Chambre de Commerce et d'Industrie Paris-IdF ———— ESIEE	Unité : Signal et Classification TP filtrage adaptatif	ISBS

Remis par M. J.-F. BERCHER

ÉNONCÉ

Le but de ce TP est d'illustrer et consolider quelques concepts sur les problèmes de filtrage adaptatif. Dans un premier temps, on étudie le problème de manière analytique, puis dans un second temps, on expérimente et étudie le problème par simulation. En particulier, et il faut bien le faire une fois, on programmera et on mettra en oeuvre un algorithme LMS ; on étudiera les propriétés de convergence, le rôle du pas d'adaptation, etc. On considèrera un problème d'identification, puis différents problèmes de soustraction de bruit.

1 Implantation d'un algorithme LMS

Programmez un algorithme LMS. La syntaxe d'appel sera la suivante :

```
    def lms(d,u,w,mu):
    """
    Calcule une itération de l'algorithme du gradient stochastique (LMS)\n
    $w(n+1)=w(n)+\mu u(n)\left(d(n)-w(n)^Tu(n)\right)$$ \n
    Entrées :
        d : séquence désirée à l'instant n \n
        u : vecteur de longueur p des échantillons d'entrée \n
        w : filtre de wiener à mettre à jour \n
        mu : pas d'adaptation
    Sorties :
        w : filtre mis à jour
        erreur : y-yest
        dest : prédiction = $u(n)^T w$
[...]
    return (w,erreur,dest)
    """
```

2 Identification adaptative

Vous testerez cet algorithme sur ***un problème d'identification***. Vous utiliserez comme signal test la sortie d'un filtre excité par un bruit blanc, selon, par exemple

```
N=200
x=randn(N)
htest=10*array([1, 0.7, 0.7, 0.7, 0.3, 0 ])
L=size(htest)
yo=zeros(N)
for t in range(L,200):
    yo[t]=htest.dot(x[t:t-L:-1])
y=yo+ 0.1*randn(N)
# ce qui est équivalent à :
y2=lfilter(htest,[1],x)+0.1*randn(N)
```

Pour mettre en oeuvre la procédure d'identification, il suffit de voir que l'on recherche un filtre, qui excité par $x(n)$ présente une sortie $z(n)$ la plus proche possible de $y_0(n)$. On prend donc ainsi 'u=x', et 'd=y'(la séquence désirée est $y_0(n)$, que l'on doit remplacer par $y(n)$ – y_O n'est pas connu).

1. Débutez par quelques commandes directes (initialisation et une boucle '`for`' sur le temps) pour effectuer cette identification – Si nécessaire, la fonction '`squeeze()`' permet de supprimer les entrées monodimensionnelles du tableau n-D (*e.g.* transforme un tableau (3,1,1) en vecteur de dimension 3)
2. Mettez ensuite en œuvre une procédure d'identification adaptative, en écrivant une fonction `ident` utilisant une boucle sur l'algo `lms` appelé avec les paramètres pertinents
3. Pour évaluer le comportement de l'algorithme, vous pourrez tracer l'erreur d'estimation, les coefficients du filtre identifié en fonction des itérations, ainsi que l'erreur quadratique entre le filtre identifié et le filtre exact. Ceci pourra être effectué pour différents ordres p (l'ordre du filtre n'est pas connu...) et pour différentes valeurs du pas d'adaptation μ. L'erreur quadratique pourra être évaluée simplement par une *liste en intention* selon

   ```
   Errh=[sum(he-w[:,n])**2 for n in range(N+1)]
   ```

4. Examinez la vitesse de convergence en fonction du pas d'adaptation μ, ainsi que les capacités d'adaptation, en introduisant une non-stationnarité lente dans le signal, par exemple selon

   ```
   ### Non stationnarité lente

   N=1000
   u=randn(N)
   y=zeros(N)
   htest=10*array([1, 0.7, 0.7, 0.7, 0.3, 0 ])
   L=size(htest)
   for t in range(L,N):
       y[t]=dot((1+cos(2*pi*t/N))*htest,u[t:t-L:-1])
   y+=0.01*randn(N)
   ```

5. Fort aimablement, on vous fournit une implantation de l'algorithme des mondres carrés récursifs. Reprendre alors les expérimentations précédentes. Comparez et concluez.
   Pour importer les définitions, faire un `from ada_defs import *`, ou copiez collez les définitions.

3 Soustraction de bruit

3.1 Implantation et premiers tests

Le signal que l'on cherche à estimer est s, à partir du mélange $x = s + b$, et de la référence bruit u. Vous cherchez à retrouver le signal s. Vous appliquerez donc une structure de soustraction de bruit, où le filtre sera identifié de manière adaptative à l'aide d'un algorithme LMS. Vous disposez pour vos expérimentations, de signaux tests contenus dans les fichiers `sb1.npz` et `sb2.npz`.

```
# Pour le premier fichier :
f=numpy.load('sb1.npz')
# le contenu est donné par
f.keys()
# On affecte à des variables locales
obs=f['obs']
ref1=f['ref1']
ref2=f['ref2']
```

Le premier fichier contient deux références bruit, l'une stationnaire, l'autre non stationnaire, que vous considérerez successivement. Vous prendrez des valeurs de pas comprises entre 0.01 et 1. Pour le second signal test, contenu dans le fichier `sb2.npz`, on a une non-stationnarité beaucoup plus importante et un rapport signal-à-bruit très défavorable. Il pourra être utile de traiter le problème en « deux passes », afin d'obtenir une première estimée de la réponse impulsionnelle du filtre, qui sera utilisée lors de la seconde « passe ». Vous pourrez éventuellement utiliser un algorithme LMS normalisé.
Vous implanterez une fonction avec la syntaxe d'appel suivante :

```
def sousb(ref,signal,h_ini,mu):
#
# Algorithme de soustraction de bruit :
# Entrées :
#  ref : reference bruit seul
# signal : voie signal (signal composite s +b, ou b est correllé avec b
# h_ini : réponse impulsionnelle initiale
#  mu : pas d'adaptation
# Sortie :
# s : signal identifié par soustraction de bruit
# h : réponse impulsionnelle identifiée
```

et qui revoie le tuple

```
return (s,h)
```

3.2 Soustraction de bruit et signal de parole

On dispose d'un signal de parole perturbé par un cri d'insecte (il s'agit d'une decticelle bariolée). On dispose aussi du signal de l'insecte seul, pris par le deuxième microphone tout près de celui-ci.

1. On charge les données par

   ```
   f=numpy.load('parole_bruitee.npz')
   g=numpy.load('decticelle.npz')
   ```

   puis les variables sont affectées par

   ```
   d=f['d']
   u=g['u']
   ```

2. Effectuer la soustraction de bruit. Vous pourrez ensuite vous amuser à écouter le résultat de vos expérimentations avec la fonction 'sound' qui a été bricolée par votre serviteur.