BACHELARD Maéva
GALLIEN Anthony

TP3 de Méthodes de Signal Avancées
Soustraction adaptive de bruit

I. Partie I

1. Préparation

Equations Algorithme RLS :

- initialisation de $w^{(0)}$ et $s^{(0)}$
- à chaque instant n :

$$k_n = \frac{1}{\lambda} * \frac{1}{1+\frac{1}{\lambda} * x_n^{\dagger} * s^{(n-1)} * x_n} s^{(n-1)} * x_n$$

$$e_n^{(n)} = w^{(n-1)\dagger} * x_n - d_c(n)$$

$$w^{(n)} = w^{(n-1)} + e_n^{(n)} * k_n$$

$$s^{(n)} = \frac{1}{\lambda} * s^{(n-1)} - \frac{1}{\lambda} * k_n * x_n^{\dagger} * s^{(n-1)}$$

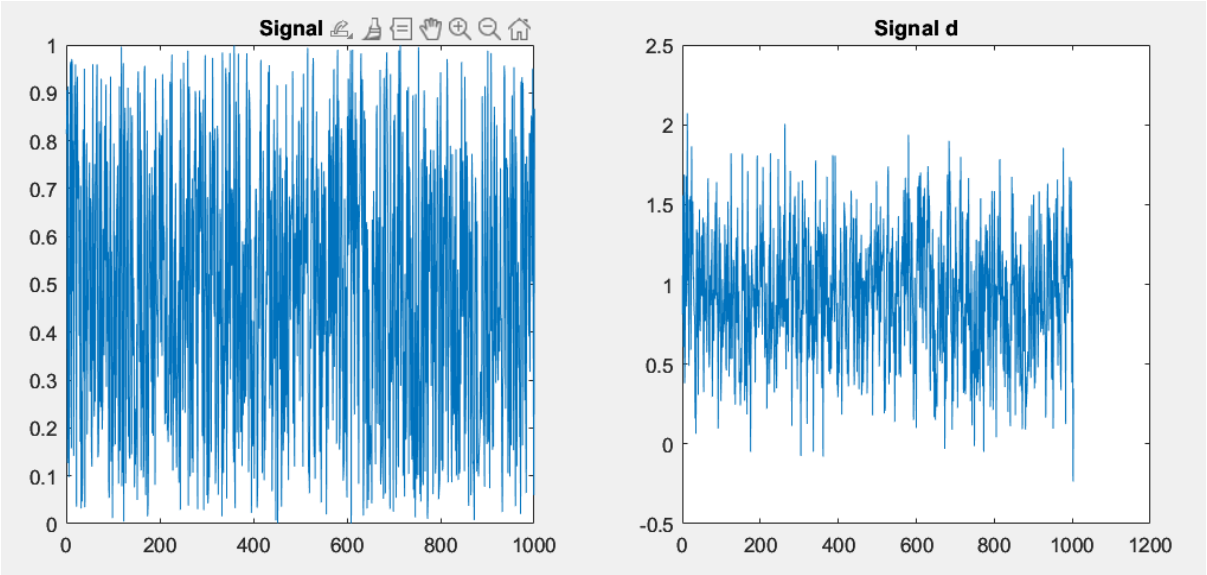
2. Mise en oeuvre de l'algorithme sur matlab

On réalise la mise en œuvre de l'algo pas à pas. Tout d'abord on initialise tous les éléments puis on met en œuvre les 4 équations ci-dessus.

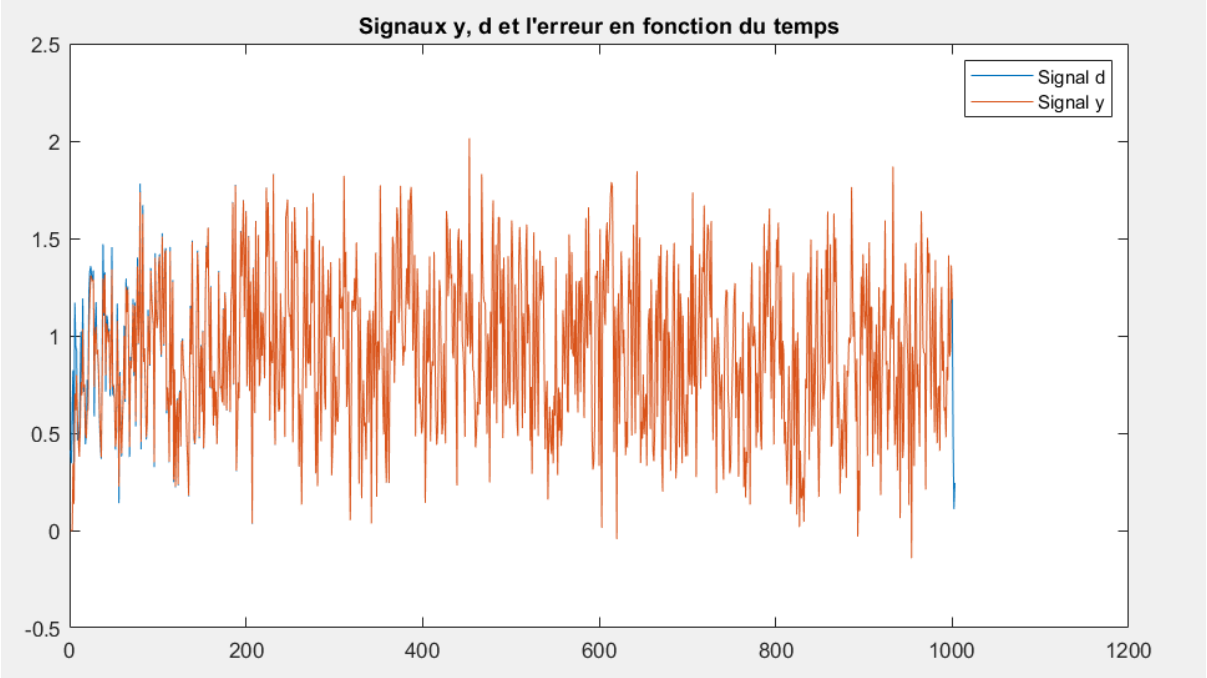
```
function [w, y, e] = algoms_RLS(x, d, order, alpha, lambda)

    % Initialisation
    N = length(x);
    y = zeros(N, 1);
    w = zeros(order, 1);
    S = alpha * eye(order);
    e = zeros(N, 1);

    % Hérédité
    for n = order:N

        k = (1/lambda) * 1 / (1 + (1/lambda) * x(n-order+1:n)' * S * x(n-order+1:n)) * S * x(n-order+1:n);
        e(n) = w' * x(n-order+1:n) - d(n);
        w = w - e(n) * k;
        S = (1/lambda) * S - (1/lambda) * k * x(n-order+1:n)' * S;
        y(n) = x(n-order+1:n)' * w;

    end
end
```

3. Validation de l'algorithme

Pour valider l'algorithme, nous créons un signal bruit blanc auquel on applique le filtre h = [1 0.8 −0.3 0.4]t.
Le signal en sortie du filtre est le signal d ci-dessous :



Après application de la fonction RLS avec $\alpha = 0.98$ (d'après le cours), nous obtenons les résultats suivants :

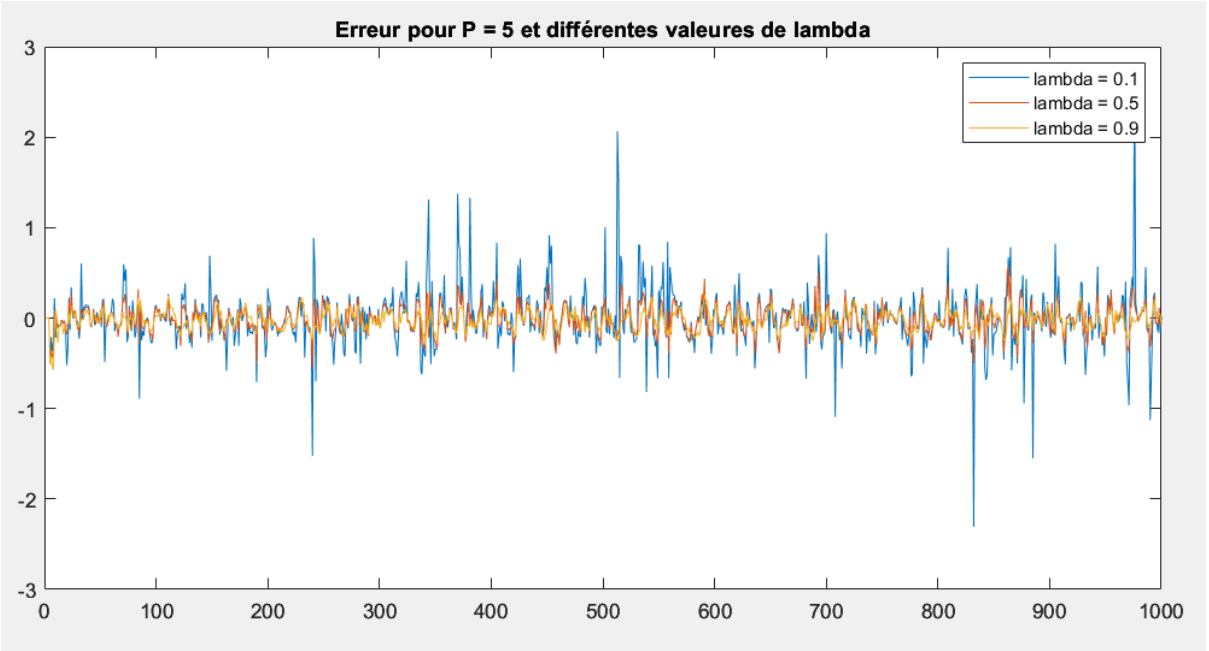


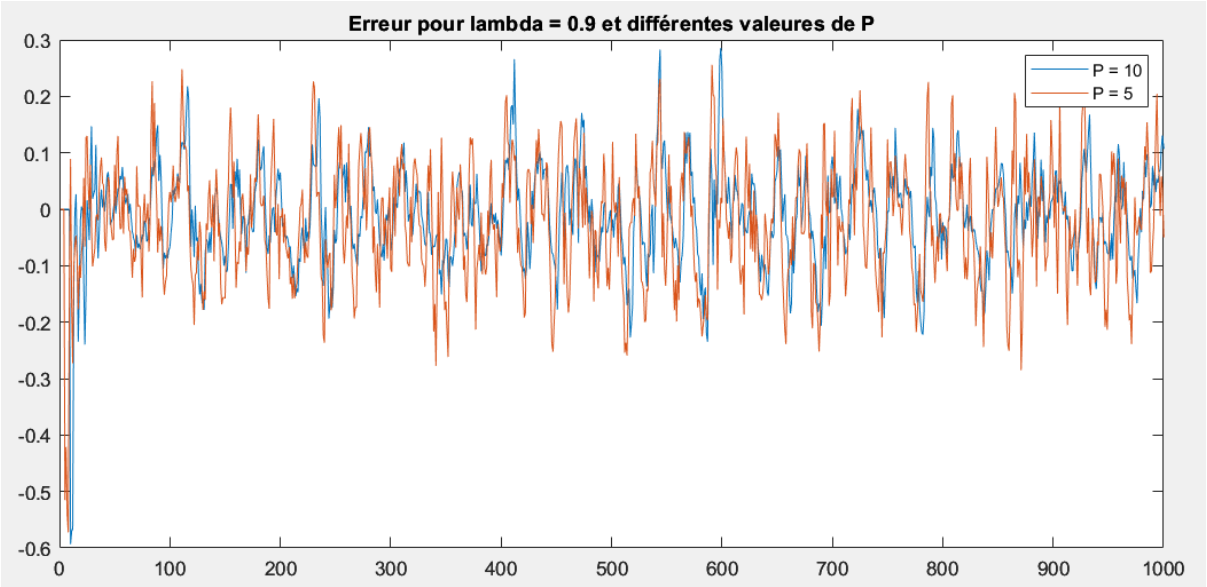

```
w =

    0.4000
   -0.3000
    0.8000
    1.0000
```

Comme on le voit, notre algorithme converge après une centaine d'itérations. On obtient bien les bonnes valeurs de w correspondant à notre filtre. Cette convergence témoigne du bon fonctionnement de notre algorithme.

4. Test de l'algorithme RLS avec un signal simulé

Nous simulons un signal bruit blanc filtré avec un ajout de bruit. Nous allons sur ce signal tester les paramètres lambda et P :

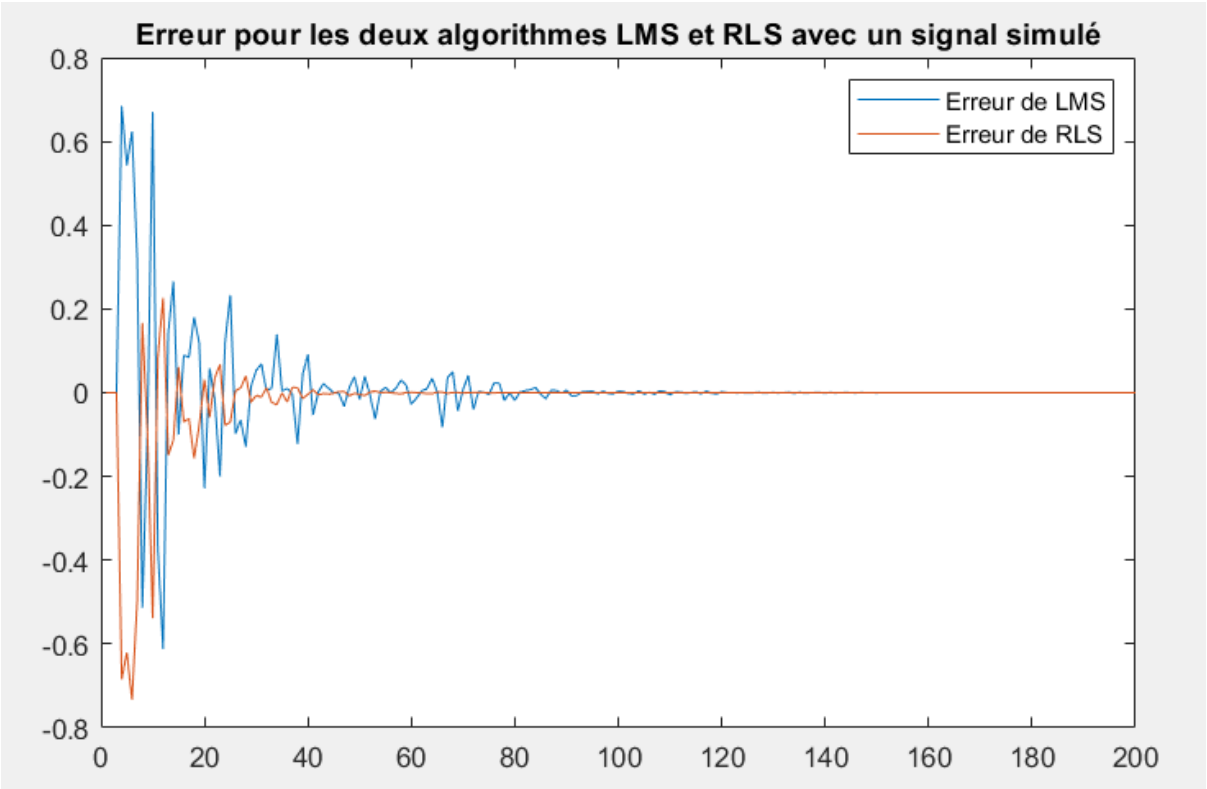




Après avoir appliqué l'algorithme RLS avec des valeurs de lambda différentes, nous voyons que l'erreur dépend grandement de cette valeur. En effet, plus lambda est proche de 1, meilleur est l'erreur. C'est pour cela que dans le cours nous prenions $\lambda \simeq [0.98,\ 0.99]$.

Par ailleurs, la différence en une erreur pour une taille de filtre égale à 5 ou à 10 n'est pas flagrante. L'influence de P est moins importante pour l'algo RLS que l'algo LMS.

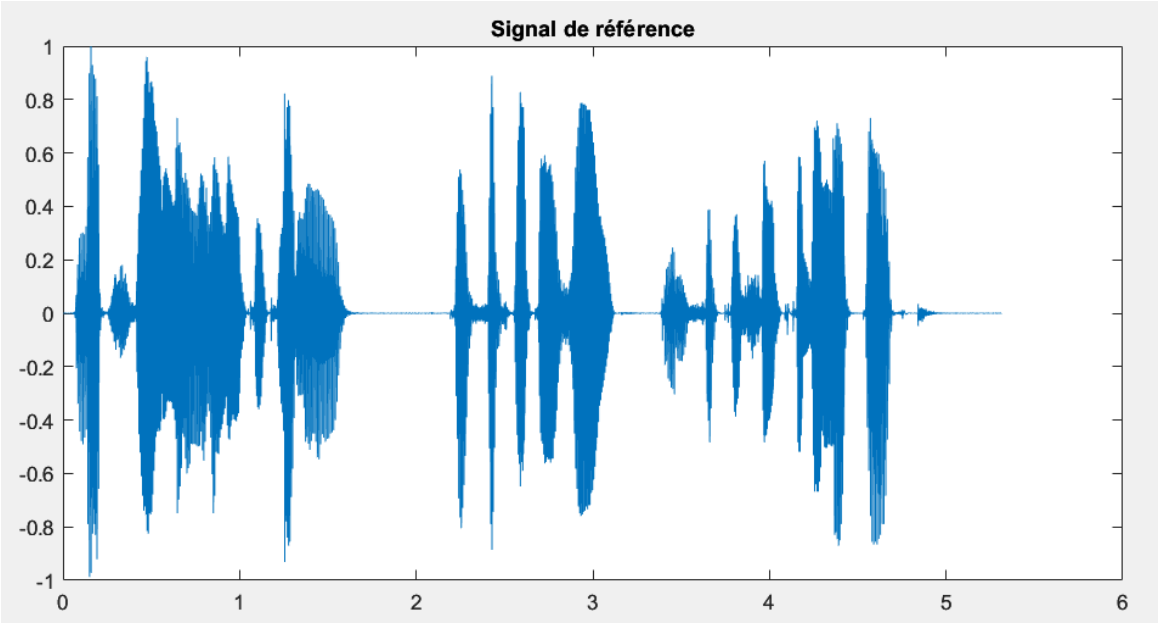
5. Comparaison des performances RLS et LMS



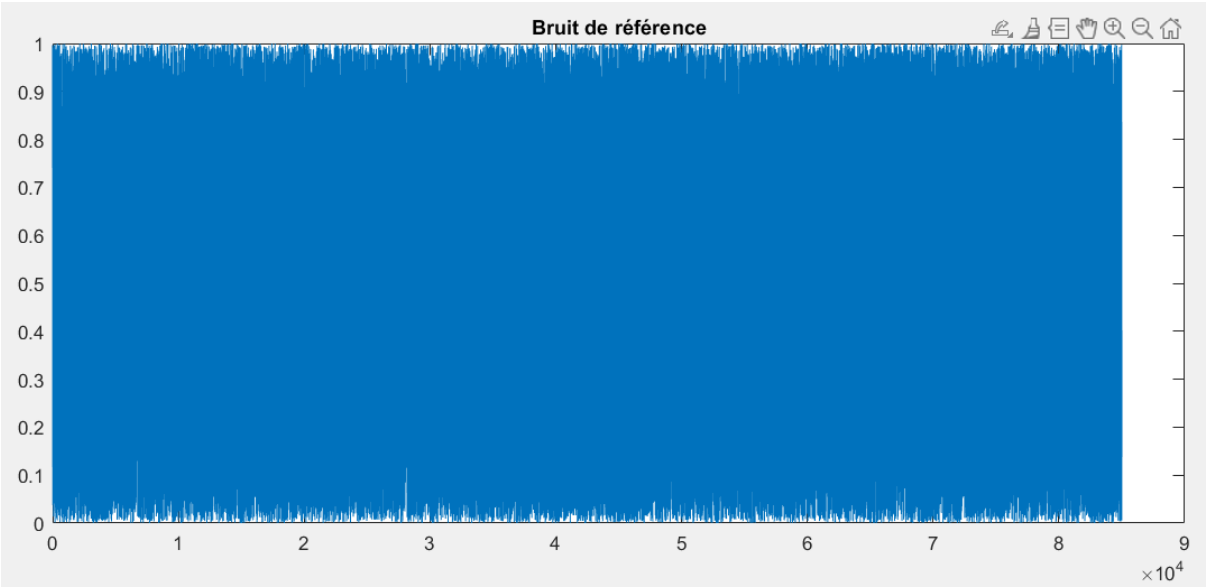
En appliquant les algorithmes RLS et LMS au même signal (celui créée au 4), nous voyons bien que l'erreur converge bien plus rapidement avec l'algo RLS qu'avec l'algo LMS.
Ici nous avons appliqué les algorithmes avec leurs paramètres optimaux.

II. Partie II

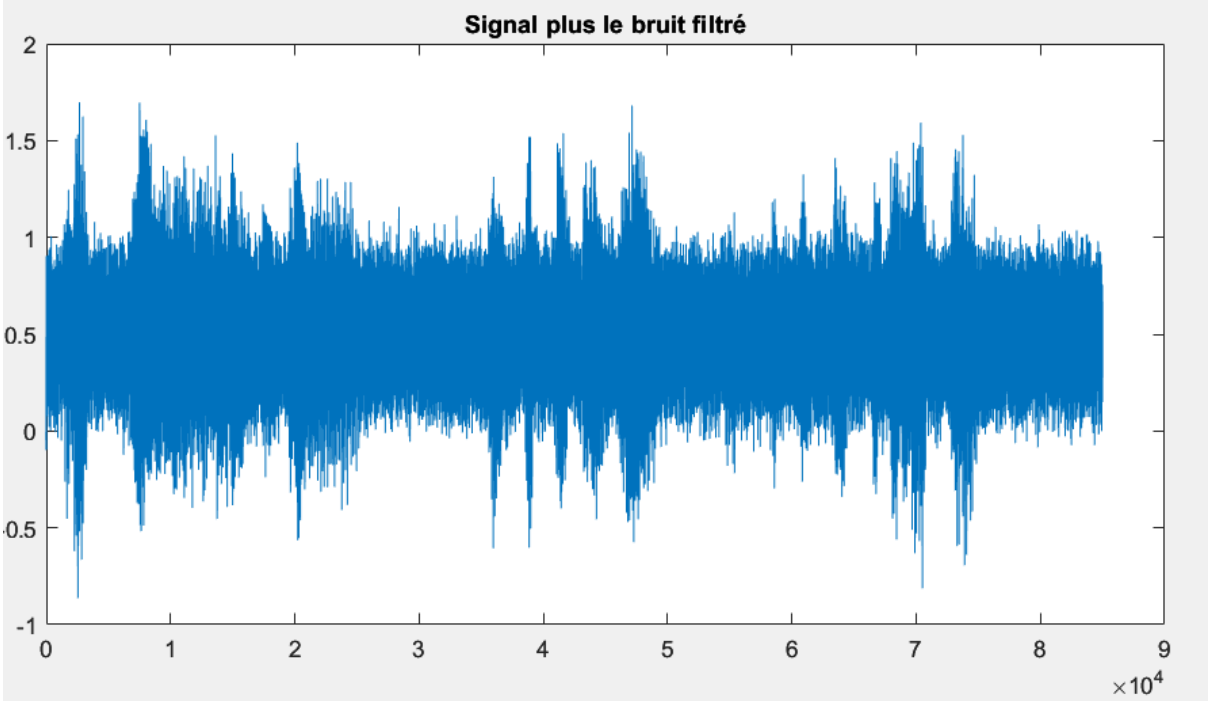
Voici le signal que l'on a chargé :



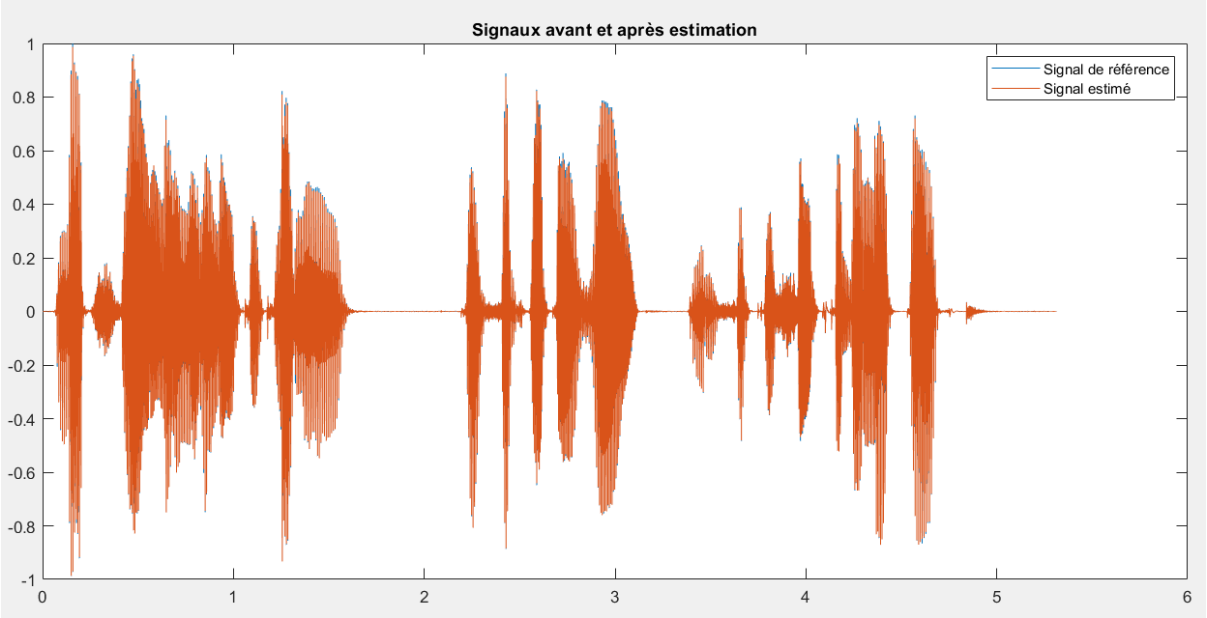
Voici notre bruit de référence N1 :



Voici notre signal auquel on a ajouté le bruit filtré :



Voici notre estimation du signal après application de l'algorithme RLS :



Après visualisation et écoute du signal de sortie on voit que le résultat est très satisfaisant (presque parfait).

III. Partie III

1. Equation de l'algorithme NLMS

- initialisation de $w^{(0)}$
- à chaque instant n :

$$e_c^{(n)}(n) = w^{(n)\dagger *} x_n - d_c(n)$$

$$w^{(n+1)} = w^{(n)} + \frac{\mu}{||x_n||^2} * e_c^{(n)}(n) * x_n$$

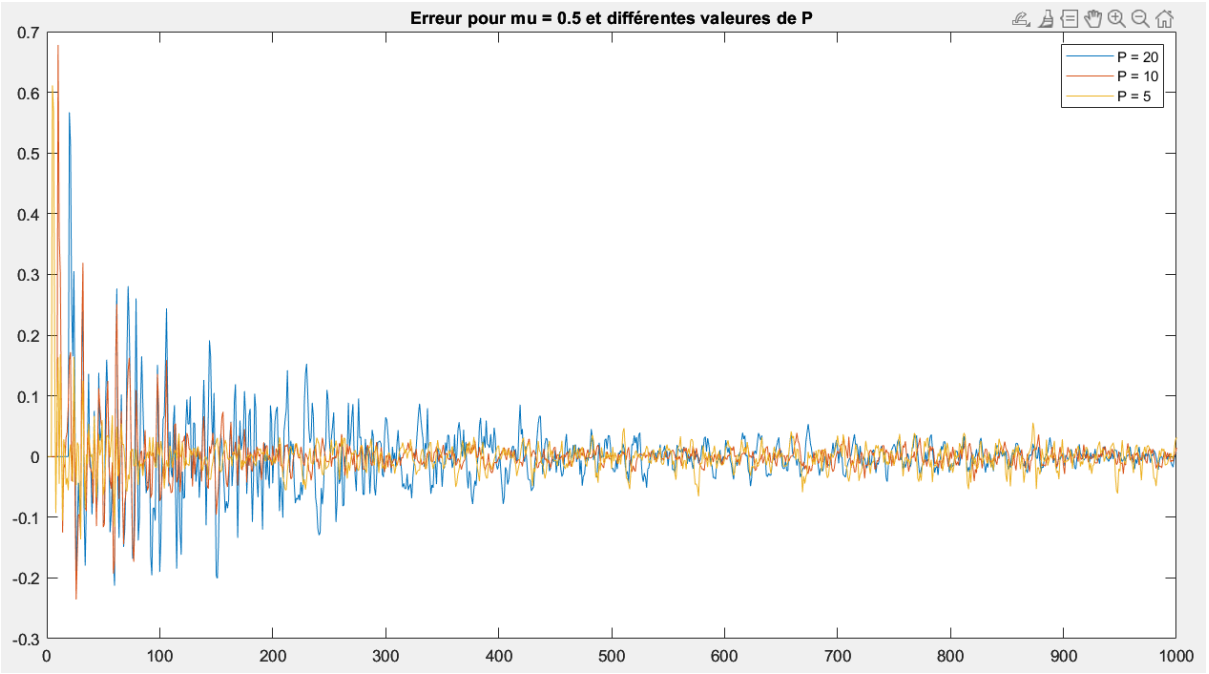
2. Implémentation de l'algorithme NLMS sur matlab

Nous reprenons l'algorithme LMS que nous avions créé à la séance de TP précédente en le normalisant. Ceci revient à diviser μ par la norme de x :

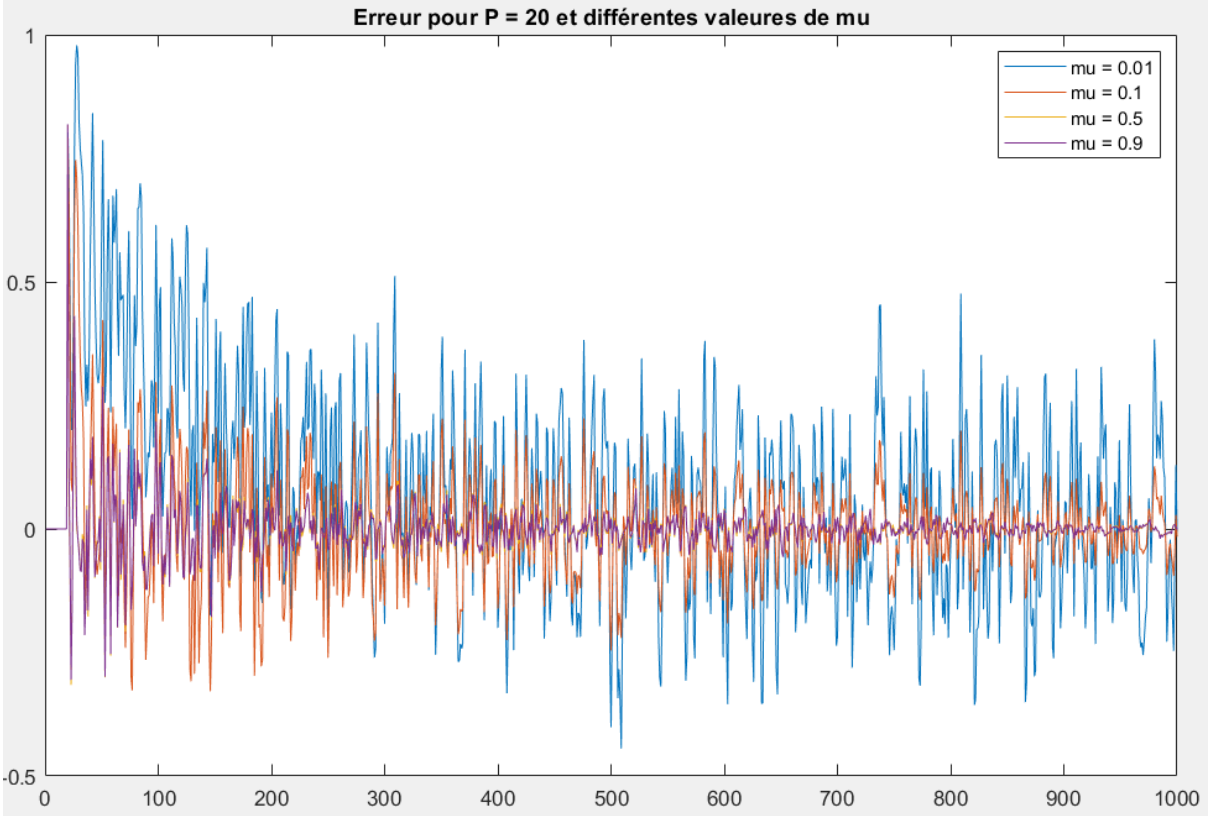
```
function [w, y, e] = algoms_nLMS(x, d, P, mu)

    % Initialisation
    N = length(x);
    w = zeros(P, 1);
    e = zeros(N, 1);
    y = zeros(N, 1);

    % Hérédité
    for n = P:N
        x_actual = x(n:-1:n-P+1);
        y(n) = w' * x_actual;
        e(n) = d(n) - y(n);
        w = w + (mu/norm(x_actual)^2) * e(n) * x_actual;
    end
end
```

3. Test de l'algorithme NLMS sur un signal simulé

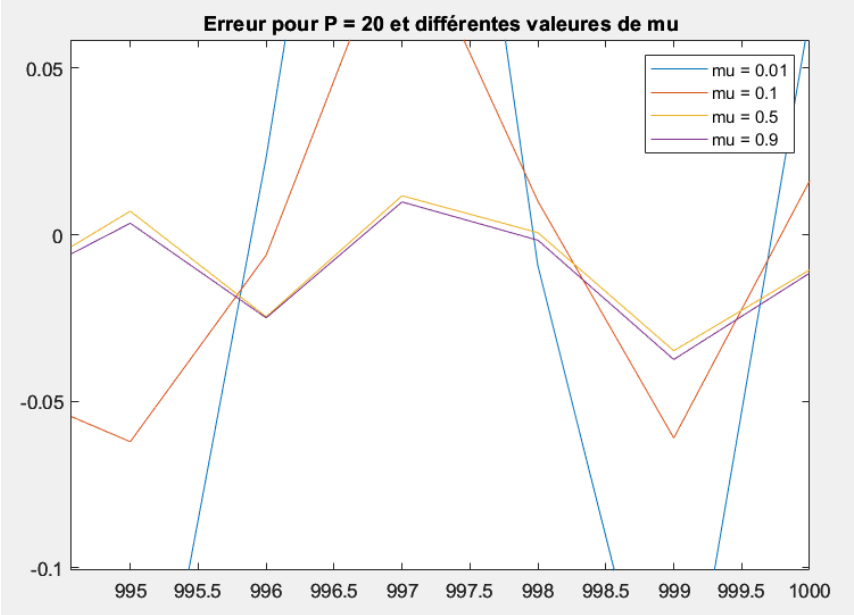
On reprend maintenant le signal créé à la partie I.4 afin de tester notre algorithme:



Nous voyons sur ce graphique que diminuer P permet d'avoir une erreur plus faible. Tout comme pour le LMS.

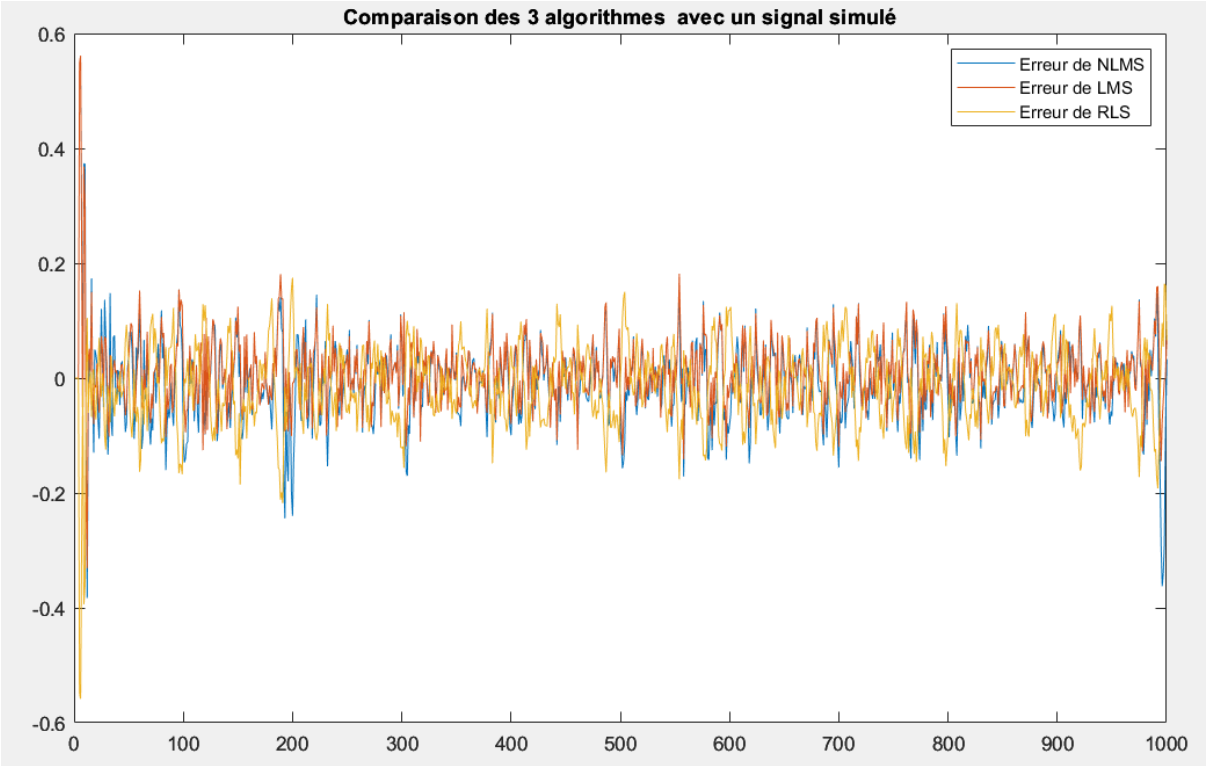


De plus, il semble que pour mu = 0.5 ou mu = 0.9 l'erreur converge.



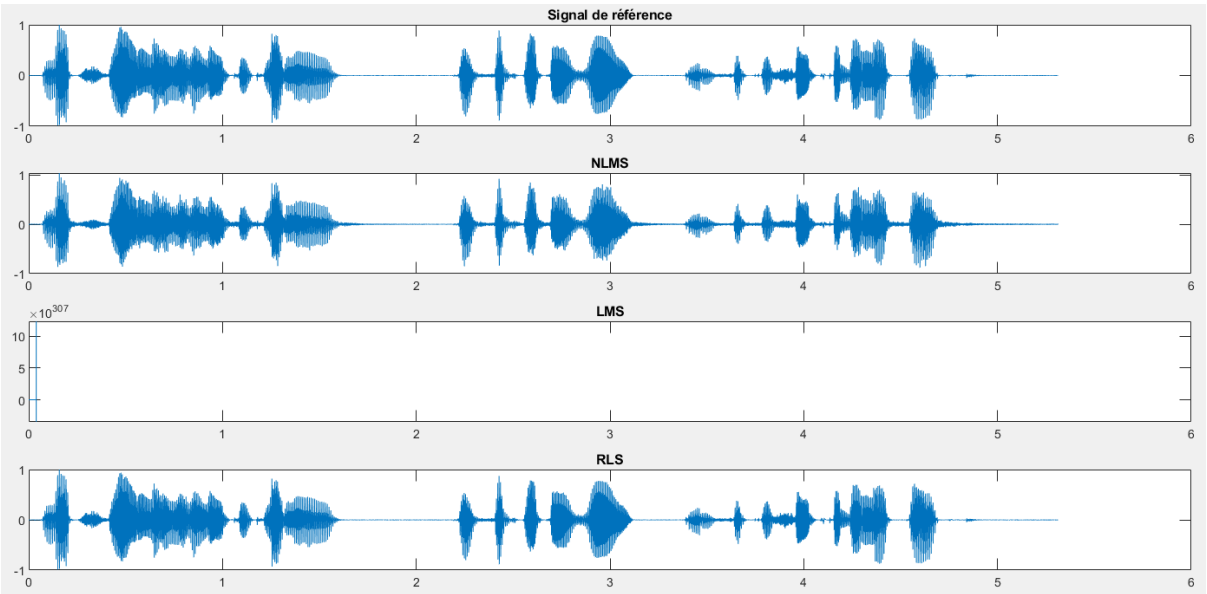
En zoomant sur le graphique précédent, nous voyons que pour mu=0.5 ou mu=0.9 l'erreur est très similaire. Une valeur de mu trop faible empêche donc le bon apprentissage.

4. Comparaison des algorithmes LMS, RLS et NLMS



On a calculé l'erreur en partant du signal simulé et en lui appliquant les trois algorithmes (avec leurs paramètres optimaux).
On voit que sur ce signal les trois algorithmes sont très performants.

Nous comparons maintenant ces trois algorithmes sur un signal de parole bruité :



En appliquant le signal de voix, on voit que l'algo LMS diverge très rapidement et est donc inutilisable. Le NLMS permet de normaliser le LMS, ainsi le signal ne diverge pas. Cependant après écoute, on entend qu'il y a toujours un bruit résiduel avec le NLMS que l'on ne retrouve pas avec le RLS.
Pour ce test, le RLS est le meilleur algorithme de traitement de parole.

En conclusion, l'efficacité des trois algos dépendent grandement de leurs paramètres d'entrée et des signaux à analyser. En appliquant les "meilleurs" paramètres, nous trouvons cependant que l'algorithme RLS est le meilleur sur le traitement vocal grâce à son pas adaptatif.