PAN Jimmy-Antoine
LIEU Valentine

Projet MSA

Sujet du projet :
https://moodle.ensea.fr/pluginfile.php/68766/mod_resource/content/1/sujet_MSA_FR.pdf

S1 Estimation de l'autocorrélation

Objectifs de cette première séance : découverte du sujet du projet, estimation fiable de l'autocorrélation, critère pour déterminer le caractère voisé d'un son.

Au début du projet, nous disposons de deux fichiers matlab : **main.m** et **affichage.m** .

1. Données sur lesquelles travailler

Pour déterminer si un son est voisé ou non, nous utilisons deux estimations : l'estimation de la moyenne et l'estimation de l'autocorrélation.
Un son est **voisé** lorsque ses variations temporelles présentent une pseudo-périodicité. Il est représenté par une impulsion périodique filtrée. Un son est **non voisé** lorsque ses variations temporelles sont peu structurées. Il est représenté par un bruit blanc filtré.

Dans un premier temps, testons nos estimateurs à l'aide de réalisations de processus dont nous connaissons l'autocorrélation.

2. Estimateur biaisé d'autocorrélation

Créons les fichiers **estim_moy.m** et **estim_auto.m** qui contiennent les fonctions prenant un processus en paramètre et permettant d'obtenir respectivement l'estimation de la moyenne et l'estimation de l'autocorrélation.
L'estimation de la moyenne d'un processus X est la moyenne de X :

$$\frac{1}{N}\sum_{k=0}^{N-1} X_k$$

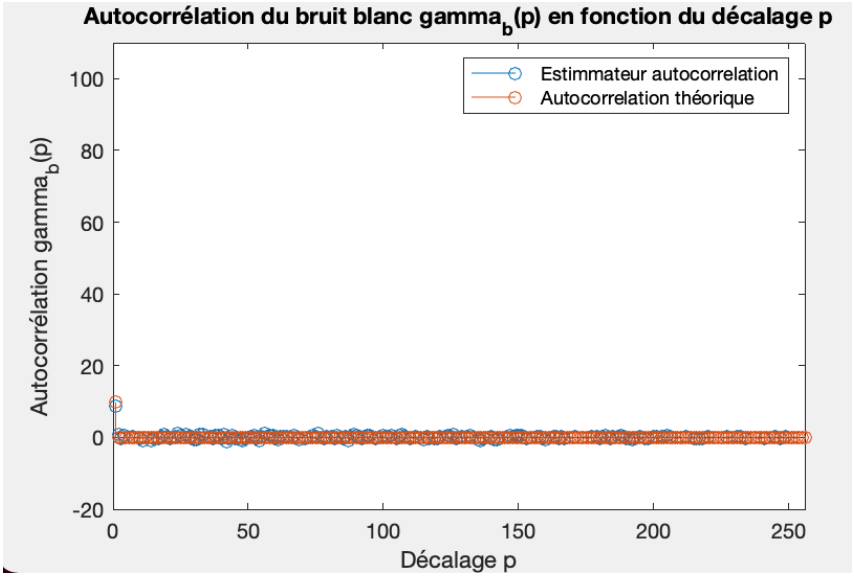
L'estimation de l'autocorrélation d'un processus X est l'autocorrélation de X :

$$\gamma_N(p) = \frac{1}{N}\sum_{k=p}^{N-1} X_k X_{k-p}$$

Bruit blanc

Créons le fichier **blanc.m** qui contient la fonction permettant de générer un bruit blanc gaussien. Cette fonction prend en paramètre une variance (constante pour un bruit blanc).
Remarque : Pour coder blanc.m, nous utilisons la fonction randn(). Remarquons que randn() renvoie une gaussienne centrée réduite. Notre bruit blanc est centré mais pas réduit.
Pour $\sigma_e^2 = 0.5$, l'estimation de la moyenne nous renvoie une valeur très proche de 0. Ceci est cohérent car le bruit blanc suit une loi normale uniforme. Sa moyenne est donc nulle.
L'estimateur de la moyenne est fonctionnelle pour les bruits blancs.
Remarque : Il ne faut pas choisir une variance trop grande, sinon l'estimation de la moyenne est mauvaise. Par exemple pour $\sigma_e^2 = 100$, l'estimation renvoyée est de l'ordre de 0.5 ce qui n'est pas proche de 0 (matlab considère 0 = 10^{-12}).

Voici ce que donne l'estimation de l'autocorrélation biaisée pour une variance de $\sigma_e^2 = 100$:

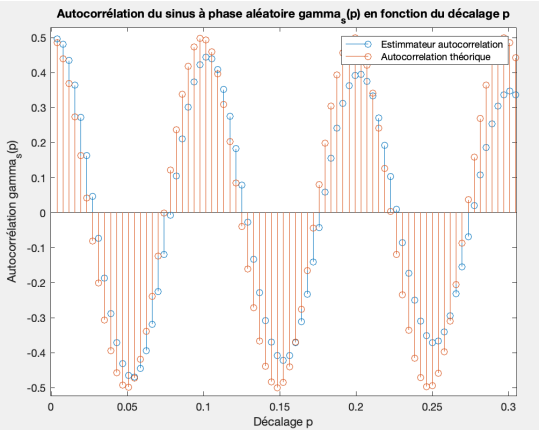


L'estimation de l'autocorrélation du bruit blanc $\gamma_N(p)$ donne 10 lorsque p = 0 et 0 sinon.
Ceci est cohérent car le bruit est non structuré. Il n'y a pas de motifs distinguables, un décalage donne une valeur totalement différente pour chaque échantillon. Il n'y a donc pas de ressemblance entre un bruit blanc et son décalage.
Nous constatons que l'estimation de l'autocorrélation biaisé est fidèle à l'autocorrélation théorique du bruit blanc.

Sinus avec phase uniformément aléatoire sur $[0; 2\pi]$

Créons le fichier **sinus_rd.m** qui contient la fonction permettant de générer un signal sinusoïdal avec une phase uniformément aléatoirement choisie entre 0 et 2π .
Pour obtenir une analyse plus précise sur une période donnée, nous réduisons la période d'étude de $[0; N]$ à $[0; 1]$ en divisant tous les points de l'abscisse par 1/N.

Voici ce que donne l'estimation de l'autocorrélation biaisée :



L'autocorrélation biaisée d'un sinus à phase aléatoire est un sinus de même fréquence et phase que le sinus de l'autocorrélation théorique mais avec une amplitude qui décroît. C'est ce que l'on observe avec notre estim_auto.m pour une fonction de sinus_rd.m .
estim_auto.m est donc cohérent pour les sinus.
La décroissance de l'amplitude s'explique par le biais d'estimation qui se retrouve à l'aide de la formule suivante :

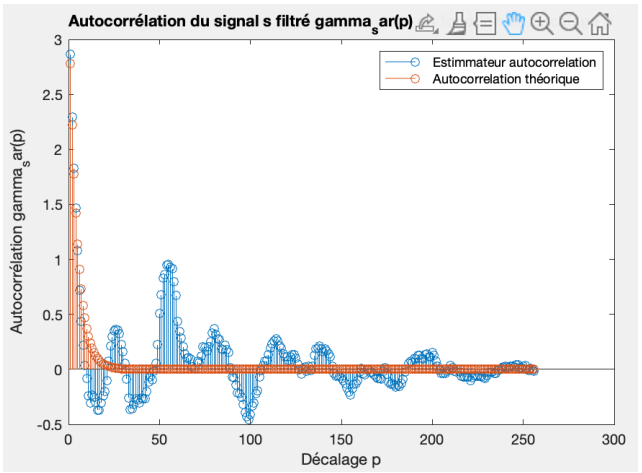
$$E(\gamma_{X_b}(p,w)) \; - \; \gamma_X(p,\,w) \; = \; \frac{-p}{N}\gamma_x(p)$$

Nous souhaitons avoir une estimation biaisée de l'autocorrélation proche de l'autocorrélation réelle. La différence des deux est proportionnelle à -p/N . Ceci explique donc la diminution de l'amplitude de l'autocorrélation biaisée du sinus. Celui-ci est fidèle à l'autocorrélation théorique lorsque p est petit et, lorsque p devient grand et se rapproche de la valeur de N, l'autocorrélation biaisée est éloignée de la valeur de l'autocorrélation théorique.
Nous constatons que l'estimation de l'autocorrélation biaisé n'est (partiellement) pas fidèle à l'autocorrélation théorique du bruit blanc.

AR(1)

Créons le fichier **AR1.m** qui contient la fonction permettant de générer un signal AR(1) de la forme :
X(n, ω) = aX(n − 1, ω) + B(n, ω) avec |a| = 0,8 < 1
Cette fonction prend en paramètre un réel a et une variance (pour générer le bruit blanc associé au signal) $\sigma_e^{\,2}$.

Voici ce que donne l'estimation de l'autocorrélation biaisée :

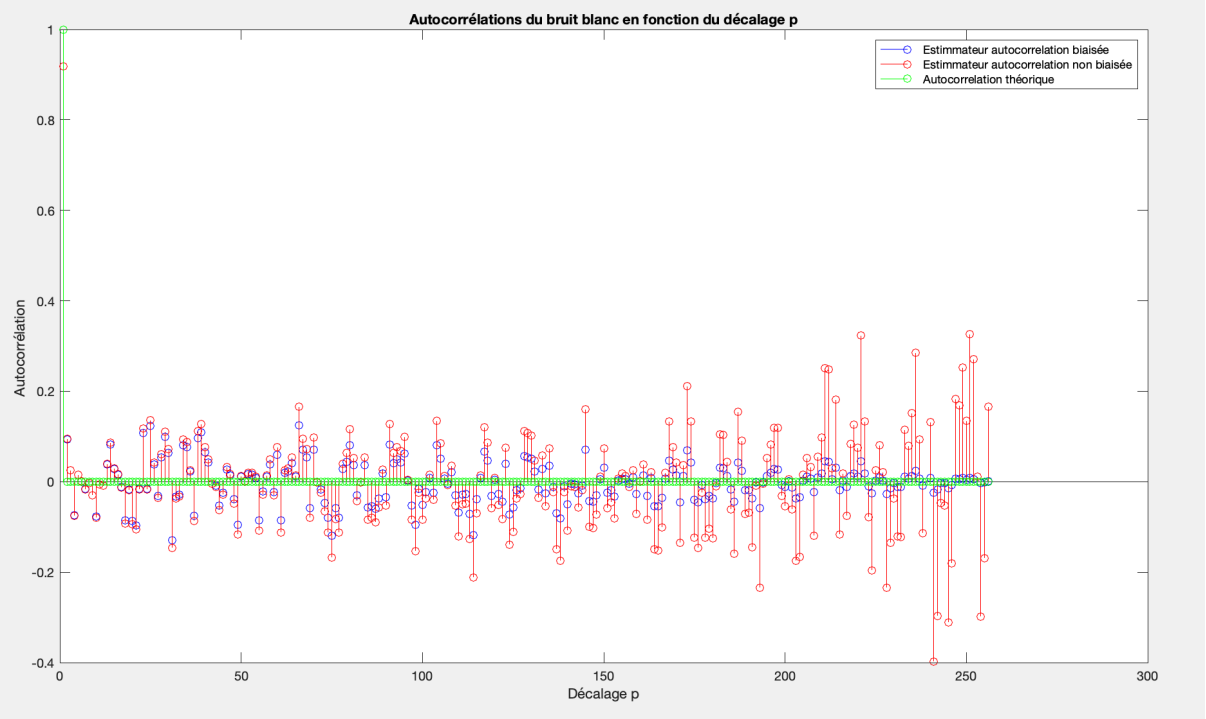


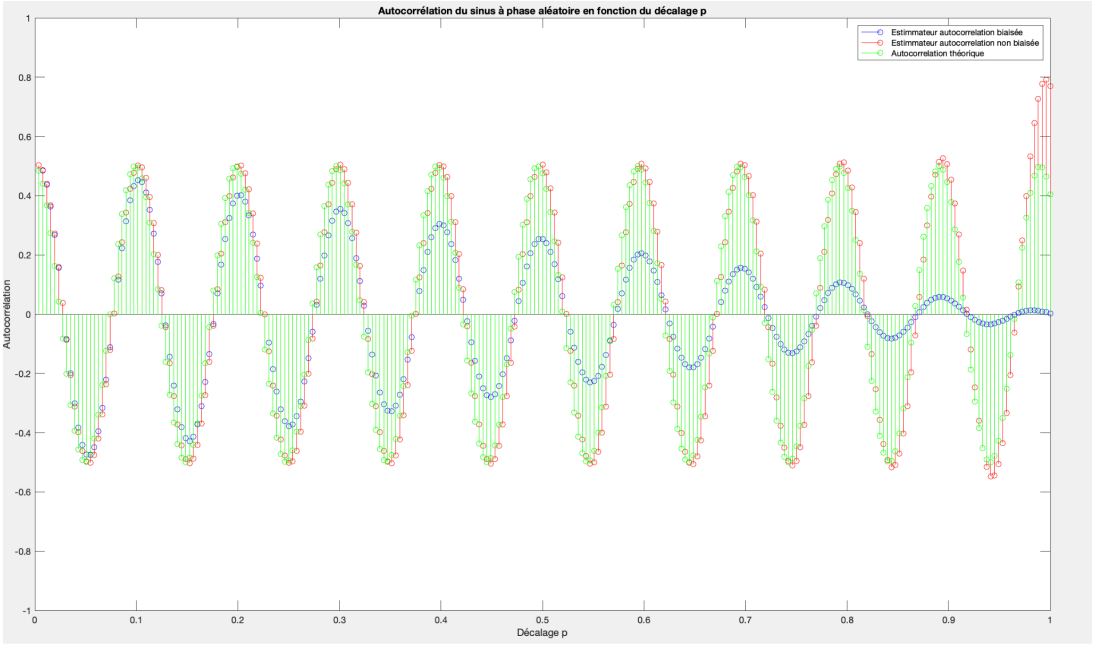
Nous observons que l'autocorrélation biaisée fluctue autour de 0 avec une amplitude qui diminue de plus en plus. Nous observons que l'autocorrélation biaisée est dans un premier temps fidèle à l'autocorrélation théorique. Ensuite, la fidélité n'est plus respectée à cause des fluctuations. Lorsque p augmente, l'amplitude des fluctuations diminue et tend vers 0 : la fidélité avec l'autocorrélation théorique est revenue.
L'influence des termes antécédents du bruit blanc explique les fluctuations de l'estimation biaisée. Celle-ci diminue et tend vers 0 puisque |a|<0.8 . La fidélité est obtenue par la fidélité du bruit blanc. Globalement, nous pouvons constater que l'estimation de l'autocorrélation biaisé est fidèle à l'autocorrélation théorique du signal AR(1).

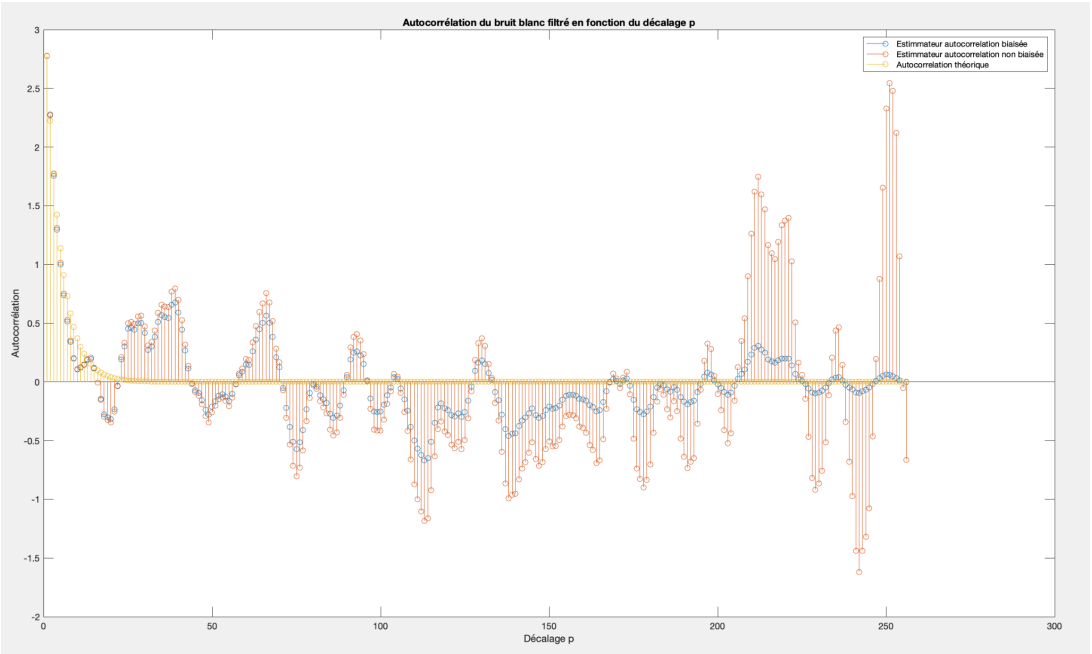
En testant nos estimations sur les courbes théoriques que nous connaissons, nous constatons que l'estimation biaisée est fidèle pour les signaux non structurés modélisables par un bruit blanc. Pour les signaux qui présentent une périodicité, le biais évolue en -p/N : l'estimation biaisée n'est pas fidèle à la valeur théorique recherchée.
Finalement, nous concluons que l'estimation biaisée est adéquate pour les signaux non voisés. Elle n'est pas adaptée aux signaux voisés.

3. Estimateur non biaisé d'autocorrélation

Nous affichons les autocorrélations biaisées et non biaisées des différents signaux







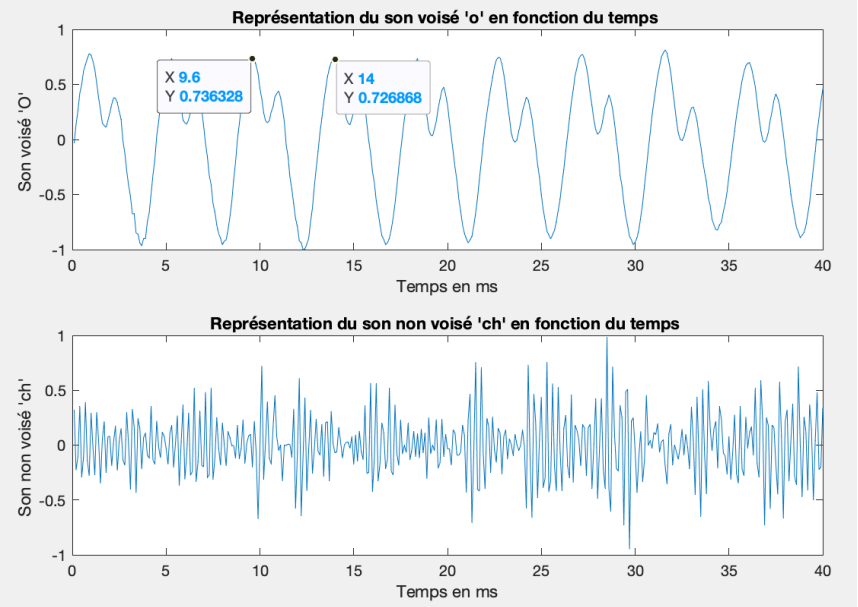
On remarque pour les trois figures que l'allure de l'estimateur non biaisé est proche de l'estimateur biaisé. Cependant, nous remarquons que plus le décalage p augmente, l'estimateur non biaisé s'éloigne grandement de l'estimateur biaisé.
En effet, la relation liant ces deux estimateurs est :

$$\text{Var}\,(\gamma_b(p,\omega)) = \left(\frac{N-p}{N}\right)^2 * \text{Var}\,(\gamma_{nb}(p,\omega))$$

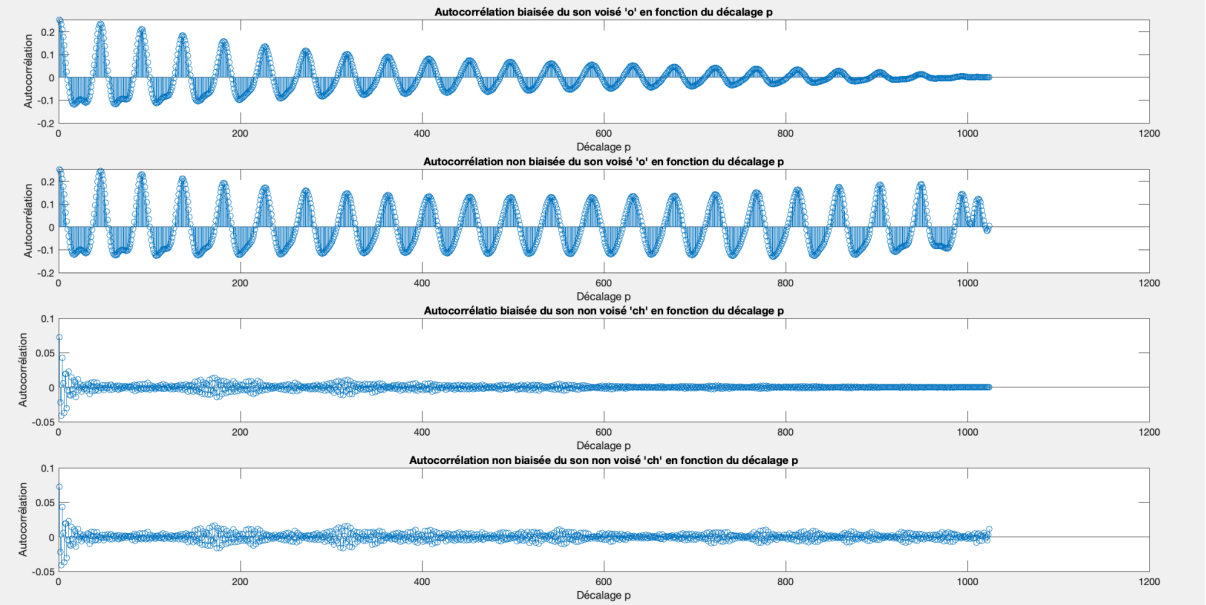
En faisant passer le facteur $\left(\frac{N-p}{N}\right)^2$ à droite, on remarque que la variance de l'autocorrélation augmente considérablement lorsque p augmente. Cela explique pourquoi l'autocorrélation subit de forte variation lorsque le décalage se rapproche de N.

4. Établissement d’un critère pour le caractère voisé

Pour des sons de paroles dont la fréquence est comprise entre 100Hz et 400Hz, les pseudo-périodes associées sont comprises entre 2,5ms et et 10ms.



On détermine une pseudo-période égale à 4,4ms. Cette valeur est bien comprise entre 2.5ms et 10ms.



On retrouve ici les résultats de nos hypothèses, le son voisé a une autocorrélation proche de celle du sinus et le son non voisé a une autocorrélation proche du bruit blanc.
Nous observons, en particulier, que l’autocorrélation biaisée du son non voisé subit une décroissance pour les décalages p croissant. Nous observons une diminution d’au moins 50% par rapport au décalage nul.

Nous avons donc la condition pour définir notre critère pour distinguer les sons voisés des sons non voisés: il suffit d'effectuer l'autocorrélation biaisée du signal à analyser et de vérifier que sa valeur diminue bien d'au moins 50% entre le décalage nul et les décalages suivants.

La partie de la séance I "Critère "voisé" ou "non voisé", nous avons seulement 4 échantillons au-dessus de 50% du max sur un total de N=1024 échantillons pour l'autocorrélation biaisée du son non voisé. Nous en avons 46 sur 1024 pour l'autocorrélation biaisée du son voisé. Nous allons étendre le critère crit=4 à crit=10 (ce qui correspond à 1% de N) dans notre cas pour prendre en compte les incertitudes.

De cette étude, nous pouvons implémenter notre fonction isVoiced qui retourne un booléen qui décrit si le signal d'entrée est voisé ou non. Notre fonction est vérifiée pour 4 sons différents (2 sons voisés et 2 sons non voisés).

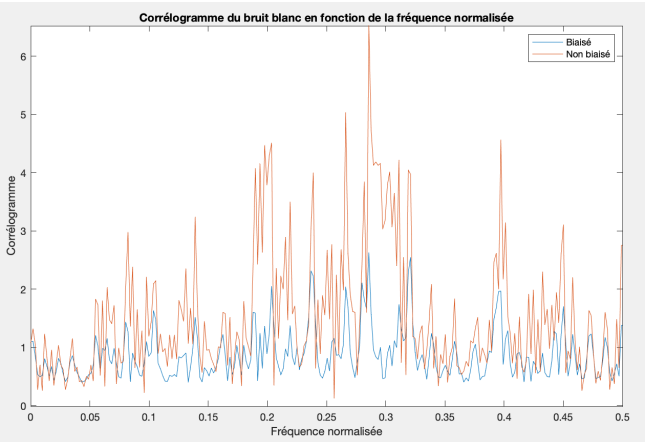
S2 Analyse spectrale

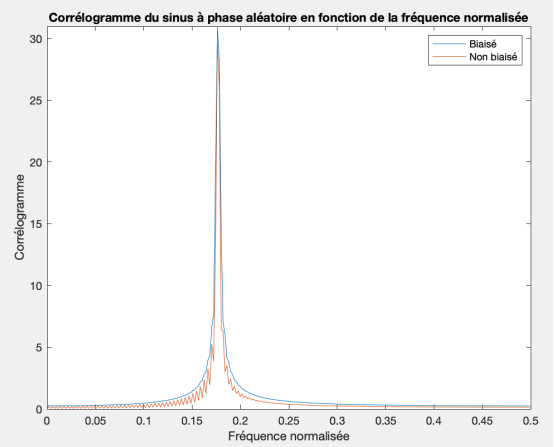
Objectifs de cette deuxième séance : estimation de la densité spectrale de puissance (DSP) d'un signal.

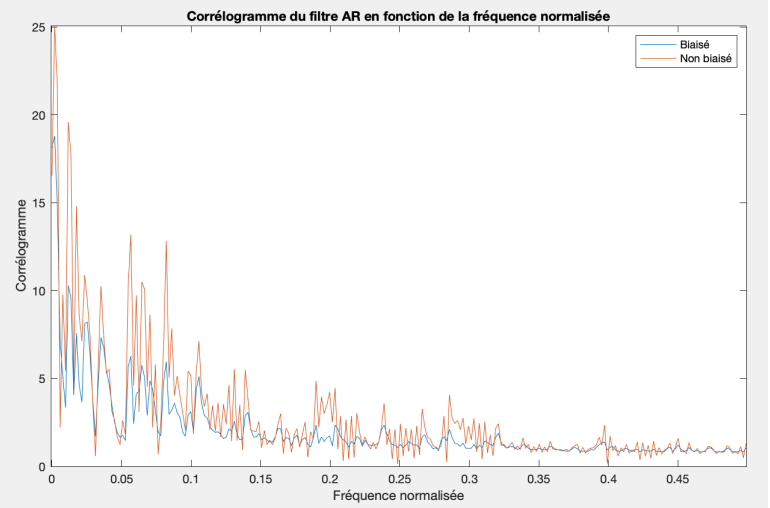
1. Caractérisation d'un processus aléatoire dans le domaine fréquentiel
2. Et du fenêtrage

La valeur minimale de Nfft est 2*N pour éviter un repliement du spectre.

On trace les corrélogrammes des trois signaux à partir de leurs autocorrélations biaisées et non biaisées.



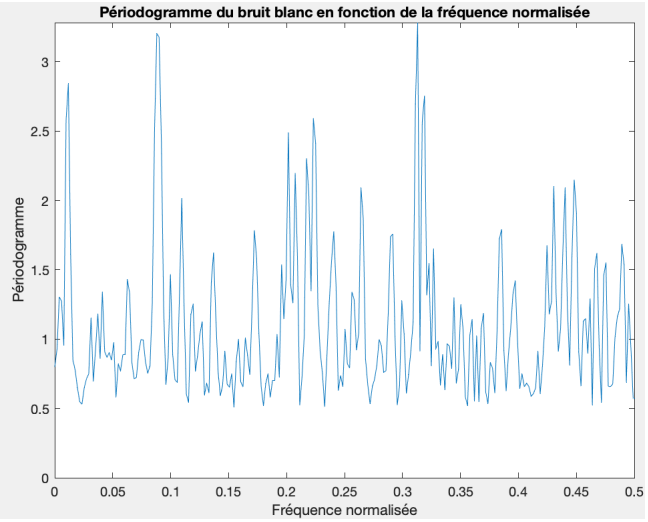


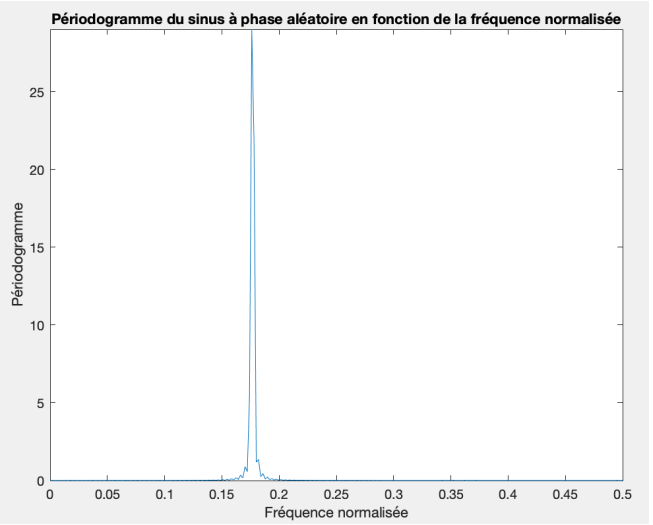


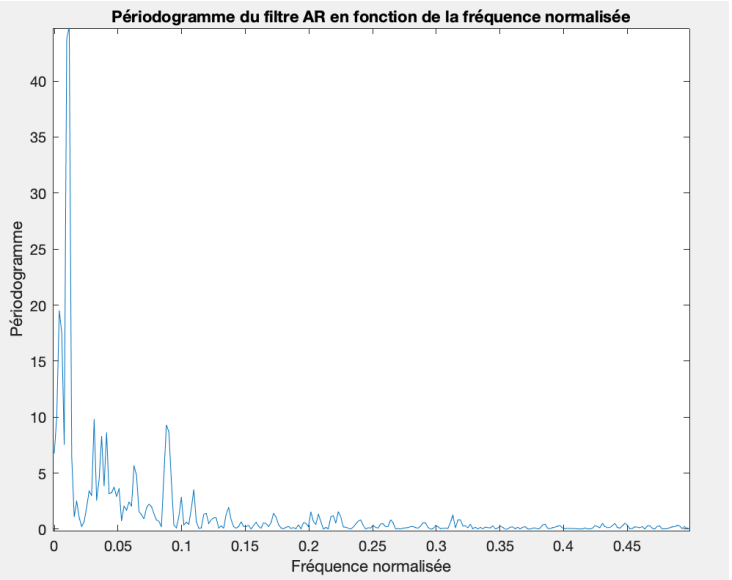
Les allures des corrélogrammes biaisés et non biaisés sont semblables
Nous retrouvons des résultats du cours:
-Pour le bruit blanc, le corrélogramme biaisé admet une variation d'amplitude plus petite que celle du corrélogramme biaisié. En effet, le corrélogramme biaisé est l'estimateur qui se rapproche le plus vers la DSP quand N augmente (ici DSP=0).
-Pour le sinus à phase aléatoire, nous retrouvons un sinus cardinal en $\nu 0$. Si on augmente le nombre d'échantillon, nous voyons que nous nous approchons d'un pic en $\nu 0$. Donc l'autocorrélation serait un signal sinusoïdale de fréquence $\nu 0$*fe. Ce sont des formes que nous retrouvons dans les parties précédentes.

Périodogrammes:

Nous pouvons aussi tracer les périodogrammes. Ils se calculent à partir des signaux initiaux et non plus des autocorrélations.







En comparant les périodogrammes et les corrélogrammes, nous observons que les allures des périodogrammes sont proches de celles des corrélogrammes issues de l'autocorrélation biaisée.

S3 Prédiction linéaire

Objectifs de cette deuxième séance : estimation de la densité spectrale de puissance (DSP) d'un signal.

1. Résolution des équations de Yule-Walker

De la relation de Yule-Walker découle la relation suivante :

$$R_X a_{opt} = p_x \quad (1)$$

avec $\mathbf{a_{opt}}$ la réponse impulsionnelle du filtre prédicteur :

$$a_{opt} = (a_0, a_1, \dots, a_{M-1})^T$$

tel que

$$X(n,w) = \sum_{k\geq 0} a_k X(n-k,w)$$

avec X(n,w) note signal stationnaire au 2e ordre.

Nous avons également $\mathbf{R_X}$ la matrice de covariance du vecteur $X_M(n,w) = (X(n,w), X(n-1,w), \dots, X(n-M+1,w))$:

$R_X =$

$$\begin{pmatrix} \gamma_X[0] & \gamma_X[-1] & \cdots & \gamma_X[-M+1] \\ \gamma_X[1] & \gamma_X[0] & \cdots & \gamma_X[-M+2] \\ \vdots & \vdots & \ddots & \vdots \\ \gamma_X[M-1] & \gamma_X[M-2] & \cdots & \gamma_X[0] \end{pmatrix}$$

Nous avons enfin $\mathbf{p_X}$ le vecteur des autocorrélations :

$$p_X = (\gamma_X[1], \gamma_X[2], \dots, \gamma_X[1])$$

Remarque : L'autocorrélation étant hermitienne et le signal que nous manipulons étant réel, nous avons $\gamma_X[p] = \gamma_X[-p]$. Ainsi R_X est une matrice symétrique.

Nous construisons le fichier **YuleWalkerSolver.m** qui contient la fonction permettant de résoudre l'équation (1). Cette fonction prend en paramètre l'autocorrélation théorique du signal à prédire (à décorréler) et retourne a_{opt} .

Nous y intégrons également le calcul de la puissance de l'erreur de prédiction $e_{opt}(n,w)$ définie par l'équation :

$$E(|e_{opt}(n,w)|^2) = \gamma_X[0] - p_X{}^T a_{opt}$$

Testons la fonction.

Test pour l'ordre k= 1 :

Pour tester la fonction à l'ordre k = 1, on utilise un filtre AR(1) : $H(z) = \frac{Z}{Z-a}$

avec N = 512 , $\sigma^2 = 0.1$ et a = 0.8.
En utilisant le résultat théorique de l'autocorrélation pour un filtre AR(1), on trouve a_1 = 0.8 .

L'équation de récurrence pour un filtre AR(1) est : X(n,w) = B(n,w) + a*X(n-1,w) .
Comme a = 0.8, nous trouvons bien a_1 = a .

La puissance de l'erreur de prédiction est égale à la variance e= 0.1 .

Test pour l'ordre 2 :

Pour tester la fonction à l'ordre k = 2, nous utilisons un filtre AR(2) que nous appliquons à un signal sinusoïdal à phase aléatoire :

$$x[n] = a\cos(2\pi\nu_0 n + \varphi) \text{ avec } \nu_0 = fe = 10000.1\text{Hz}$$

L'équation de récurrence du signal est donc la suivante :

$$X(n,w) = 2\cos(2\pi\nu_0) X(n-1,w) - X(n-2,w)$$

Nous nous attendons à trouver $a_1 = 2\cos(2\pi\nu_0) = 1.62 \quad et \quad a_2 = -1$

En appliquant la fonction à notre sinusoïde, nous trouvons a_1 = 1.618 et a_2 = -1 .
Les valeurs sont cohérentes avec la théorie.

La puissance de l'erreur de prédiction est de 1.10^{-16} donc quasiment 0 .

Lorsque l'ordre est supérieur à 1 et que l'on utilise le filtre AR(1), les coefficients a_k pour k>2 sont tous nuls car même avec l'application d'un ordre supérieur à 1, le filtre garde le comportement d'un AR(1) :

$$X(n,w) = B(n,w) + a_1 X(n-1,w) + 0 * X(n-2,w) + \ldots$$

Lorsque l'ordre est supérieur à 2 sur un signal sinusoïdal à phase aléatoire, les coefficients a_k pour k>3 devraient tous être nuls d'après l'équation de récurrence : X(n,w) = 2cos(2 Pi nu0)*X(n-1,w) - X(n-2,w) + 0*X(n-3,w) …
Matlab nous retourne des valeurs non nulles (pour k =3 : -0.33, 0.08, -1.21) de ces coefficients en affichant un warning. En effet, Matlab ne parvient pas à calculer a_{opt} car R_X n'est pas inversible. Nous ne pouvons donc pas procéder au calcul $a_{opt} = R_X^{-1} p_X$.

Ainsi notre fonction est valable théoriquement pour des filtres d'ordre 1 et 2.

2. Réalisations de processus connus

Dans cette partie, nous utilisons la même fonction que celle utilisée dans la partie précédente (**YuleWalkerSolver.m**) en remplaçant le paramètre d'autocorrélation théorique par son estimateur biaisé.

Effectuons de nouveau les tests.

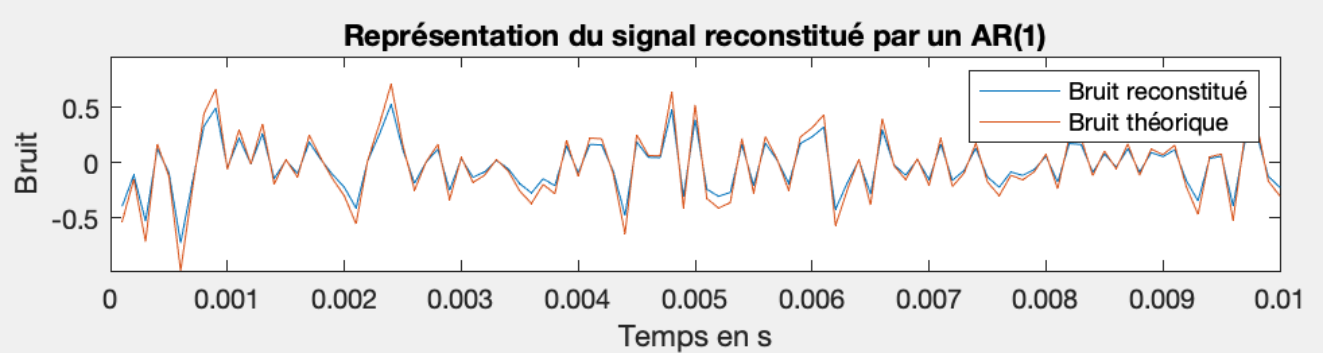
Test pour l'ordre k= 1 :

Nous utilisons toujours le filtre AR(1) : $H(z) = \frac{Z}{Z-a}$

avec N = 512 , $\sigma^2 = 0.1$ et a = 0.8.

En effectuant les calculs, nous obtenons a_1 = 0.82 et une puissance de l'erreur à 0.08.
Ces deux valeurs sont proches de ce que nous avons trouvé pour l'autocorrélation théorique (où a_1 = 0.8 et la puissance de l'erreur est égale à 0.1).

On peut maintenant reconstituer le bruit :



Nous pouvons remarquer que le bruit reconstitué présente la même allure que le sinus théorique.

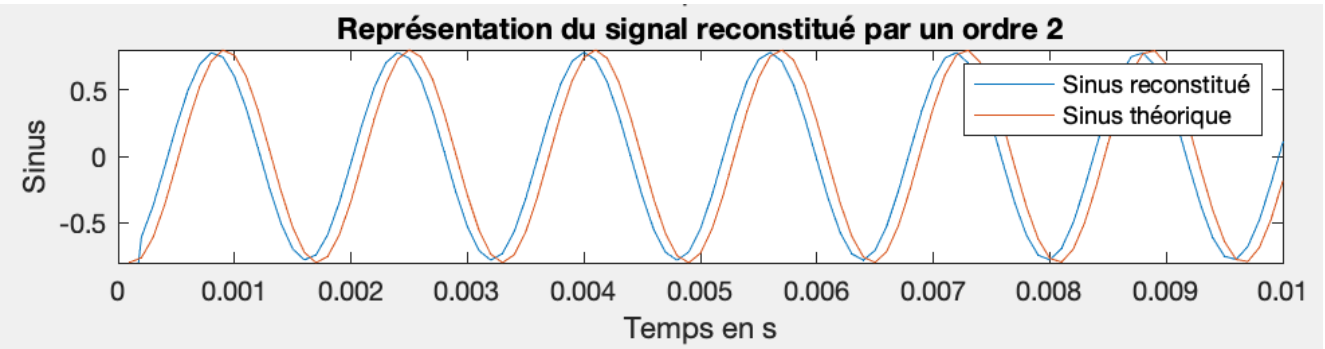
YuleWalkerSolver fonctionne pour un filtre d'ordre.

Test pour l'ordre 2 :

Toujours avec le même filtre que dans la partie précédente, nous obtenons cette fois a_1 = 1.81 et a_2 = - 0.96 . La puissance de l'erreur est de 0.003.

Dans la partie précédente, nous avions obtenu a_1 = 1.62, a_2 = -1 et une puissance de l'erreur proche de 0. Les résultats pour un ordre 2 sont cohérents avec la partie théorique.

On peut maintenant reconstituer le sinus :



Le sinus est bien représenté. Cependant, nous pouvons voir un déphasage entre les deux sinus.

YuleWalkerSolver fonctionne pour un filtre d'ordre 2.

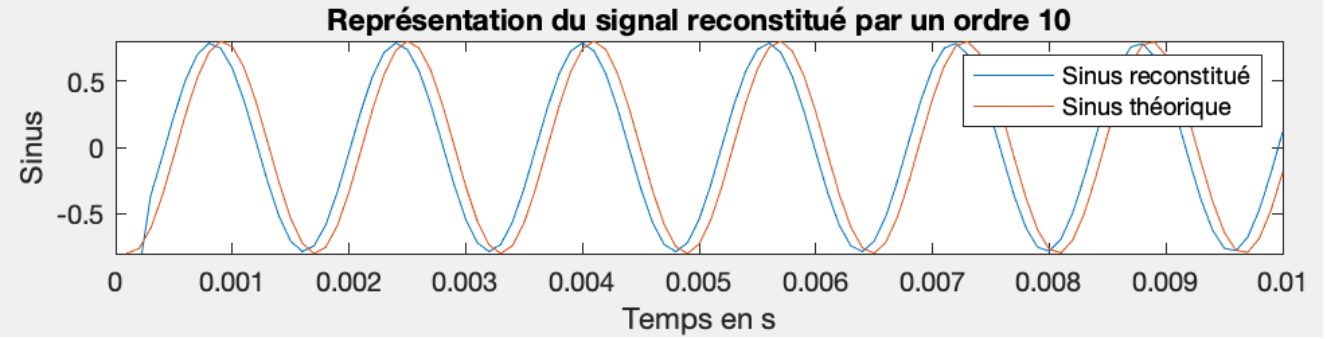
Nous pouvons toujours tester la reconstitution pour un ordre supérieur à 2 pour affiner notre reconstitution:

Test pour l'ordre K>2 :

Nous appliquons notre fonction pour M=10 à notre sinus et nous obtenons les valeurs des ak suivantes : 1.4898,-0.4710, -0.1660, -0.0585, -0.0206, -0.0071 -0.0022 0.0001 0.0024 -0.0039

Nous étions censés obtenir : 1.61, -1, 0, 0, 0,... Cependant, nous remarquons que plus k augmente, plus ak est proche de 0. On peut émettre l'hypothèse que les valeurs non nulles compensent les deux premières valeurs de ak.

Testons la reconstitution:



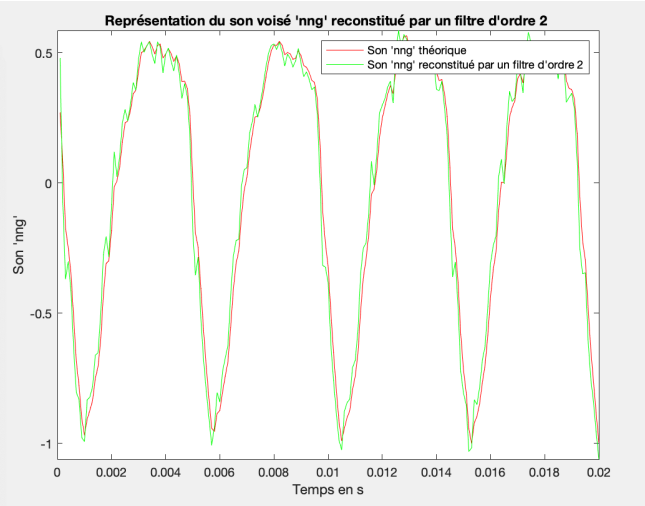
La sinus reconstitué est bien fidèle à notre sinus. L'allure est identique à celle du sinus reconstitué avec un ordre 2. Même si nous obtenons des valeurs des ak très différente de la théorie, la reconstitution est cependant fidèle. La variance reste également faible (e=0.0016).
Nous pouvons ainsi valider la fonction YuleWalkerSolver pour un ordre K>2 (du moins, jusqu'à l'ordre 10).

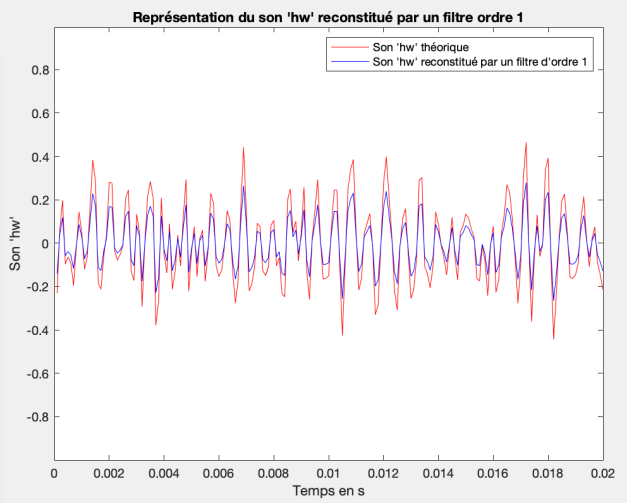
Remarque : plus l'ordre est grand, plus la reconstitution est fidèle à notre sinus de départ. Nous remarquons passé un certain ordre, la reconstitution est assez fine. Pour notre sinus, une reconstitution avec un filtre d'ordre 2 suffit.

3. Sons voisés et non voisés

Nous allons maintenant constituer les sons voisés et non voisés vus dans la première séance. Cette dernière nous dit qu'un son voisé présente une allure de sinus, et qu'un son non voisé présente une allure d'un bruit. Nous allons donc reconstituer le son non voisé à l'aide d'un filtre d'ordre 1 et le son voisé à l'aide d'un filtre d'ordre 2.

Nous appliquons notre programme sur le son voisé "nng" et le son non voisé "hw"





Nous observons pour les deux signaux que le signal reconstitué est très fidèle au signal réel. Notre programme de reconstitution du signal fonctionne pour les sons voisés et non voisés.

S4 Codage, décodage et effets sur la parole

Objectifs de cette deuxième séance : mettre en œuvre le codage/décodage du signal de parole par prédiction linéaire en traitant le signal par tronçon. On comparera les signaux réels et synthétiques avec les outils de traitement du signal (autocorrélation, spectre) et à l'oreille. En modifiant les caractéristiques analysées sur les tronçons, on pourra appliquer des effets sur la voix.

1. Analyse d'un tronçon

Nous allons implémenter la fonction qui permet d'analyser un tronçon de signal de N=256 échantillons avec la fonction BlockAnalysis.

Dans un premier temps, nous allons coder une fonction intermédiaire PitchDetector.m qui nous retourne la fréquence réduite d'un son voisé (ou -1 si le signal est non voisé).
Elle prend en entrée les informations du signal à analyser (autocorrélation Cx et fréquence d'échantillonnage Fe) ainsi que les bornes de fréquences Fmin et Fmax que l'on donne à notre signal.

Nous la testons pour notre son voisé "ooo" et non voisé "hw" et voici les fréquences réduites:

- υ = -1 pour le son non voisé "hw". C'est normal, c'est un son non voisé
- υ =0.0234 pour le son voisé "ooo". On avait auparavant déterminer une période de ce son T=4.4ms d'où f=227 Hz. Puisque la période d'échantillonnage est de 10 kHz, υ = 0.0227. Les deux υ sont du même ordre de grandeur. La différence peut être réduite si l'on augmente l'échantillonnage.

Nous allons maintenant implémenter le programme BlockAnalysis qui utilise les fonctions déjà codées précédemment et permet d'analyser un signal. On applique BlockAnalysis sur nos sons voisés et non voisés:

	Pitch	Sigma2	Aopt
Son voisé "ooo", M=2	0.0234	0.0014	1.89; -0.93
Son voisé "nng", M=2	0.0215	0.0027	1.77, -0.080
Son non voisé "ch", M=1	-1	0.0658	-0.3000
Son non voisé "hw", M=1	-1	0.0421	0.599

Les fréquences d'échantillonnages de tous les sons valent 10 kHz. Puisque pitch donne des informations sur le signal lorsqu'il est non voisé, nous pouvons ajouter dans le code la possibilité de déterminer M lui-même.
Les valeurs de variances sont cohérentes: elles sont beaucoup plus faibles pour les sons voisés car ce sont des sinus et plus élevées pour des bruits.

2. Synthèse d’un tronçon

Nous allons maintenant implémenter la fonction BlockSynthesis qui permet de synthétiser un tronçon de signal à partir des paramètres du filtre.

3. Traitement d’un signal et effets

Nous découpons notre signal en paquet de 1024 échantillons (c’est la taille des signaux voisés et non voisés que nous avons manipulés). Il faut ensuite appliquer BlockAnalysis puis BlockSynthesis pour récupérer les caractéristiques de notre audio (variance, voisé ou non, filtre) puis de le reconstituer à partir de ces dernières informations en concaténant les paquets côte à côte.