (Nx1) formed from $\alpha(n-i)$ for i=0 to i=N-1.

The set of equations (32) with the expressions relating $\alpha(n)$ and y(n) (eq.(6b) at time n-N+1,.....,n) form an exact equivalent of (6) for a whole block of output y(n-N+1),.....,y(n). Eq.(28) is an FIR filtering, whose coefficients remain unchanged during the whole block of outputs, and is thus amenable to a reduction of the arithmetic complexity through the techniques explained in [8].

Eq. (29) requires more inspection, since both vectors on each side of the equation depend on the same unknowns Y_n through eq.(6b). Nevertheless, since S(n) is strictly lower triangular, (29) can be solved in a manner strictly parallel to the computation of the solution of a linear system with a lower-triangular matrix: First initialize y(n-N+1)=y'(n-N+1) then obtain α(n-N+1) by (6b) at time n-N+1, solve in y(n-N+2) using the second line of (29) from which

$\alpha(n-N+1)$ is obtained (6b).Then, $\alpha(n-N+1)$ and $\alpha(n-N+2)$ allow the computation of $y(n-N+3)$ by the third line of (29). Iterating the process provides both Y_n and α_n in (32b). Eq. (32c) then provides the values of H to be used in the next iteration. It is seen that the filter weights are updated once per data block instead of once per data sample. Nevertheless, this updating is performed in such a manner that the weights are equal to those that could have been found in the initial CMA at the same time. The only drawback of this blockwise adaptation, besides the explicit knowledge of a single weight vector per block, is a delay in the output, which is equal to the block length, hence is easily under control.

In this way, expressions (32) are an exact block formulation of the CMA with the advantage that arithmetic complexity can be saved by using the same techniques as described in [6].

4.2. Block Toeplitz formulation

Let as assume that L=NM, M a positive integer. A formulation of (28-31) using subsampled versions of the different signals involved allows the derivation of the fast algorithm for any N:

Define:

(33)

$$A_j = [\, x(n-j) \quad x(n-N-j) \quad \ldots \quad x(n-Ni-j) \quad \ldots \quad x(n-L+N-j) \,]^t$$
$$j = 0,1,\ldots\ldots,2N-2$$
$$i = 0,1,\ldots\ldots,(L/N)-1$$

a vector of length L/N; and

(34)

$$H^k_{n-N+1} = [\, h_k \quad h_{k+N} \quad \ldots\ldots \quad h_{k+Ni} \quad \ldots\ldots \quad h_{k+L-N} \,]^t (n-N+1)$$
$$k = 0,1,\ldots\ldots,N-1$$

Then, eq.(28) becomes:

(35)

$$Y_n' = \begin{bmatrix} A_{N-1}^t & A_N^t & \cdots & A_{2N-3}^t & A_{2N-2}^t \\ A_{N-2}^t & A_{N-1}^t & \cdots & & A_{2N-3}^t \\ \vdots & & & & \vdots \\ \vdots & & & & \vdots \\ A_1^t & & & & A_N^t \\ A_0^t & A_1^t & \cdots & & A_{N-1}^t \end{bmatrix} \begin{bmatrix} H_{n-N+1}^0 \\ H_{n-N+1}^1 \\ \vdots \\ \vdots \\ H_{n-N+1}^{N-2} \\ H_{n-N+1}^{N-1} \end{bmatrix}$$

and (31) gives:

(36)

$$\begin{bmatrix} H_{n+1}^0 \\ H_{n+1}^1 \\ \vdots \\ \vdots \\ H_{n+1}^{N-2} \\ H_{n+1}^{N-1} \end{bmatrix} = \begin{bmatrix} H_{n-N+1}^0 \\ H_{n-N+1}^1 \\ \vdots \\ \vdots \\ H_{n-N+1}^{N-2} \\ H_{n-N+1}^{N-1} \end{bmatrix} - \begin{bmatrix} A_{N-1}^* & A_{N-2}^* & \cdots & A_1^* & A_0^* \\ A_N^* & A_{N-1}^* & \cdots & A_1^* & \\ \vdots & & & \vdots & \\ \vdots & & & \vdots & \\ A_{2N-3}^* & & & A_{N-2}^* & \\ A_{2N-2}^* & A_{2N-3}^* & \cdots & A_{N-1}^* & \end{bmatrix} \begin{bmatrix} \alpha(n-N+1) \\ \alpha(n-N+2) \\ \vdots \\ \vdots \\ \alpha(n-1) \\ \alpha(n) \end{bmatrix}$$

Expressions (35) and (36) play a key role for reducing the arithmetic complexity of the algorithm, since they can be seen as a filtering equation with elements replaced by vectors. Hence, all non-commutative fast FIR filtering algorithms such as those explained in [5,6,8] apply on this kind of matrix-vector products. Eq. (29) has not been changed by this block formulation, and it seems that the most efficient way to compute the matrix S is the use of the following recursions:

A first equation (38) provides the expression of the first column of matrix S:

(38)

$$s_i(n-N+i+1) = s_i(n-N) + \sum_{j=0}^{i} x(n-N+i-j+1)x^*(n-N-j+1) - \sum_{j=0}^{i} x(n-L-N+i-j+1)x^*(n-L-N-j+1)$$

$$i = 1,\ldots\ldots,N-1$$

Equation (39) provides the computations to be performed along the diagonals:

(39)

$$s_i(n+1) = s_i(n) + x(n+1)x^*(n-i+1) - x(n-L+1)x^*(n-L-i+1)$$

The above considerations allow to evaluate precisely the number of arithmetic operations required for operating this block-CMA, whatever the block size N is. It should be noted that the number of operations to be performed per output point can be decomposed in two terms: the first one is due to the "fixed" coefficient filtering and to the update of H. This term decreases with N: working with larger blocks results in more efficient algorithms. The second one is due to the computation of matrix S, and this term increases with N. Therefore, for a given filter length, there exist an optimum blocksize that also depends on the type of fast algorithm that is used. The next two sections study two special cases of interest, where both the block-size and the filter's length are powers of 2.

4.3. FCMA based on short-length FIR algorithms

The first case of interest is the recursive application of the computation we used in section III. We showed in this section that for a block size of N=2, the fast algorithm requires 3 subfilters of length L/2. If each of these subfilters is in turn decomposed, the next iteration will result in 9 subfilters subsampled by 4. Hence, if the block length is a power of 2 ($N=2^n$), a fast FIR algorithm can be obtained by applying **n** times the decomposition (21a), thus resulting in 3^n subfilters of size L/N, the inputs and outputs of which are sub-sampled by a factor N compared to the input of the system. A precise description of the resulting schemes is provided in ref.[5] in the case of fixed coefficient FIR filtering.

An evaluation of the number of operations required by this algorithm for computing a block of $N = 2^n$ outputs is provided in appendix A.1. It is shown that, if the 4 mult-2 add complex multiply scheme is used in the filtering part (FCMA1), the number of operations to be performed per output point for a filter of length $L=2^n M$ is:

(37)

$8(3/2)^n M + 6\,2^n - 1$ multiplications

(38)

$4\,(\,3\,(3/2\,)^n - 1\,)\,M + 7\,2^n + 8\,(\,3/2\,)^n - 15$ additions

If the "fast" complex mutliply scheme is used (FCMA2, see section 2.2.2.), the resulting number of operations per output point are:

(39)

$6\,(3/2)^n\,M + 6\,2^n - 1$ multiplications

(40)

$4\,(3\,(3/2)^n - 1)\,M + 7\,2^n + 12\,(3/2)^n - 13 - 2/2^n$ additions

Note that as long as $M \geq 4$ (i.e. the filter is at least 4 times as long as the block length) and whatever the blocksize may be, the FCMA requires fewer operations than the CMA. This means that a reduction of the arithmetic complexity is feasible even for such short filters as L=8.

Furthermore, if we suppose that $M=2^n$,we see that the arithmetic complexity of FCMA varies with $0(3^n)$ instead of $0(4^n)$ for the CMA . This shows the efficiency of this approach.

The important point concerning these numbers is that the precise arithmetic complexity involves a term growing with N (updating of S) and another one diminishing with N (fast FIR). Hence, equations (37) and (39) have a minimum. The zeroes of the derivations of these functions provide the approximate value of the optimum block length :

(41)

$n \approx -0.6 + 0.7 \log_2 L$ for the FCMA1

(42)

$n \approx -0.9 + 0.7 \log_2 L$ for the FCMA2

Table 2 provides a comparison of the number of operation per output point required by the various algorithms for the approximate optimum blocksize given by (41) and (42). A reduction by a factor of 2 of the total number of operations is seen to be very easily obtained for filters longer than L=64, and a block length as small as N=8.

4.4. FFT-based implementation of FCMA

It is well known that the FFT can be used for a fast implementation of an FIR filter, through

the use of overlap-add or overlap-save techniques. Since (35) has the form of an FIR filter equation, the FFT technique can be applied. The main differences with the classical technique [4] are that the FFT length is twice the blocklength instead of twice the filter's length, and that sufficient care has been taken in the block formulation of the algorithm in order to maintain the rate of convergence of the CMA.

A simple way of understanding this method consists in extending the size of the block-Toeplitz matrix of eq. (35) in such a way that the resulting matrix is cyclic. The resulting equation is:

(43)

$$\begin{bmatrix} Y_n' \\ Y_n'' \end{bmatrix} = \begin{bmatrix} T(n) & T'(n) \\ T'(n) & T(n) \end{bmatrix} \begin{bmatrix} H_{n-N+1} \\ O \end{bmatrix}$$

where T (n) is the block-Toeplitz matrix of eq.(35), O a null vector of size N, Y_n'' a set of outputs that do not need to be computed (overlap-save technique).

T'(n) is chosen (44) in order to give the block-cyclic property to the above matrix:

(44)

$$T'(n) = \begin{bmatrix} A_{N-1}^t & A_0^t & \cdots & A_{N-3}^t & A_{N-2}^t \\ A_{2N-2}^t & A_{N-1}^t & \cdots & & A_{N-3}^t \\ \vdots & & & & \vdots \\ \vdots & & & & \vdots \\ A_{N+1}^t & & & & A_0^t \\ A_N^t & \cdots & & & A_{N-1}^t \end{bmatrix}$$

eq.(43) involves inner products of the form $A_i^t H^k$. Let us denote by $C_i(n)$ the matrix (2Nx2N) made from the i^{th} term of the blocks of matrix of eq.(43). When developing all inner products in terms of the individual components, (43) is rewritten as:

(45)

$$\begin{bmatrix} Y_n' \\ Y_n'' \end{bmatrix} = \sum_{i=0}^{(L/N)-1} C_i(n) \begin{bmatrix} \tilde{H}_{n-N+1}^i \\ O \end{bmatrix}$$

where

$$\tilde{H}^i_{n-N+1} = [\, h_{Ni} \quad h_{Ni+1} \quad \ldots\ldots \quad h_{Ni+N-2} \quad h_{Ni+N-1} \,]^t (n-N+1)$$
$$i = 0,1,2,\ldots\ldots,(L/N)-1$$

Each matrix $C_i(n)$ is cyclic, hence can be diagonalized by a Fourier matrix of size 2N:

(46)

$$\begin{bmatrix} Y'_n \\ Y''_n \end{bmatrix} = F_{2N}^{-1} \left[\sum_{i=0}^{(L/N)-1} \left[F_{2N}\, C_i(n) F_{2N}^{-1} \right] F_{2N} \begin{bmatrix} \tilde{H}^i_{n-N+1} \\ O \end{bmatrix} \right]$$

where

$$F_{2N}\, C_i(n) F_{2N}^{-1} = D_i(n)$$

is a diagonal matrix, whose elements are the DFT of the first column of $C_i(n)$.

Furthermore, it can easily be seen that $D_i(n)=D_{i-1}(n-N)$. This implies that eq.(46) represents 2N complex filters of length L/N in the Fourier domain, subsampled by a factor 1/N. The overall organization of this scheme is provided in Fig.3. It is seen to require (L/N)+2 FFT of length 2N per block of data of size N, plus 2N complex filters of length L/N which run at a rate divided by N.

The same kind of work has to be performed for the updating of the coefficients: eq.(36) is first extended to become block-cyclic. Considering separately each i^{th} term of the vector H^k and A_i results in the following set of equations:

(47)

$$\begin{bmatrix} \tilde{H}^i_{n+1} \\ O \end{bmatrix} = \begin{bmatrix} \tilde{H}^i_{n-N+1} \\ O \end{bmatrix} - W\, C_i^{\dagger}(n) \begin{bmatrix} \alpha_n \\ O \end{bmatrix}$$
$$i = 0,1,2,\ldots\ldots\ldots,(L/N)-1$$
$$W = \mathrm{Diag}\{1,1,\ldots\ldots,1,0,0,\ldots\ldots\ldots,0\}$$

and finally:

(48)

$$\begin{bmatrix} \tilde{H}^i_{n+1} \\ O \end{bmatrix} = \begin{bmatrix} \tilde{H}^i_{n-N+1} \\ O \end{bmatrix} - W\, F_{2N}^{-1}\, D_i^{*}(n) F_{2N} \begin{bmatrix} \alpha_n \\ O \end{bmatrix}$$
$$i = 0,1,2,\ldots\ldots\ldots,(L/N)-1$$

where $D_i^*(n)$ is the complex conjugate of the matrix $D_i(n)$.

Eq.(46),(48), together with (29) are seen to represent the FFT-based implementation of the FCMA. A precise count of the required number of operations is provided in Appendix A.2. and compared in table 2 for a number of filter lengths of interest. It is seen that this method requires the lowest number of multiplications among all considered methods for lengths greater than 32, and a lower number of operations (adds plus mults) above L=128. For a filter of length 512, the FFT-based implementation requires a number of operations divided by 4.5 compared to the initial algorithm. Note that this performance is obtained for quite small blocklengths. Table 2 makes the distinction between two versions of FFT-based CMA, depending on the algorithm chosen for the complex filters of length L/N, as seen in section 2.2.2. Note that the number of operations for both FFT-based algorithms are similar. Hence, the choice between them will rely on structural considerations.

V. SIMULATIONS

Some of these algorithms have been simulated, to verify our affirmations concerning:

a) The exact equivalence between the CMA and the FCMA,

b) the speed of our algorithm in relation to the initial one.

The complex input signal is (see Fig.1):

$x(n)=s(n)+s_M(n)+b(n)$

where $s(n)=\exp(j\ \phi(n))$ is the constant modulus transmitted signal ($|s(n)|=1$) and $j^2=-1$, $s_M(n)=\beta s(n-\tau)$ the signal due to the effects of multipath propagation and b(n) a zero-mean white noise. The objective is to use the CMA to provide an output y(n) which is an estimation of the transmitted signal s(n):

$y(n) = \hat{s}(n)$

Our aim here is not to concentrate on properties of the CMA itself, but to check the equivalence between the initial and fast versions of this algorithm. Fig.4 provides the convergence curve (J(n) averaged on 64 points) of both algorithms CMA1 and FCMA1 in the case of a complex FIR filter of length L=256 and a blocksize of N=32. These curves are

identical. As for the speeding up of the algorithm, we observed that the FCMA1 saved 40% computation time compared with the CMA1, which is nearly the ratio of the total number of operations, while exhibiting exactly the same convergence behaviour. This gives an indication on the accuracy issue of the FCMA: The updating of coefficients s_i being performed recursively, one may wonder if this recursivity could introduce any major drawback. We can show that in the LMS case, and for a fixed-point computation, this way of computing only results in a slight increase of the residual error. In any realistic case, these errors cannot result in an instability of the algorithm. The same demonstration holds for the FCMA case, and will not be repeated here, due to lack of space.

It is interesting to note that the matrix S depends only of the input signal, and some approximations are feasible [6], depending on some knowledge of the input signal properties.

VI. CONCLUSION

In this paper, we provided a new algorithm which allows for a reduction of arithmetic complexity of the CMA. This reduction is possible whatever the blocksize is, and even for the smallest blocklength (N=2).

Furthermore, we showed that it was also possible to work in the frequency-domain and that the obtained algorithm is strictly equivalent to the initial CMA.

All these algorithms share the same advantages: same convergence as CMA with a lower arithmetic complexity, and small blocksize. The small blocksize allows the memory requirements to remain reasonable, and reduces the overall system delay.

The algorithms based on short-length FIR algorithms are efficient for very small blocklengths, while FFT-based algorithms are more efficient for medium size ones.

Furthermore, there is a possibility that the convergence rate can be improved using the FFT. This work is under consideration and will be reported.

AKNOWLEDGMENTS

The authors would like to thank S. Mayrargue and O. Rioul from the CNET for careful reading of the manuscript.

APPENDIX A

Arithmetic complexity of the proposed algorithms for blocklengths $N=2^n$.

The precise evaluation is cumbersome, and we provide here only partial results that can easily be checked. The total number of operations is derived from these partial results.

A.1. Based on short-length FIR filters

This algorithm turns the fixed coefficient filtering (28) of length L into 3^n complex filters of length L/N, that are used for computing a block of N outputs. The precise number of real operations required by these filters depends on the chosen type of implementation (see section 2.2.2.). This transformation is obtained by linear combinations of subsampled input sequences (recursive application of (35))which cost a total of:

(49)

$4 (3^n - N)$ complex additions.

The second step is the correction of Y_n' in order to obtain Y_n. This first requires the computation of the elements of the matrix S(n), by application of eq.(38) and (39). This requires:

(50)

$2N (N - 1)$ complex multiplications,

$5N (N - 1) / 2$ complex additions.

Once S(n) is obtained, eq.(29) is solved by substitution, as explained in section 4.1., which requires a total of:

(51)

N real multiplications,

$N (N + 1)$ complex multiplications,

N^2 complex additions.

The final step is the updating of H to be used in the next block computation. This first requires the computation of $\underline{X}^{\dagger}(n)\, \alpha_n$, which is computed in the same manner as Y_n'. By taking

into account the fact that the combinations of the input samples need not to be computed again, this requires:

(52)

$2\ 3^n\ L/N$ complex multiplications,

$3^{n+1}\ L/N - 2L + 3^n - N$ complex additions.

Finally, once the impulse response is obtained, the linear combinations of the coefficients H^k need to be computed. This requires:

(53)

$3^n\ L/N - L$ complex adds,

and the final computation of (36) requires L more complex additions.

When the 4-mult 2-add complex multiplication sheme is used in the FIR filtering, the computation of a full block of outputs by the resulting algorithm (FCMA1) requires a total of:

(54)

$8\ 3^n\ L/N + N\ (\ 6N - 1\ )$ real mults,

$12\ 3^n\ L/N - 4L + 8\ 3^n + N\ (\ 7N - 15\ )$ real adds.

Finally, for the so-called FCMA2, where the complex filter scheme is that of Fig.2, we obtain the following number of operations (for a full block of outputs):

(55)

$6\ 3^n\ L/N + N\ (\ 6N - 1\ )$ real mults,

$12\ 3^n\ L/N - 4L + 12\ 3^n + N\ (\ 7N - 13\ ) - 2$ real adds.

A.2. FFT-based implementation

The overall organization of the algorithm is the same one as before, the differences being found in the complex filtering sheme and in the updating of the coefficients:

In fact, the FFT scheme transforms the length-L filter into 2N complex filters of length L/N, at the cost of L/N length-2N FFT's for computing the weights, one FFT for the determination of

$D_0(n)$ (note that $D_i(n)$, i=1,2,...,(L/N)-1, have already been computed, since $D_i(n)=D_{i-1}(n-N)$) and one length-2N inverse FFT for recovering the outputs.

As for the updating of the weights, the overall computation, as given in (48) requires (L/N)+1 length-2N FFT's, plus 2L complex multiplications and 3L complex additions.

Furthermore, let us assume that the FFT is computed using the split radix algorithm [10], we obtain as a result:

For the computation of a set of N outputs by the FFT-based FCMA1:

(56)

$4L(\log_2 N+2)+8L/N+6N^2+6N\log_2 N-13N+12$ real mults,

$2L(6\log_2 N+7)+8L/N+7N^2+18N\log_2 N-9N+12$ real adds,

and for the computation of a set of N outputs by the FFT-based FCMA2:

(57)

$4L(\log_2 N+1)+8L/N+6N^2+6N\log_2 N-13N+12$ real mults,

$2L(6\log_2 N+7)+8L/N+7N^2+18N\log_2 N-N+12$ real adds.

REFERENCES:

[1] D.N. Godard, "Self-recovering equalization and carrier tracking in two-dimensional data communication systems", IEEE Trans. COM-28, N°11, Nov. 1980, pp.1867-1875.

[2] J.R. Treichler and B.G. Agee, "A new approach to multipath correction of constant modulus signals", IEEE Trans. ASSP-31, N°2, April 1983, pp.459-471.

[3] J.R. Treichler and M.G. Larimore, "The tone capture properties of CMA-based interference suppresors", IEEE Trans. ASSP-33, N°4, Aug. 1985, pp.946-958.

[4] J.R. Treichler, S.L. Wood and M.G. Larimore, "Convergence rate limitations in certain frequency-domain adaptive filters", Proc. ICASSP-1989, pp.960-963.

[5] Z.J. Mou and P. Duhamel, "Short-length FIR filters and their use in fast FIR filtering", to appear, IEEE Trans. on ASSP, June 1991.

[6] J. Benesty and P. Duhamel, "A fast exact least mean square adaptive algorithm", Proc. ICASSP-1990, pp.1457-1460.

[7] R.E. Blahut, Fast Algorithms for Signal Processing, Addison-Wesley, Reading, MA, 1985.

[8] Z.J. Mou, P. Duhamel, "Fast FIR filtering: algorithms and implementation", Signal Processing, Dec. 1987, pp.377-384.

[9] A. Benveniste, M. Goursat and G. Ruget, "Robust identification of a nonminimum phase system: Blind adjustment of a linear equalizer in data communications", IEEE Trans. AC-25, N°3, June 1980, pp.385-399.

[10] P. Duhamel, "Implementation of split-radix FFT algorithm for complex, real and real-symmetric data", IEEE Trans. ASSP-34, N°2, April 1986, pp.285-295.

Figure captions

Fig.1: Overall organization of the CMA.

Fig.2: Efficient implementation of the complex filter found in the CMA in terms of real operations.

Fig.3: Implementation of the complex filter based on shorter FFT's.

Fig.4: Error curve of the CMA and the FCMA.

Table captions

Table 1: Comparison of the arithmetic complexity per output point of the CMA and the FCMA for a blocksize of N=2.

Table 2: Comparison of the number of operations per output point required by the various algorithms.

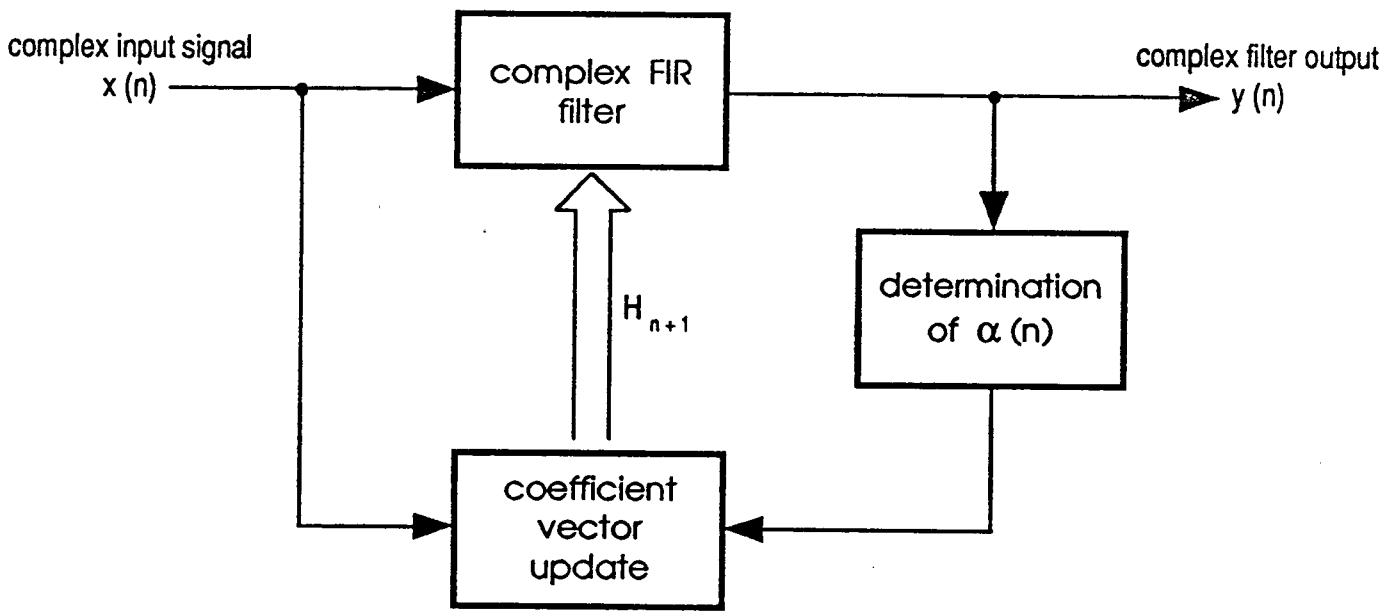


Fig. 1 : The Constant Modulus Algorithm.

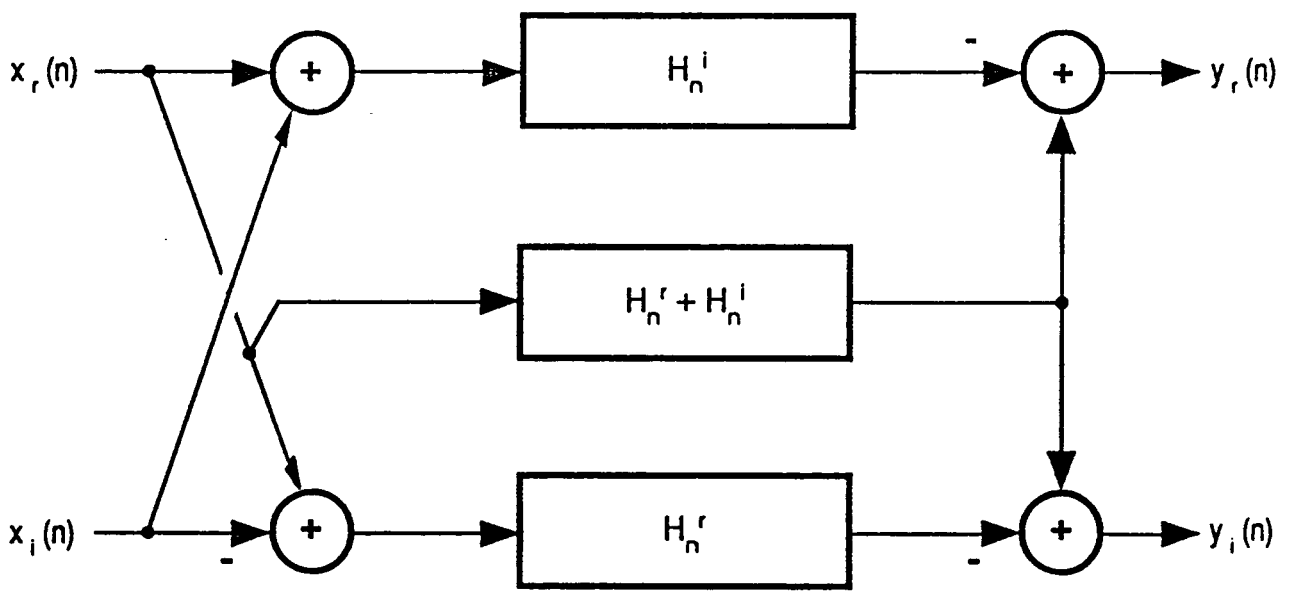


Fig. 2 : Complex FIR filtering using three real FIR filters.

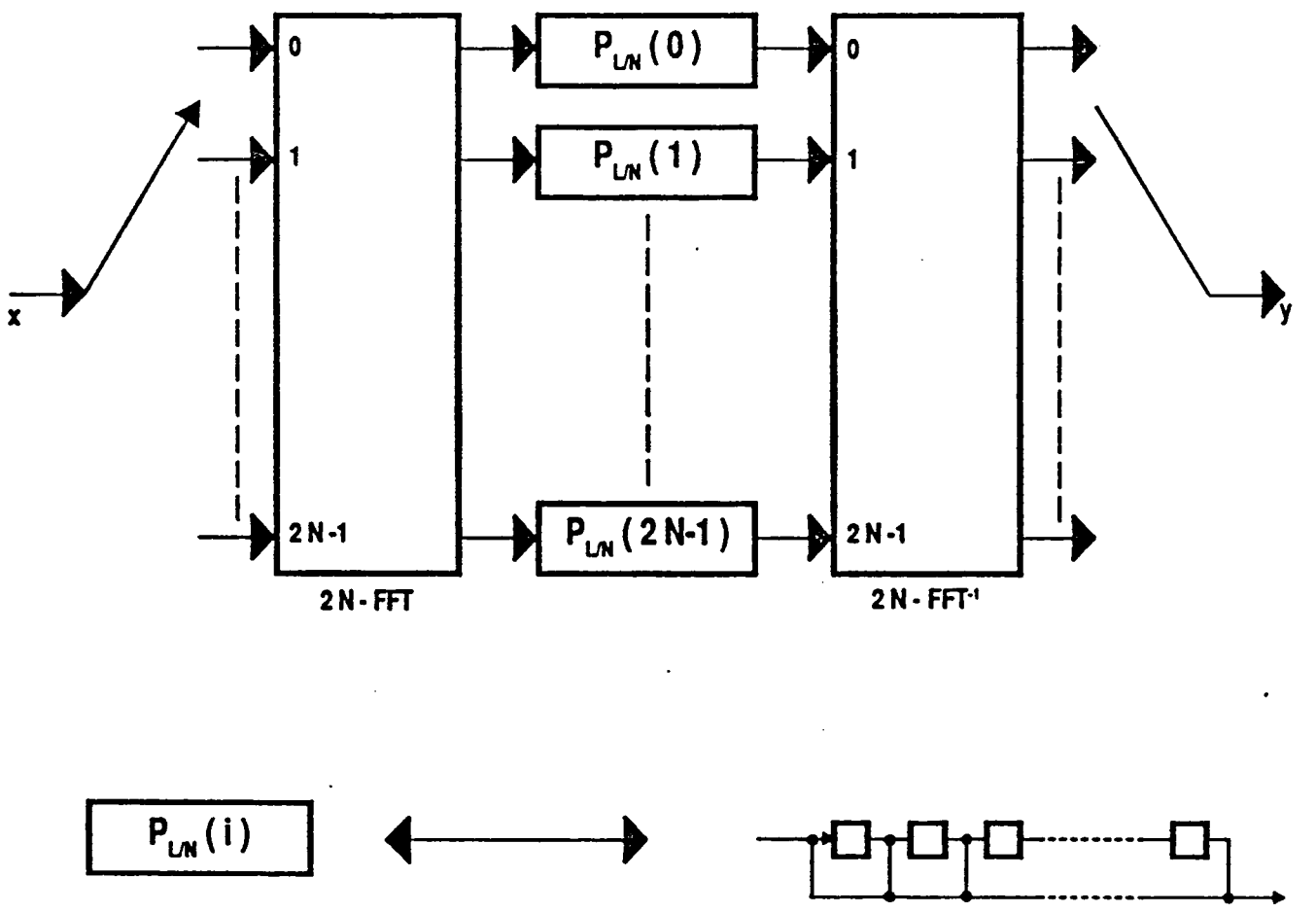
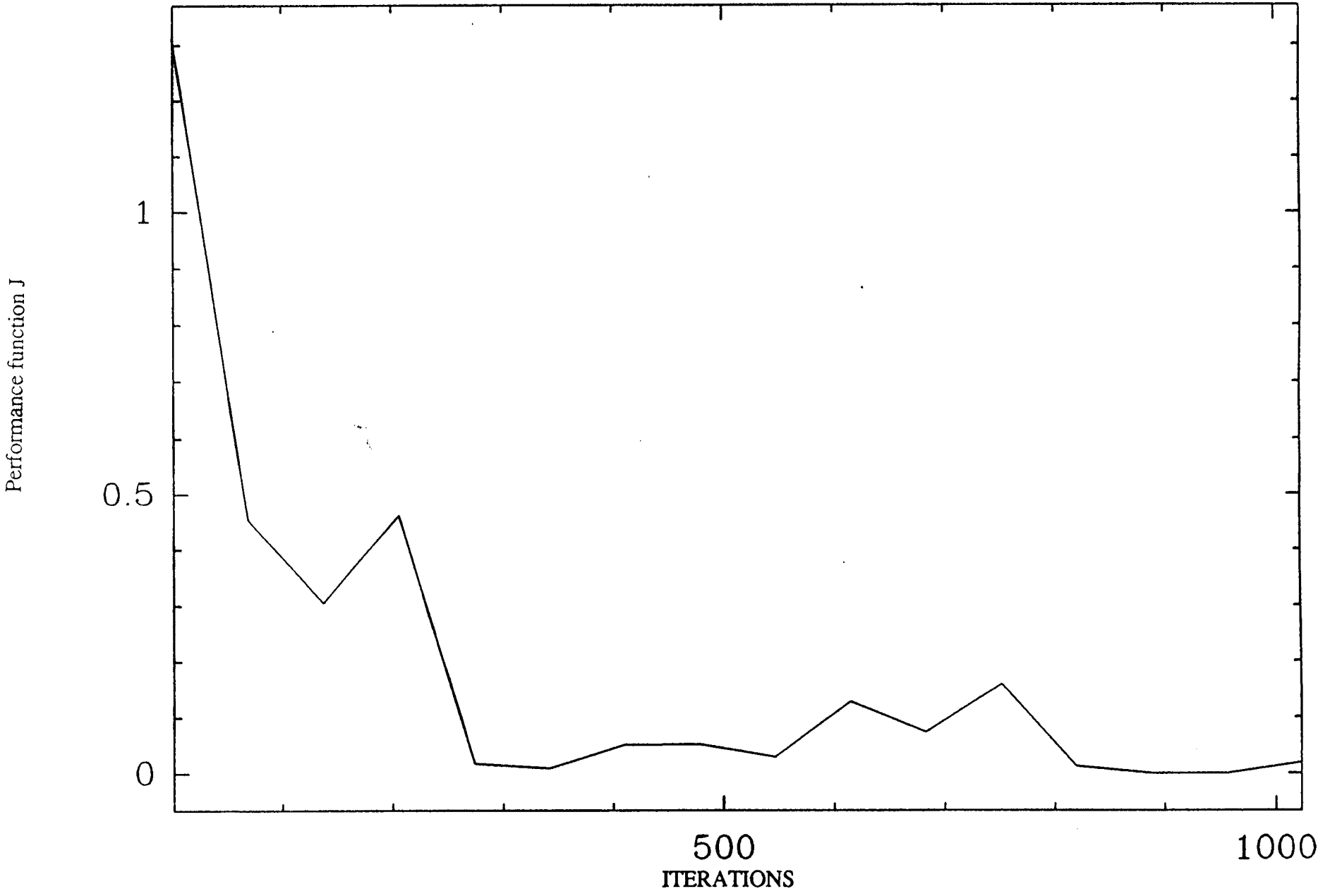


Fig. 3 : Shorter FFT - based FIR filtering scheme.



Benesty 16-MAY-1990

Fig.4 Error curve of the CMA and the FCMA

Algorithm / Number of operations	CMA1	CMA2	FCMA1	FCMA2
Number of additions	8 L	8 L + 4	7 L + 11	7 L + 18
Number of multiplications	8 L + 5	6 L + 5	6 L + 11	4,5 L + 11
Total number of operations	16 L + 5	14 L + 9	13 L + 22	11,5 L + 29

Table 1 : Comparison of the number of operations per output point of the CMA and the FCMA for a blocksize of N = 2

	CMA1		CMA2		FCMA1			FCMA2			FFT - CMA1			FFT-CMA2		
filter length L	number of additions	number of multiplications	number of additions	number of multiplications	block length N	number of additions	number of multiplications	block length N	number of additions	number of multiplications	block length N	number of additions	number of multiplications	block length N	number of additions	number of multiplications
32	256	261	260	197	8	214	155	8	229	128	8	275	143	8	283	127
64	512	517	516	389	8	346	263	8	361	209	16	378	206	16	386	190
128	1024	1029	1028	773	16	591	419	16	614	338	16	628	304	16	636	272
256	2048	2053	2052	1541	32	967	677	32	999	555	32	839	435	32	847	403
512	4096	4101	4100	3077	64	1586	1112	32	1696	920	64	1164	664	64	1172	632

Table 2 : Comparison of the number of operations per output point required by the various algorithms.