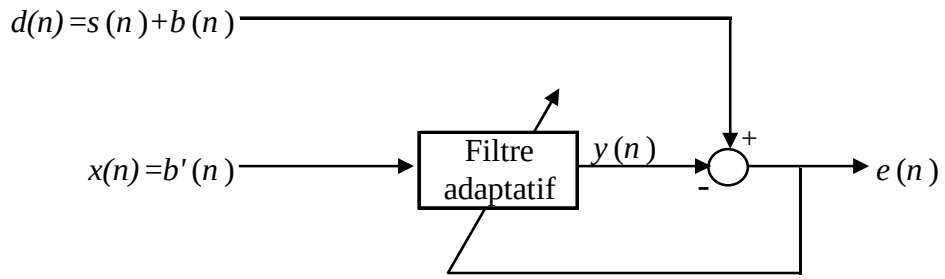
SE-4201A	**TP Matlab de filtrage adaptatif** **Etude des algorithmes LMS et RLS appliqués à un problème de soustraction de bruit**
	Durée : 5 heures
2021-2022	Texte de F. Nadal, d'après des TP conçus par J.-F. Bercher et P. Jardin

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.

Pour rappel, il est toujours utile et souvent indispensable de lire l'aide en ligne sur les fonctions Matlab. Pour obtenir de l'aide sur une commande, il suffit de taper `help` suivi du nom de la commande.

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`). Il s'agit de mettre en application vos connaissances en filtrage adaptatif pour restituer un signal de parole propre à partir des deux signaux disponibles. 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 $s(n)$.

Dans la mesure où les signaux $x(n)$ et $d(n)$ ne sont pas stationnaires, les grandeurs $r_{xx}(k) = E\left[x(n)x(n-k)^* \right]$ et $r_{dx}(k) = E\left[d(n)x(n-k)^* \right]$ n'ont de sens qu'à très court terme (sur des intervalles de quasi-stationnarité). Le filtre devra donc être identifié de manière adaptative (à l'aide de l'algorithme LMS ou de l'algorithme RLS).

I. Acquisition des signaux

1) Chargez les signaux à l'aide de la commande `load` :
```
load parole_bruitee;
load decticelle ;
```
Les signaux sont alors chargés dans l'environnement sous les noms `d` et `x`.

2) Tracez ces signaux (commande `plot`) et écoutez-les (commande `soundsc`).
N.B. : les signaux sont échantillonnés à $F_e = 8192$ Hz, mais dans l'ensemble du TP, on pourra se contenter de faire les tracés temporels directement en fonction de l'indice Matlab.

II. Algorithme LMS

1) Écrivez une fonction Matlab de syntaxe d'appel `[e,w]=algo_LMS(x,d,P,mu)` implantant l'algorithme LMS. Cette fonction prend en entrée :
   - les vecteurs `x` et `d` contenant tous les échantillons des signaux $x(n)$ et $d(n)$,
   - le nombre de coefficients P de la réponse impulsionnelle (RI) du filtre FIR recalculé à chaque instant,
   - le pas d'adaptation μ (`mu`).

   Cette fonction devra fournir en sortie :
   - le vecteur `e` contenant tous les échantillons de l'erreur $e(n)$,
   - la matrice `w` contenant la suite des filtres obtenus (chaque colonne de cette matrice devra contenir les coefficients de la RI du filtre calculée à un instant n donné (vecteur noté $\underline{w}(n)$ en cours)). La matrice `w` comportera donc P lignes.

2) Ecrivez un programme principal `soustraction_bruit.m` pour tester votre fonction `algo_LMS` avec le pas $\mu = 10^{-10}$ et une longueur de filtre P égale à 3 en visualisant :
   - le signal d'erreur $e(n)$,
   - l'évolution des coefficients du filtre $\underline{w}(n)$ au cours du temps,
   - l'évolution de l'erreur sur les coefficients du filtre $\left|\underline{w}(n) - \underline{w}_{opt}\right|$, sachant que le filtre optimal est donné par : $\underline{w}_{opt} = \left[1,\ 1/2,\ 1/4\right]^T$,
   - l'évolution de la norme 2 au carré de l'erreur sur le filtre.

   Commentez ces résultats. Ecoutez aussi le signal d'erreur $e(n)$. Qu'en pensez-vous ?

   N.B. : pour les deuxième et troisième tracés, on pourra utiliser le fait que, si `M` est une matrice, alors `plot(M)` trace toutes les colonnes de `M`. Pour tracer les lignes de `M`, il suffit donc de faire : `plot(M.')`.

3) Etudiez l'influence du pas d'adaptation sur la convergence de l'algorithme pour un nombre P de coefficients fixé ($P = 3$) :
   - On commencera par utiliser un pas d'adaptation de la forme : $\mu = \dfrac{2}{\alpha P\, r_{xx}(0)}$. Le problème est que $r_{xx}(0)$ n'existe pas car $x(n)$ n'est pas stationnaire et que $\left|x(n)\right|^2$ est d'amplitude très variable selon la valeur de n. Il convient donc de choisir le pas d'adaptation correspondant au cas le plus contraignant, c'est-à-dire : $\mu = \dfrac{2}{\alpha P \max\left(\left|x(n)\right|^2\right)}$. Déterminez la valeur minimale de α pour laquelle l'algorithme converge. Testez ensuite l'algorithme pour différentes valeurs de α (par exemple 0.1, 0.5, 1, 2, 3, 30). Commentez les résultats. Quelle est d'après vous la valeur de α permettant d'obtenir le meilleur compromis entre vitesse de convergence et erreur résiduelle ?
   - Testez l'utilisation d'un pas variable (décroissant) en modifiant votre fonction `algo_LMS` en `algo_LMS_dec`. La fonction `algo_LMS_dec` aura pour syntaxe d'appel : `[e,w,mu]=algo_LMS_dec(x,d,P,mu_init)`. Le paramètre d'entrée `mu_init` correspondra au pas initial, et le vecteur de sortie `mu` contiendra la suite des pas d'adaptation. Le pas d'adaptation μ_n sera tel que $\mu_n = \dfrac{\mu_{n-1}}{1.0001}$. Complétez votre programme principal pour tester cette fonction en prenant un pas initial égal à 10^{-9}. Visualisez les mêmes courbes que précédemment, ainsi que l'évolution des pas μ_n. Ecoutez aussi le signal d'erreur $e(n)$. Interprétez les résultats.

4) Complétez le programme principal pour tester vos fonctions `algo_LMS` et `algo_LMS_dec` avec un nombre de coefficients P différent de l'optimum (en prenant $P = 2$ puis $P = 5$ par exemple). Visualisez le signal d'erreur $e(n)$ et l'évolution des coefficients du filtre $\underline{w}(n)$ au cours du temps. Ecoutez aussi le signal d'erreur $e(n)$. Interprétez les résultats.

III. Algorithme RLS

1) Écrivez une fonction Matlab de syntaxe d'appel `[e,w]=algo_RLS(x,d,P,lambda,delta)` implantant l'algorithme RLS. Cette fonction prend en entrée :
   - les vecteurs `x` et `d` contenant tous les échantillons des signaux $x(n)$ et $d(n)$,
   - le nombre de coefficients P de la réponse impulsionnelle du filtre FIR recalculée à chaque instant,
   - le facteur d'oubli λ (`lambda`),
   - le talon δ (`delta`) servant à initialiser l'algorithme (cf. cours).

   Cette fonction devra fournir en sortie :
   - le vecteur `e` contenant tous les échantillons de l'erreur $e(n)$ (erreur a posteriori),
   - la matrice `w` contenant la suite des filtres obtenus (chaque colonne de cette matrice devra contenir les coefficients de la RI du filtre calculée à un instant n donné (vecteur noté $\underline{w}(n)$ en cours)). La matrice `w` comportera donc P lignes.

2) Complétez le programme principal `soustraction_bruit.m` pour tester la fonction `algo_RLS`. On travaillera avec $P=3$.
   - Fixez d'abord la valeur du talon δ à 0.01, et étudiez l'influence du facteur d'oubli λ en testant l'algorithme par exemple avec $\lambda=0.9$, $\lambda=0.95$ et $\lambda=1$. Commentez les résultats obtenus.
   - Fixez ensuite la valeur du facteur d'oubli λ à celle ayant fourni les meilleurs résultats à la question précédente, et étudiez l'influence du talon δ en testant l'algorithme par exemple avec $\delta=0.1$, $\delta=0.01$ et $\delta=0.001$.
   - Comparez les résultats à ceux obtenus précédemment avec l'algorithme LMS.

IV. Filtrage optimal au sens des moindres carrés

En faisant l'hypothèse que les signaux sont stationnaires et ergodiques, on peut estimer les coefficients $r_{xx}(k)$ et $r_{dx}(k)$ en effectuant une moyenne temporelle à la place de la moyenne statistique (espérance mathématique). Ces estimées temporelles d'auto- et d'inter-corrélations seront obtenues à l'aide de la fonction `xcorr` de Matlab.

1) Complétez le programme principal `soustraction_bruit.m` avec les commandes suivantes :
```
P=3;
rxx=xcorr(x,P-1);
rdx=xcorr(d,x,P-1);
```
   A l'issue de ces commandes, vous obtenez deux vecteurs colonnes `rxx` et `rdx` qui contiennent les estimées des coefficients $r_{xx}(k)$ et $r_{dx}(k)$:
   - le vecteur `rxx` correspond à $\left[\, \hat{r}_{xx}(-(P-1)), \ldots, \hat{r}_{xx}(-1), \hat{r}_{xx}(0), \hat{r}_{xx}(1), \ldots, \hat{r}_{xx}(P-1)\right]^T$,
   - le vecteur `rdx` correspond à $\left[\, \hat{r}_{dx}(-(P-1)), \ldots, \hat{r}_{dx}(-1), \hat{r}_{dx}(0), \hat{r}_{dx}(1), \ldots, \hat{r}_{dx}(P-1)\right]^T$.

   A partir du vecteur `rxx` précédemment obtenu, construisez sous Matlab (à l'aide de la commande `toeplitz`) la matrice `Rxx`, estimée de la matrice d'autocorrélation $\underline{\underline{R}}_{xx} = E\left[\underline{x}(n)\underline{x}(n)^H\right]$ où $\underline{x}(n) = \left[x(n), x(n-1), \ldots, x(n-P+1)\right]^T$. A partir du vecteur `rdx` précédemment obtenu, construisez sous Matlab le vecteur `Rdx`, version estimée du vecteur $\underline{R}_{dx} = E\left[d(n)\underline{x}(n)^*\right]$.

2) Déduisez-en le filtre optimal au sens des moindres carrés $\underline{w}_{LS}$ que l'on nommera sous Matlab `w_LS`. Pour rappel, le vrai filtre optimal est donné par : $\underline{w}_{opt} = \left[1,\ 1/2,\ 1/4\right]^T$. Retrouvez-vous ce filtre ?

3) Filtrez le signal $x(n)$ par le filtre $\underline{w}_{LS}$ (en utilisant la commande `filter`) pour obtenir le signal de sortie $y_{LS}(n)$, puis calculez, tracez et écoutez $e_{LS}(n) = d(n) - y_{LS}(n)$. Comparez le résultat à celui obtenu avec $e_{opt}(n) = d(n) - y_{opt}(n)$ où $y_{opt}(n)$ est le signal obtenu en sortie du filtre $\underline{w}_{opt}$ attaqué par l'entrée $x(n)$.

4) Comparez le filtre $\underline{w}_{LS}$ au dernier filtre obtenu avec l'algorithme RLS (avec $\lambda=1$).

V. Etude dans le cas d'un système non stationnaire

Jusqu'à présent, le système étudié était stationnaire (la réponse impulsionnelle du vrai filtre optimal était égale à $\underline{w}_{opt} = [1,\ 1/2,\ 1/4]^T$). On souhaite maintenant étudier le comportement des algorithmes déjà développés dans le cas où une non-stationnarité lente est introduite dans le système. Pour ce faire, on va maintenant travailler sur un nouveau signal $d(n)$ nommé $d_{test}(n)$, que l'on définira comme suit :

```
N=length(x);
P=3;
w_opt=[1;1/2;1/4];
s=d-filter(w_opt,1,x);
dtest=d;
for n=P:N
    dtest(n)=s(n)+w_opt.'*(1+cos(2*pi*n/N))*x(n:-1:n-P+1);
end
```

Dans toute la suite, on fixera le nombre P de coefficients du filtre à 3 et on remplacera `d` par `dtest` dans les appels aux fonctions déjà développées. Pour chaque test demandé aux questions 2), 3), 4) ci-dessous, on visualisera le signal d'erreur $e(n)$ et l'évolution des coefficients du filtre $\underline{w}(n)$ au cours du temps. On écoutera aussi le signal d'erreur $e(n)$.

1) Tracez le signal $d_{test}(n)$ et écoutez-le. Comparez-le au signal $d(n)$ précédemment considéré.

2) Testez vos fonctions `algo_LMS` et `algo_LMS_dec`. Commentez les résultats. Que pensez-vous de l'utilisation d'un pas décroissant dans ce cas ?

3) Testez votre fonction `algo_RLS` avec $\lambda = 0.8$, $\lambda = 0.9$, $\lambda = 0.95$ et $\lambda = 1$. Expliquez les résultats obtenus.

4) Recalculez le filtre optimal au sens des moindres carrés $\underline{w}_{LS}$ (il suffit de remplacer `d` par `dtest` dans le calcul du vecteur `rdx`, puis de recalculer les vecteurs `Rdx` et `w_LS`). Filtrez le signal $x(n)$ par le filtre $\underline{w}_{LS}$ (en utilisant la commande `filter`) pour obtenir le signal de sortie $y_{LS}(n)$, puis calculez, tracez et écoutez $e_{LS}(n) = d_{test}(n) - y_{LS}(n)$. Commentez les résultats.