TP Matlab et filtrage adaptatif

Etude des algorithmes LMS et RLS appliquées à un problème de soustraction de bruit

Sommaire :

I- Introduction. 2

II- Acquisition des signaux. 3

III- Algorithme LMS. 6

IV-Algorithme RLS. 28

V. Filtrage optimal au sens des moindres carrés. 37

VI. Étude dans le cas d’un système non stationnaire. 41

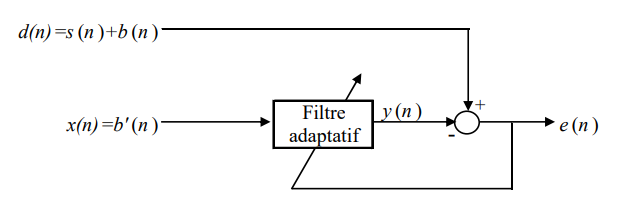
VII- Bibliographie 52

VIII- Annexe 53

I- Introduction.

L’objectif de ce TP est d’implanter et d’expérimenter sur Matlab les algorithmes LMS et RLS en les appliquant à un problème de soustraction de bruit.

On dispose d’un signal de parole perturbé par un cri d’insecte, en l’occurrence une decticelle bariolée (fichier parole\_bruitee.mat). On dispose également du signal de l’insecte seul, pris par un deuxième microphone, situé tout près de celui-ci (fichier decticelle.mat). On rappelle ci-dessous le schéma de principe de la soustraction de bruit :



Dans le problème qui nous occupe, le signal d(n) correspond au signal de parole bruité et le signal x (n)= b′(n) correspond au signal enregistré par le deuxième microphone, situé à proximité de la decticelle. Ce signal b′(n) est corrélé au signal de bruit additif b(n) qui vient entacher le signal de parole seul (signal s(n) ). L’objectif de ce dispositif est de produire une sortie y(n) qui ressemble le plus possible à b(n) , de façon à ce que le signal e(n) corresponde au plus près au signal de parole non bruité s(n) .

II- Acquisition des signaux.

1)Dans un premier temps, nous avons chargez les signaux à l’aide de la commande load:

|  |
| --- |

On obtient alors nos deux signaux chargés dans l’environnement sous les noms “d” et “x” avec lesquels nous allons travailler.

On vérifie bien cela dans le workspace:

|  |
| --- |

On observe donc que nos signaux sont deux vecteurs de taille 25 001x1.

2) Après avoir réalisé notre première étape, nous allons observer nos deux signaux à l'aide de la commande plot.

|  |
| --- |

On obtient alors les figures suivantes:

| signal d , la parole bruitée. | signal x, decticelle |
| --- | --- |
|  |  |

On observe graphiquement que nos signaux ont des grandes similitudes ce qui est cohérent avec le fait que notre signal parole bruité soit une addition de la voix et du signal decticelle.

Pour conclure notre étude préliminaire sur nos signaux, nous allons écouter nos signaux. Pour cela, nous allons utiliser la commande soundsc(). On ne précise pas la fréquence Fe lors de l'appel de la commande. Cependant, la fréquence Fe est égale à 8192 Hz.

A l'écoute, on distingue mal ce que la personne veut dire dans le signal bruité (d). Le son du decticelle empêche cette bonne compréhension.

Le but du TP est de remédier à ce problème en supprimant le bruit du decticelle afin d’avoir une parole qui soit audible.

III- Algorithme LMS.

On sait que les signaux obtenus sont non stationnaires. Cela signifie que le filtrage optimal doit être variable et fonction du temps. C’est pourquoi, nous pouvons appliquer le filtre de Kalman ou les algorithmes adaptatifs du type LMS et RLS.

Nous allons appliquer dans un premier temps l’algorithme LMS (Least Mean Square).

C’est un algorithme de la famille des gradient stochastiques. Ils donnent une approximation du gradient à partir des données disponibles.

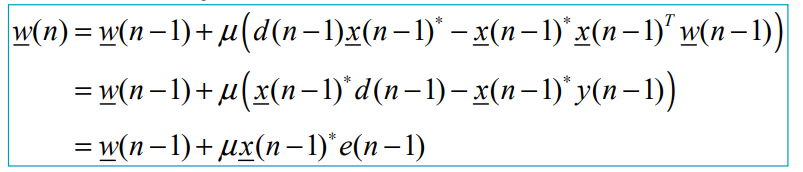
Autrement dit, les filtres adaptatifs sont des systèmes capables de mettre à jour leurs paramètres (coefficients du filtre) en fonction des propriétés de l’environnement qui peuvent elles-mêmes varier, afin de satisfaire un objectif donné. Un algorithme adaptatif part de conditions initiales prédéterminées et met à jour de façon récursive les coefficients du filtre pour s’adapter au processus.

Pour débuter, nous avons créé une fonction *algo\_LMS* dans matlab afin d’y implémenter l’algorithme de notre filtre LMS. Notre fonction prend en paramètre deux vecteurs *x* et *d* contenant tous les échantillons de nos signaux *d(n)* et *x(n)* ainsi que le nombre de coefficients *P* de la réponse impulsionnelle (RI) du filtre FIR (que nous allons mettre à jour à chaque instant) ainsi que le pas d’adaptation *mu* qui sera constant pour cette version de notre algorithme.

Notre fonction nous retourne alors en sortie le vecteur *e* contenant tous les échantillons de l’erreur *e(n)* ainsi que notre matrice *w* contenant la suite des filtres obtenus.

(coefficients du filtre *w(n)* au cours du temps).

Pour constituer notre algorithme, on utilise la formule de mise à jour du filtre. On regarde dans le cours, on obtient alors cette formule.



Pour effectuer cette fonction, on a recours aux formules de la sortie du filtre ainsi que l’erreur d’estimation.

|  |
| --- |

On remarque que l’erreur d’estimation associée est une sortie de notre algorithme.

On constate que pour obtenir le vecteur *w* a l’instant n , on utilise la valeur de n-1. On peut donc conclure que n devra être supérieur à 1. De plus, on sait que dans Matlab l’indice du premier élément du vecteur est égale à 1 . On doit donc prendre en compte cela, n vaudra ainsi 2.

Nous initialisons notre filtre à 0 (coefficient nul) pour w(0).

Après avoir fait les prérequis nécessaires, on réalise une boucle for pour calculer à plusieurs instant n.

Premièrement, on va calculer le vecteur x(n-1). On sait que le vecteur x(n) vaut:

|  |
| --- |

On constate que notre vecteur x(n) est de taille P. On remarque aussi que nos éléments dans le vecteur sont de plus en plus petits.

On modifie alors notre formule de notre vecteur x(n) afin d’avoir x(n-1). On obtient alors:

| x(n-1)=[x(n-1), x(n-2), …, x(n-P+1),x(n-P)] |
| --- |

Après avoir calculer x(n-1), on va pouvoir calculer la sortie y(n-1) du filtre à partir du vecteur X des coefficients du filtre w(n-1). Par la suite, on peut calculer l’erreur e(n-1) à chaque instant n.

Enfin, nous mettons à jour les coefficient du filtre w(n).

À la fin de la boucle, on obtient le vecteur e contenant tous les échantillons de l’erreur e(n) ainsi que notre matrice w contenant la suite des filtres obtenus (coefficients du filtre w(n) au cours du temps).

|  |
| --- |

2) Ensuite, nous allons tester notre algorithme de notre filtre LMS. Pour cela, on crée un programme principal *soustraction\_bruit.m* pour tester votre fonction *algo\_LMS*. On utilise un pas *µ* de 10^-10 et une longueur de filtre *P* égale à 3.

Dans notre programme *soustraction\_bruit*, on appelle notre fonction précédemment créée. On passe en paramètre notre deux vecteurs x et d ainsi que notre P qui est égale à 3 et notre pas mu de 10^-10.

Suite à cet appel de fonction, on peut afficher à l'aide de la commande plot notre erreur e(n-1) à chaque instant.

Dans une deuxième figure, on affiche à l'aide de la commande plot l’évolution des coefficients du filtre w(n) au cours du temps. On utilise aussi la commande légende afin de pouvoir légender nos différentes courbes.

On souhaite maintenant connaître l’évolution de l’erreur sur les coefficients du filtre:

|  |
| --- |

On sait que le filtre optimal est donné par:

|  |
| --- |

Ce vecteur ne varie pas dans le temps.

On ajoute donc dans notre programme notre vecteur wopt. Ensuite, on peut réaliser notre calcul de l’erreur sur les coefficients. Après, on réutilise les commandes plot et légendes afin d’afficher dans une troisième figure nos résultats.

Pour finir, on va calculer l’évolution de la norme 2 au carré de l’erreur sur le filtre. On calcule la somme de nos coefficients de l’erreur au carré. On affiche dans une quatrième figure à l'aide de la commande plot.

|  |
| --- |

On obtient donc pour le plot de e(n):

|  |
| --- |

|  |  |
| --- | --- |

On observe notre courbe bleu qui converge vers la valeur 1. De plus, notre courbe rouge converge vers 0.5 et notre courbe jaune converge vers 0.25. Or on sait que notre filtre optimal a pour valeur [1, 1/2 , 1/4]. On retrouve bien nos valeurs sur la courbe de notre filtre.

| évolution de l’erreur sur les coefficients du filtre | évolution de la norme 2 au carré de l’erreur du filtre |
| --- | --- |
|  |  |

On observe que l’erreur est décroissante dans le temps. On remarque que cette dernière est relativement lente. Ce qui est en lien avec la convergence vers le filtre optimal de notre signal.

Pour terminer, on écoute notre nouveau signal filtré. On s'aperçoit qu’au début de notre audio, on entend toujours le bruit mais il s'atténue / est très léger. Ensuite, on entend le son de la voix de l’homme dans la suite de l’audio. Cela est dû au temps d'adaptation du filtre. Ce qui est cohérent avec la convergence des filtres adaptatifs.

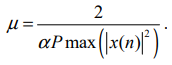
3) On étudie à présent l’influence du pas mu sur la convergence de l’algorithme pour un nombre P de coefficients fixé. Ici P=3.

Dans un premier temps, on commence par modifier notre pas mu dans notre algorithme. Notre pas mu suit la formule suivante:



Or on a vu précédemment que x(n) n’est pas stationnaire donc Rxx(0) n’existe pas. De plus, la valeur absolue au carré du signal x(n) est d’amplitude très variable selon la valeur de n. Le paramètre μ contrôle l’importance de la correction apportée au vecteur de coefficients W lors de la kième itération.

On utilise donc une formule plus adaptée à notre problème. Il convient donc de choisir le pas d’adaptation correspondant au cas le plus contraignant. Notre pas mu s’exprime donc de la manière suivante:



Nous allons tester notre programme avec différentes valeurs d' alpha. On pourra ainsi déterminer la valeur minimale de alpha pour laquelle l’algorithme converge.  
On teste notre programme avec alpha qui vaut 0.1, 0.5, 1, 2, 3, 30.

On modifie donc notre code dans le programme *soustraction\_bruit* pour effectuer notre modification du pas mu:

|  |
| --- |

On utilise les fonctions abs() pour réaliser prendre la valeur absolue. De plus, on utilise aussi la fonction max() afin de prendre le maximum de notre signal.

Dans un premier temps, on teste pour **alpha=0.1**. On obtient:

**Pour alpha=0.1 :**

|  |
| --- |
|  |

**Pour alpha=0.2 :**

|  |
| --- |
|  |

**Pour alpha=1 :**

|  |
| --- |
|  |

**Pour alpha = 2 :**

|  |
| --- |
|  |

**Pour alpha = 3 :**

|  |
| --- |
|  |

**Pour alpha=30:**

|  |
| --- |
|  |

On remarque que les courbes pour alpha = 0.1 et alpha=0.2 ne converge pas le filtre optimal.Avec l’interprétation des courbes, dire que la valeur minimale d’alpha pour laquelle l’algorithme converge est 1.On remarque que plus on augmente la valeur de l'alpha, plus notre pas est faible.

On peut observer que selon la valeur d’alpha, la vitesse de convergence est différente. Plus l’alpha est élevé, plus la convergence est lente. De plus, on remarque que plus l’alpha est petit, plus il y a de fluctuations autour de la trajectoire moyenne.

Pour conclure, on peut interpréter que le choix d'**alpha=2** est le meilleur choix pour obtenir le meilleur compromis entre la vitesse de convergence et l’erreur résiduelle.

Après avoir réalisé nos tests sur alpha, nous allons modifier le pas mu. Le pas mu sera variable et décroissant . Nous allons donc créer une nouvelle fonction *algo\_LMS\_dec*.

Cette fonction a le même principe que la fonction *algo\_LMS*. Cependant, notre nouvelle fonction a un paramètre d’entrée modifié et une sortie en plus. Le paramètre d’entrée *mu\_init* correspondra au pas initial, et le vecteur de sortie mu contiendra la suite des pas d’adaptation.On calcule le pas mu a partir de la formule suivante:



On peut voir que notre fonction prend en paramètre un pas initial en entrée que l’on vient modifier pour chaque n.

On obtient alors :

|  |
| --- |

On initialise *mu\_init* à 10^-9 dans notre fichier *soustraction\_bruit2.m*. On appelle notre nouvelle fonction dans ce fichier.

|  |
| --- |

On obtient alors:

| évolution des coefficients du filtre w(n) au cours du temps | évolution de l’erreur sur les coefficients du filtre |
| --- | --- |
|  |  |
| evolution de la norme 2 au carré de l’erreur | évolution du pas mu |
|  |  |

On observe sur la courbe de l’évolution de *mu*, la valeur de mu diminue conformément à ce qui est attendu. Il est donc grand au départ et diminue. On peut mettre ce résultat en corrélation avec ce que l’on peut observer sur la courbe des évolutions des coefficients. On observe que les fluctuations sont légères au cours du temps. La modification de *mu* permet de limiter nos fluctuations.

De plus, on observe sur cette même figure que nos courbes convergent bien vers notre filtre optimal. On constate aussi que la convergence est très rapide. Ce qui donne de la pertinence a cette méthode.

On s'intéresse à la courbe de l’évolution de la norme 2 des erreurs. On remarque que l’erreur est minime et converge vers 0 très rapidement aussi.

Pour conclure, on écoute notre signal de sortie *e(n)*. On se rend compte que le son est plus agréable à l'écoute et le bruit de l’insecte a quasi-disparu. De plus, contrairement à un *mu* constant, au début de notre écoute, nous n’entendons plus le bruit de l’insecte. On a donc réussi à rendre l'adaptation beaucoup plus rapide et pertinente pour le filtrage de nos signal.

4)On modifie maintenant le coefficient *P* qui était jusqu’à présent fixé à 3. On teste ainsi sur les fonctions que nous avons créées. Dans un premier temps, nous mettons notre valeur de P à 2. Dans le second, on met la valeur de P à 5. On obtient donc de nouveaux résultats qui sont les suivants :

**Pour l’algorithme *algo\_LMS* :**

**P=2:**

| évolution des coefficients du filtre w(n) au cours du temps | Signal d’erreur e(n) |
| --- | --- |
|  |  |

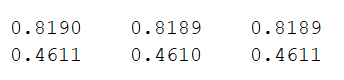
Lorsqu’on écoute le signal de l’erreur correspondant, on remarque que le son est toujours marqué par le son des insectes.

Dans le premier graphique des coefficients, on constate que ceux-ci convergent assez rapidement vers des valeurs qui sont proches des coefficients optimums qui composent le vecteur Wopt. Cependant on constate que ceux-ci oscillent pas autour des coefficients optimaux. En effet, W(0) = 0.8,W(1) = 0.5.

De plus, la précision des coefficients n’est pas assurée, car au fur et à mesure du temps la valeur des coefficients oscille entre + et - 0.5.

On peut voir sur le signal d’erreur e(n) que le bruit de fond des decticelles est encore présent mais atténué.

On observe que la convergence n’est pas excellente. On constate conformément à P=2 qu’il y a bien deux courbes. On observe aussi qu’un des coefficients converge vers 0.8. Cette valeur est loin de celle du filtre optimal.



**P=5**

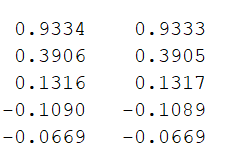
| évolution des coefficients du filtre w(n) au cours du temps | Signal d’erreur e(n) |
| --- | --- |
|  |  |

Après avoir écouté le signal de l’erreur, on remarque que le bruit des insectes a fortement diminué. Cependant, nous le distinguons toujours.

On remarque que la convergence devient beaucoup plus compliquée et plus lente pour l’algorithme LMS lorsque P = 5.

La convergence de cet algorithme est donc un compromis entre vitesse et fidélité. De plus, ces deux facteurs sont impactés par l’ordre P du filtre.

Le signal d’erreur e(n), nous fournit l’information complémentaire concernant le grésillement des insectes qui est très réduit, mais existe toujours, le filtre a donc grandement réduit le bruit des decticelles mais pas assez. Cela s’explique aussi par le fait que les coefficients du filtre sont encore loin de la solution optimale.

****

**Pour l’algorithme *algo\_LMS\_dec* :**

**P=2:**

| évolution des coefficients du filtre w(n) au cours du temps | Signal d’erreur e(n) |
| --- | --- |
|  |  |

On voit que les coefficients des filtres varient autour de deux valeurs qui sont 0.8 et 0.5 avec de grandes imprécisions.

Lorsqu’on écoute le signal d’erreur, on entend toujours le bruit des insectes.

On constate que le filtre LMS\_dec lorsque l’on a P=2, les coefficients ont du mal à converger sur toute la durée du filtrage ainsi, on est loin d’obtenir un filtrage qui est optimal. Cela se traduit par le signal d’erreur qui comporte encore le bruit de fond qu’on voit lorsqu’il n’y a pas de pics.

**P=5**

| évolution des coefficients du filtre w(n) au cours du temps | Signal d’erreur e(n) |
| --- | --- |
|  |  |

On remarque que chaque coefficient du filtre se stabilise autour d’une valeur. Ces dernières sont très proches des valeurs optimum souhaitées. Sur le signal d’erreur e(n), on constate qu’il n’y a pratiquement pas de bruit diffus dans le fond qui est introduit par les bruits des decticelles.

Cela se traduit à l'écoute, on se rend compte que le bruit des insectes a été presque entièrement supprimé. En effet, lorsqu’on monte le volume on se rend compte qu’on distingue toujours le bruit parasite des insectes mais cela est très léger. D’autant plus que, pour une écoute avec un volume de son correct, nous ne l’entendons pas.

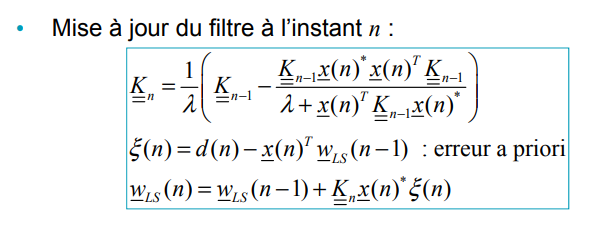
IV) Algorithme RLS

L’objectif de cette partie est de traiter nos signaux initiaux avec un second algorithme afin de les comparer. On utilise un algorithme adaptatif RLS.

Cet algorithme consiste à mettre à jour la solution au cours du temps, c’est-à-dire à déterminer W\_ls(n).

1)Pour cela, on crée une fonction *algo\_RLS* implémentant notre algorithme RLS.La fonction fournira en sortie le vecteur e contenant les échantillons de l’erreur et la matrice ainsi que la matrice w contenant la suite des filtres obtenus. Nous souhaitons minimiser la fonction de coût J(e(n)).

Cette fonction prend en paramètre, nos deux signaux *x* et *d* ainsi que *P* le nombre de coefficients P de la réponse impulsionnelle du filtre FIR. De plus, cette fonction prend en paramètre un facteur d’oubli lambda et un talon delta servant d’initialisation pour notre algorithme.



On doit donc initialiser la matrice K. Pour K(0), on doit donc initialiser selon la valeur suivante:



Dans matlab, on utilise la fonction eye() afin de créer la matrice identité. Cette matrice est de taille P.

On a alors établi le code ci-dessous :

|  |
| --- |

2) Dans un premier temps, on fixe la valeur du talon δ à 0.01

On observe l’influence du facteur d’oubli λ quand il vaut 1. Un facteur d’oubli est introduit afin d’atténuer l’influence des données les plus anciennes et ainsi rendre l’algorithme adaptatif. Dans la version adaptative de l’algorithme RLS, le facteur d’oubli λ vérifie:

0<λ< 1.

**Pour λ = 1**

| signal d’erreur e(n). | évolution des coefficients du filtre W. |
| --- | --- |
|  |  |
| évolution de l’erreur sur les coefficients du  filtre |W-Wopt| | évolution de la norme 2 au carré de  l’erreur sur le filtre. |
|  |  |

**Observation :** On constate que les coefficients du filtre converge quasi-instantanément vers les valeurs du filtre optimal, cela est normal, car à λ = 1 , le filtre RLS est optimal. Aussi ce dernier est optimal, car le signal est stationnaire. Lors de l’écoute, on s'aperçoit que le bruit des decticelles a disparu, ce qui est cohérent avec les analyses précédentes.

**Pour λ =0.95**

| signal d’erreur e(n). | évolution des coefficients du filtre W. |
| --- | --- |
|  |  |
| évolution de l’erreur sur les coefficients du  filtre |W-Wopt| | évolution de la norme 2 au carré de  l’erreur sur le filtre. |
|  |  |

**Observations:**

On observe que les coefficients de notre filtre convergent vers notre filtre optimal. Cependant, on observe au cours de sa convergence est très irrégulière. Elles oscillent énormément. Cela est dû au fait que le filtre s’adapte quand l’interlocuteur parle. Le filtre converge très rapidement et globalement devrait permettre de grandement réduire le bruit des decticelles. On peut faiblement distinguer des grésillements introduits dans la voix de l’interlocuteur. Aussi on n’entend plus le bruit des decticelles.

**Pour λ =0.9**

| signal d’erreur e(n). | évolution des coefficients du filtre W. |
| --- | --- |
|  |  |
| évolution de l’erreur sur les coefficients du  filtre |W-Wopt| | évolution de la norme 2 au carré de  l’erreur sur le filtre. |
|  |  |

**Observations :** De la même manière que pour le cas avec un 𝛌 = 0.95, on retrouve des coefficients qui convergent vers les valeurs optimal du filtre. Cependant elles oscillent grandement quand l'interlocuteur parle. Cela se traduit par un signal d’erreur qui va être perturbé lorsque que la personne parle et c’est normal. A L'écoute on n'entend plus le bruit des decticelles, mais les grésillement sont un peu plus marqués que pour le cas précédent.

**Conclusion :**

C’est donc avec l’influence du facteur d’oubli λ qui vaut 1 que nous obtenons les meilleurs résultats. On a une qualité de son qui est bien supérieure à celle des filtres avec un facteur d’oubli qui est inférieur à 1.

Nous allons maintenant étudier l’influence du talon δ. Nous fixons λ à 1 et nous faisons varier δ entre les valeurs suivantes [0.1; 0.01; 0.01 ]

**Pour δ=0.1 :**

| signal d’erreur e(n). | évolution des coefficients du filtre W. |
| --- | --- |
|  |  |
| évolution de l’erreur sur les coefficients du  filtre |W-Wopt| | évolution de la norme 2 au carré de  l’erreur sur le filtre. |
|  |  |

**Pour δ=0.01 :**

| signal d’erreur e(n). | évolution des coefficients du filtre W. |
| --- | --- |
|  |  |
| évolution de l’erreur sur les coefficients du  filtre |W-Wopt| | évolution de la norme 2 au carré de  l’erreur sur le filtre. |
|  |  |

**Pour δ=0.001 :**

| signal d’erreur e(n). | évolution des coefficients du filtre W. |
| --- | --- |
|  |  |
| évolution de l’erreur sur les coefficients du  filtre |W-Wopt| | évolution de la norme 2 au carré de  l’erreur sur le filtre. |
|  |  |

**Observations :**

On observe la même chose pour toute les variations du talon δ. On peut donc conclure que la variation de δ n’a pas d’influence sur les résultats.

**Conclusion:**

Le filtre avec un facteur d’oubli lambda égale à 1 est le plus optimum. En effet, le converge très rapidement et les coefficients sont les coefficients optimums. Cela s’explique par le fait que le filtre agît sur tous les systèmes, or on agît dans un environnement stationnaire. Donc, on n’a pas besoin d'adapter le filtre au fur et à mesure. On constate aussi que la variation de ẟ, le talon a une influence négligeable sur le cas avec facteur d’oubli égale à 1.

Le filtre optimal est donc celui avec un facteur d’oubli égale à 1.

En comparant les deux algorithmes LMS et RLS, on constate que l’algorithme RLS converge plus rapidement vers les vraies valeurs, l’algorithme LMS converge mais lentement à cause de sa dépendance au pas µ.

**V. Filtrage optimal au sens des moindres carrés.**

Dans cette partie, nous faisons l'hypothèse que les signaux sont stationnaires et ergodiques. On peut estimer les coefficients rxx (k) et rdx (k) en effectuant une moyenne temporelle à la place de la moyenne statistique.

1) Ces estimées temporelles d’auto- et d’inter-corrélations seront obtenues à l’aide de la fonction xcorr de Matlab.  
On crée un programme avec les commandes suivantes:

|  |
| --- |

On obtient alors deux vecteurs colonnes rxx et rdx. Ces vecteurs contiennent les estimées de coefficients rxx(k) et rdx(k) de la manière suivante:

|  |
| --- |

On obtient donc dans notre workspace:

|  |
| --- |

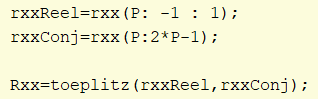
A partir du vecteur rxx, on cherche à obtenir la matrice Rxx qui est l’estimé de la matrice d'autocorrélation . La matrice Rxx est de la forme:



avec : 

On va donc prendre le vecteur rxx et le découper en partie réelle et en partie complexe. Suite à cela, on utilise la fonction toeplitz.

On obtient alors:



On réalise ensuite ,à partir du vecteur rdx précédemment obtenu, le vecteur Rdx. La valeur estimé de ce vecteur s'écrit de la forme:



On obtient alors:



2) Dans cette partie, on cherche à déterminer le filtre optimal au sens des moindres carrés Wls (vecteur).On le comparera par la suite avec le filtre optimal wopt. Le filtre wopt est de la forme:



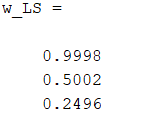
Pour réaliser notre calcul du filtre, nous allons utiliser la formule du cours suivante.



Notre code ressemblera donc à cela:



On obtient alors le résultat suivant:



On observe qu'à l'issue de ces résultats, on retrouve des valeurs très proches au millième près du filtre optimal. Cependant, on ne peut pas garantir que nous retrouvons bien le filtre optimal.

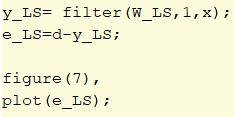
3) Après avoir obtenu notre filtre w\_LS, nous allons l’utiliser afin de filtrer avec notre signal x. On utilise pour cela la fonction filter. On obtient alors un signal filtré y LS.

Ensuite, on calcul e\_lS. On utilise la formule suivante pour le calculer e :



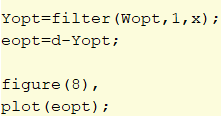
On trace à l'aide de la commande plot notre signal e. On écoute ensuite avec la commande soundsc notre signal e.

On obtient alors le code suivant:



On veut ensuite le comparer au signal e\_opt. Pour cela, on calcule y\_opt avec notre filtre optimal w\_opt de la même manière que précédemment avec la commande filter. Par la suite, on calcule e\_opt avec la formule suivante:





En exécutant le code, on obtient les résultats suivants:

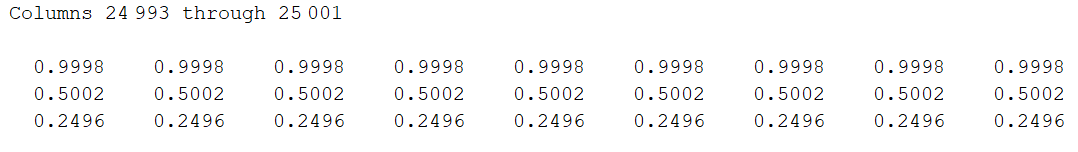
| eopt | e\_LS |
| --- | --- |
|  |  |

On observe que nous avons des courbes presque similaires à l'œil. Ce qui est en adéquation avec les valeurs de w\_LS et w\_opt qui sont presque identiques.

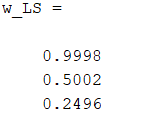
Lorsqu’on écoute, on remarque aucune différence. De plus, on observe que les sons sont bien filtrés, on n’entends plus les bruits de l’insecte. Aussi, il n'y a aucun grésillement.

4) Dans cette dernière question, on cherche a comparer notre filtre w\_LS obtenu avec l’algorithme des moindres carrés et le filtre obtenu avec l’algorithme RLS pour lambda qui est égale à 1.

Voici les coefficients obtenus avec notre filtre RLS avec labda valant 1.



Les coefficients de notre filtre w\_LS est représenté par:

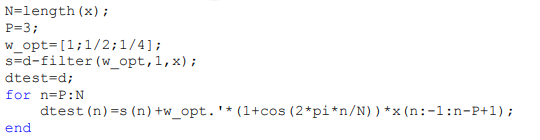


On observe bien que les résultats des coefficients sont identiques. De plus, on sait que l’algorithme RLS converge en moyenne vers la solution des moindres carrés. Ce résultat est cohérent avec ce que l’on observe.

VI. Etude dans le cas d’un système non stationnaire

On souhaite maintenant étudier le comportement des algorithmes déjà développés dans le cas où une non-stationnarité lente est introduite dans le système.

Dans un premier temps, nous allons modifier notre signal d en un signal *d\_test*. Ce signal *d\_test* permettra d’avoir une non stationnarité. Pour cela, on définit le signal *d\_test* de la manière suivante:



Pour la suite du tp, on réutilise nos fonctions déjà développées. On y mettra en paramètre, P=3, x, d\_test, mu= 10^-10 , mu\_init = 10^-9 , alpha=2.

1)Après avoir obtenu d\_test, on décide de tracer le signal dans une figure. On décide aussi de l’écouter avec la commande soundsc. On le compare ensuite au signal d.

| Signal d\_test | Signal d |
| --- | --- |
|  |  |

On observe bien que le signal *d* a été modifié. On constate que le signal *d\_test* a une amplitude modifiée. Ce qui est logique avec la modification du signal avec un cosinus.

2) On utilise donc maintenant nos fonctions déjà développées. On appelle notre fonction algo\_LMS. On lui passe en paramètre x, d\_test, P=3 et mu=10^-10. Dans un second temps, on appelle notre fonction algo\_LMS\_dec. On lui passe en argument x, d\_test, P=3 et mu\_init=10^-9. .

Par la suite, on affiche dans divers figure le signal d’erreur e(n) et l’évolution des coefficient du filtre w(n) au cours du temps.

Pour finir on écoute ensuite nos deux signaux e(n) en provenance de nos deux fonctions.

On observe alors après l'exécution de notre code:

Pour l’algorithme *algo\_LMS* :

| signal d’erreur e(n). | évolution des coefficients du filtre W. |
| --- | --- |
|  |  |
| évolution de la norme 2 au carré de  l’erreur sur le filtre. | évolution de l’erreur sur les coefficients du  filtre |W-Wopt| |
|  |  |

Pour l’algorithme *algo\_LMS\_dec* :

| signal d’erreur e(n). | évolution des coefficients du filtre W. |
| --- | --- |
|  |  |
| évolution de la norme 2 au carré de  l’erreur sur le filtre. | évolution de l’erreur sur les coefficients du  filtre |W-Wopt| |
|  |  |

**Observations :**

On constate que le signal e(n) présente encore des bruits à l'issue du filtrage avec un signal non stationnaire. A l’écoute, cela se traduira par l’entendre des bruits des decticelles en fond de l’audio. Les coefficients ne converge plus du tout et suivent les variations des grandeurs statistiques.

3) On teste maintenant notre algorithme RLS à l’aide de la fonction algo\_RLS développée plus tôt. On passe en paramètre, le signal x et le signal d\_test. De plus, on a P=3, lambda et delta=0.1.

On modifie la valeur lambda en paramètre. Lambda prend en valeur: 0.8;0.9; 0.95 et 1.

Par la suite, on affiche dans divers figure le signal d’erreur e(n) et l’évolution des coefficient du filtre w(n) au cours du temps pour chaque valeur de lambda.

Pour finir on écoute ensuite notre signal e(n) pour chaque valeur de lambda.

On commence dans un premier temps avec lambda qui vaut 1 et delta=0.1:

**Pour λ = 1 et δ =0.1 :**

| signal d’erreur e(n). | évolution des coefficients du filtre W. |
| --- | --- |
|  |  |
| évolution de la norme 2 au carré de  l’erreur sur le filtre. | évolution de l’erreur sur les coefficients du  filtre |W-Wopt| |
|  |  |

**Pour λ = 0.95 et δ =0.1 :**

| signal d’erreur e(n). | évolution des coefficients du filtre W. |
| --- | --- |
|  |  |
| évolution de la norme 2 au carré de  l’erreur sur le filtre. | évolution de l’erreur sur les coefficients du  filtre |W-Wopt| |
|  |  |

**Pour λ = 0.9 et δ =0.1 :**

| signal d’erreur e(n). | évolution des coefficients du filtre W. |
| --- | --- |
|  |  |
| évolution de la norme 2 au carré de  l’erreur sur le filtre. | évolution de l’erreur sur les coefficients du  filtre |W-Wopt| |
|  |  |

**Pour λ = 0.8 et δ =0.1 :**

| signal d’erreur e(n). | évolution des coefficients du filtre W. |
| --- | --- |
|  |  |
| évolution de la norme 2 au carré de  l’erreur sur le filtre. | évolution de l’erreur sur les coefficients du  filtre |W-Wopt| |
|  |  |

**Observations:**

On constate que pour 𝛌=1, nos coefficients du filtre W convergent. On observe que lorsqu'on modifie les valeurs de lambda, la convergence est modifiée. En concordance avec les exemples lorsque l’on avait un signal stationnaire les coefficients oscillent lorsque les interlocuteurs parlent ce qui est normal car on a une grande modification des grandeurs statistique lorsqu’il parle.

4) On finit par tester l’algorithme de filtrage optimal au sens des moindres carrés.Pour cela, on reprend notre fonction précédemment créée. Cependant, on modifie rdx avec notre signal d\_test. On recalcule donc notre notre filtre w\_LS. Avec ce nouveau filtre des moindres carrés, on va filtrer à l'aide de la commande filter notre signal x afin d'obtenir y\_LS. On décide de calculer ensuite l’erreur e(n) selon la formule suivante:



On trace ensuite à l'aide de la commande plot e(n). Pour finir, on écoute notre signal e.

|  |
| --- |

**Observation:**

A l’aide du signal e(n), on se rend compte que nous entendons encore le bruit de l'insecte. On se rend compte que aussi que le signal a bien été filtré car on peut entendre la voix de l’homme contrairement au départ. Cependant, on peut conclure que cet algorithme n’est pas adapté au environnement non stationnaire.

**Conclusion:**

En conclusion, on constate que même si on utilise différents types d’algorithmes, dans le cadre des signaux non stationnaire le filtre RLS permet de filtrer quand même le bruit des decticelles. Au final de manière générale, le filtre RLS avec un lambda 𝛌 égale à 1, quelque soit le talon 𝜹, le filtre RLS reste le meilleur en terme de filtrage, de temps de convergence, ainsi que de qualité de son. Il permet de réduire le rapport signal à bruit au maximum.

VII- Bibliographie.

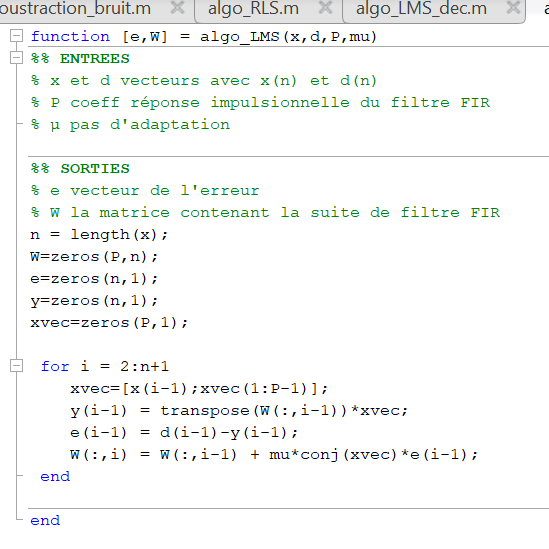
[Maximum elements of an array - MATLAB max - MathWorks France](https://fr.mathworks.com/help/matlab/ref/max.html#:~:text=M%20%3D%20max%20%28A%2C%20%5B%5D%2C%27all%27%29%20finds%20the%20maximum,over%20the%20dimensions%20specified%20in%20the%20vector%20vecdim.)

<https://en.wikipedia.org/wiki/Least_mean_squares_filter>

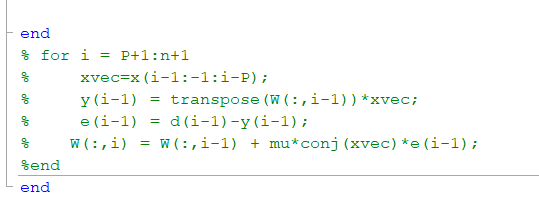
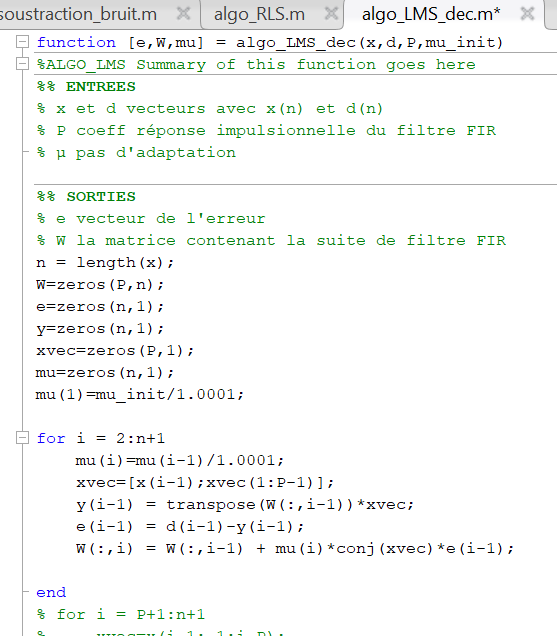
Cours\_FiltrageAdaptatif\_2122

VIII- Annexe.

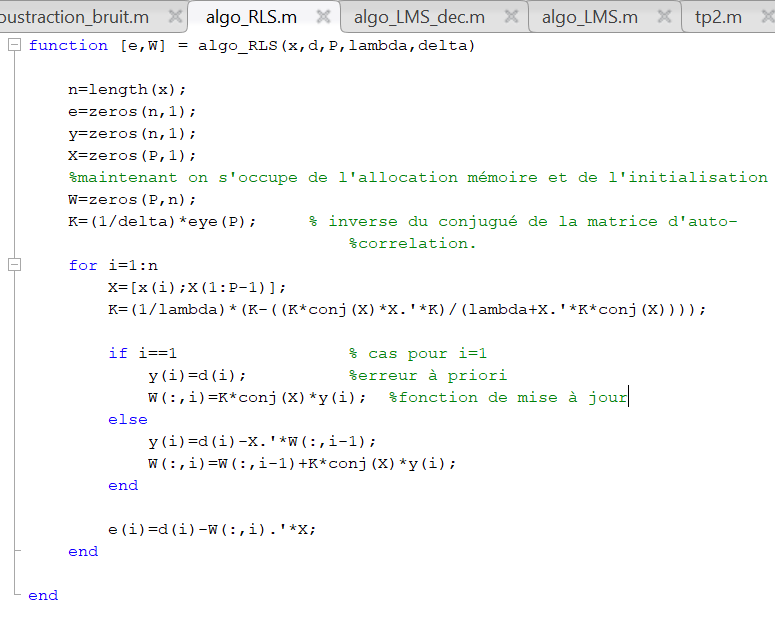
Fonction LMS



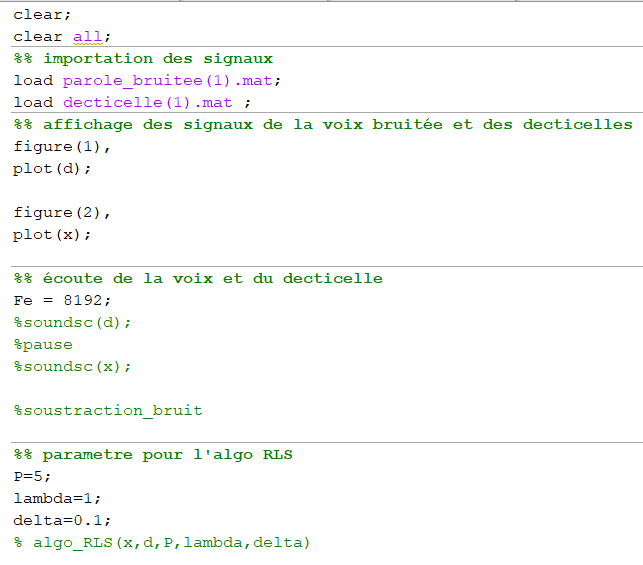
Fonction LMS\_dec



Fonction algo\_RLS



TP2.m



soustraction.m

